

TRAFFIC FLOW CONSIDERATIONS
IN NETWORK ASSIGNMENT MODEL

A THESIS

Presented To

The Faculty of the Division of Graduate
Studies and Research

By

Hsiao-Cheng David Yu

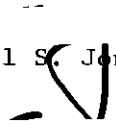
In Partial Fulfillment
of the Requirements for the Degree
Master of Science in
Operations Research

Georgia Institute of Technology

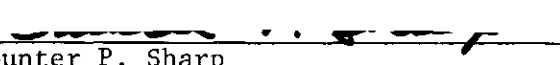
March, 1976

TRAFFIC FLOW CONSIDERATIONS
IN NETWORK ASSIGNMENT MODEL

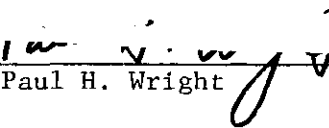
Approved:



Paul S. Jones, Chairman



Gunter P. Sharp



Paul H. Wright

Date approved by Chairman: 12 Mar 76

TO MY FATHER
ON HIS SEVENTIETH BIRTHDAY.

ACKNOWLEDGMENTS

I wish to express my deepest appreciation to my thesis advisor, Dr. P. S. Jones, who, a constant source of patience, understanding and encouragement, is always the person that lights the candle in the dark. His guidance and enthusiasm were indispensable in the completion of this work.

Appreciation is also extended to Dr. G. P. Sharp and Dr. P. H. Wright, for their serving as members of the reading committee, and their kind assistance and useful suggestions.

I offer lasting gratitude to my mother and father for their boundless love, trust and encouragement.

Also, I would like to thank Mr. James T. Baird, one of my best friends in this school, for his devotion of time to edit the manuscript. Special thanks are also due to Miss Laura Royston, a very nice lady to work with, for trying her best in typing in order to help me meet the deadline.

To all those who care about me, I owe my sincere gratitude.

TABLE OF CONTENTS

	Page
ACKNOWLEDGMENTS	iii
LIST OF TABLES	vii
LIST OF FIGURES	viii
SUMMARY	xi
CHAPTER	
I. INTRODUCTION	1
Background	1
Transportation Planning Process	1
Trip Generation	2
Trip Distribution	2
Modal Split	2
Traffic Assignment	2
Objectives	2
Realistic Traffic Assignment	3
Efficient Shortest Path Algorithm	3
Simplified Network	3
II. LITERATURE SURVEY	5
All-or-Nothing Assignment	5
Probabilistic Assignment	6
Reiterative All-or-Nothing Assignment	8
Algorithm Developed by Chicago Area Transportation Study (CATS)	9
Algorithm Developed by the U. S. Bureau of Public Roads (BPR)	10
Algorithm Developed by the Traffic Research Corp- oration	10
Mathematical Programming Models	12
Incremental Traffic Assignment	12
The Algorithm of Martin and Manheim	12
The Algorithm of Steel	13
III. DEVELOPMENT OF THE STEPWISE ASSIGNMENT PROBLEM	14
Rationale	16
Stepwise Assignment Procedure	19

TABLE OF CONTENTS (Cont.)

	Page
Experiments on a Small Network	20
IV. FORMULATION OF THE STEPWISE ASSIGNMENT ALGORITHM . . .	25
Introduction	55
Technique to Input Data	26
An Efficient Way to Input the Link Length Data . . .	26
An Efficient Method for Preparing the Travel Demand Data	30
Link Type Data	35
Technique to Assign Flow to the Links	35
The First Method	36
The Second Method	37
The Third Method	41
Technique to Update the Cost on Each Link	47
Technique to Find the Shortest Paths	50
V. SHORTEST PATH ALGORITHM	55
Introduction	55
Previous Research	56
The Algorithm of Floyd	57
The Algorithm of Dantzig	59
A New Shortest Path Algorithm	60
The First Algorithm	61
The Second Algorithm	65
The Third Algorithm	73
The Fourth Algorithm	75
Conclusions	91
VI. APPLICATION OF THE STEPWISE ASSIGNMENT MODEL TO DERIVE THE TRAFFIC DISTRIBUTION OF COLUMBUS-PHENIX CITY . . .	92
The Network	92
The Trip Demands	101
The Cost Functions	101
Cost on Nodes	101
Cost on Links	101
Results and Discussion of Results	102
Computation Time	105
Memory Requirements	105
Comparison of Flow Distribution for 1-Step, 5-Step and 10-Step Assignments	107
Discussion of the Trip Distribution Among the Three Bridges Connecting Columbus and Phenix City	111
Number of Iterations to Find the Shortest Paths . .	113

TABLE OF CONTENTS (Cont.)

	Page
Criticism of Assumptions	114
Comparison of the Results from 10-Step Assignment and the Columbus-Phenix City Transportation Study (CPTS)	115
Comparison of Congestion Conditions	115
Comparison of Travel Times	115
Stepwise Assignment with Diminishing Stepsizes	118
VII. CONCLUSIONS AND RECOMENDATIONS FOR FUTURE STUDY	120
Conclusions	120
Recomendations for Future Study	121
Improvement on the Network	121
Improvement on the Demand	121
Improvement on the Speed-Flow Relationships in the Link	121
Improvement on the Delay-Flow Relationships in the Node	121
Improvement on the Steps of Assignment	121
Improvement on the Shortest Path Finding Technique for the Stepwise Assignment Algorithm	122
APPENDIX	
I. SPEED-FLOW RELATIONSHIPS FOR MULTILANE ROADS	123
II. FORTRAN LISTING OF STEPWISE ASSIGNMENT ALGORITHM	127
III. THE DERIVATION OF THE TOTAL COST TO TRAVELING ON A PATH	134
BIBLIOGRAPHY	136

LIST OF TABLES

Table	Page
1. Path Used On Each Step In 5-Step Assignments and 10-Step Assignments	24
2. Route Description And Node Sequences Representing The Major Arterial Routes Of Columbus-Phenix City . . .	97
3. Type Of Roads, Number Of Lanes And Name Of Speed-Flow Functions Of Each Category Of Roads	99
4. Relationship Among Speed, Spacing, Concentrations And Flow Rate For Single Lane Uninterrupted Flow . . .	104
5. Number Of Links In Each Range Of Flow For 1-Step, 5-Step and 10-Step Assignment	107
6. Flow Distribution Among Links (26,12), (25,13) and (35,24) Under 1-Step, 5-Step And 10-Step Assignment . .	112
7. Flow Distribution Among Links (12,26), (13,25) and (24,35) Under 1-Step, 5-Step And 10-Step Assignment . .	113
8. The Number Of Iterations Needed To Find The Shortest Paths For Each Step Of The 1-Step, 5-Step And 10-Step Assignment	114
9. Comparison Of Travel Time On Some Uncongested Links Obtained From The 10-Step Assignment And CPTS	116
10. Number Of Links In Each Range Of Flow For 10-Speed Equal Stepsize Assignments And 11-Step Diminishing Stepsize Assignments	118

LIST OF FIGURES

Figure	Page
1. Network For The Experiments	20
2. Trip Matrix For The Experiments	21
3. Flow Distribution On The Experiment Network By 5-Step, 10-Step And 20-Step Assignments	23
4. Relation Among Arrays M1, M2 and MD	29
5. Flow Chart Of The Construction Of The Distance Matrix From Arrays M1, M2 And MD	31
6. Relation Among Arrays M4, M5 and M6	33
7. Flow Chart Of The Obtaining Of The Demand Information From Arrays M4, M5 and M6	34
8. Array AF With Elements Representing Link Flows	40
9. A Shortest Path From Node i To Node j Identified By Back Nodes	41
10. Back Node Matrix And New Back Node Matrix	44
11. Flow Chart Of The Construction Of A New MN Array	51
12. Flow Chart Of The Assignment Of Demand to The Links On The Shortest Path	52
13. A Network Contains A Link Which Is Common To Many Shortest Paths	47
14. Flow Chart Of The Conversion Of Flow Distribution From A One-Dimensional Array AF To A Two-Dimensional Array	53
15. Flow Chart Of The Updating Of Cost Matrix AA Based On The Flow Distribution Stored In A One-Dimensional AF Array	54
16. Flow Chart Of The Floyd's Algorithm	58

LIST OF FIGURES (Cont.)

Figure	Page
17. Flow Chart Of The First New Shortest Path Algorithm .	63
18. Flow Chart Of The Second New Shortest Path Algorithm	66
19. Flow Chart Of The Third New Shortest Path Algorithm	74
20. Proposed Ideal Network	76
21. The Positions Of The Node Pairs In The Examining Sequence Of All Node Pairs	77
22. Sequence Of Node Pairs Being Examined (I)	83
23. Sequence Of Node Pairs Being Examined (II)	85
24. Sequence Of Node Pairs Being Examined (III)	86
25. Sequence Of Node Pairs Being Examined (IV)	88
26. Flow Chart Of The Fourth New Shortest Path Algorithm .	90
27. Descending And Ascending Sequences Of The Node Numbers In A Shortest Path	89
28. The Aggregated Network Of Columbus-Phenix City	94
29. Major Arterial Routes Of Columbus-Phenix City	95
30. Major Arterial Routes Represented By The Networks	96
31. Type Of Roads That Each Link Represents	100
32. Delay On Intersection For 10% Right - 10% Left Turns	103
33. Speed Versus Flow Rate For Single-Lane Uninterrupted Flow	106
34. Speed Versus Concentration For Single-Lane Unin- terrupted Flow	106

LIST OF FIGURES (Cont.)

Figure		Page
35.	Flow Rate Versus Concentration For Single Lane Uninterrupted Flow	106
36.	Flow Distribution Of 1-Step Assignment	108
37.	Flow Distribution Of 5-Step Assignment	109
38.	Flow Distribution of 10-Step Assignment	110
39.	Flow Distribution of 11-Step Assignment	119

SUMMARY

This study has been stimulated by an interest in the algorithmic processes underlying transportation planning. The approach now being followed in most transportation planning requires enormous amounts of input data and characteristically requires long and expensive computer time to process a single transportation alternative. Three areas that appear to offer promise for improvement are probed:

1. A more realistic assignment of traffic to a transportation network,
2. A more efficient method for calculating the shortest path between all of the nodal pairs of a network, and
3. A method for approximating complex networks with much simpler networks.

Attention is focused on the shortcomings of existing algorithms as a guide to identifying useful changes. The stepwise assignment approach which was pursued seems very promising with respect to reflecting the user's behavior.

Combining some of the ideas of tree building and matrix algorithm, a new shortest path algorithm is devised. This new algorithm is likely to be more efficient than other matrix algorithms because of special features (1) a method of node numbering, (2) a method of alternating searching sequence, to reduce the number of iterations required to reach the solution; and (3) technique for considering a set of intermediate nodes made up only of the nodes adjacent to the destination node.

The stepwise assignment model has been applied to derive the traffic distribution of Columbus, Georgia and Phenix City, Alabama. Using an aggregated network of 97 nodes and 342 one-way links. Fairly useful results are obtained from the simplified network as compared with Columbus-Phenix City Transportation Study, in which a network of 440 nodes and 3050 links are used.

CHAPTER I

INTRODUCTION

Background

This study has been stimulated by an interest in the algorithmic processes underlying transportation planning as it is now practiced. Planning agencies now use large packages of computer programs that develop estimates of travel over specific routes in a complex transportation network. These computer programs require enormous amounts of input data and characteristically require long and expensive computer time to process a single transportation alternative.

Transportation Planning Process

To place the work presented here in proper context, it is appropriate to briefly review the transportation planning process as it is customarily performed. The process consists of four steps, each of which is generally a computer program.

The process begins when the study area has been divided into a series of geographical zones, each of which is relatively homogeneous. All of the residents of a zone typically bear similar relationships to a transportation network under study, such that they would enter the network at the same point. In typical urban transportation studies, zones may comprise less than a square block in the city center and only a few square blocks in outlying areas. Demographic data are collected for each zone, including population, employment, commercial activity, income,

education, automobile ownership and other factors. Given the zone structure and the corresponding demography, the steps of the analysis are:

1. Trip Generation - Trip origins and destinations are computed for each zone on the basis of the demographic data. The calculation is adjusted so that the number of originating and terminating trips are the same.
2. Trip Distribution - Origins and destinations are matched by a gravity model or opportunity model to reflect actual travel desires.
3. Modal Split - Trips are assigned to the different available services on the basis of cost, travel time and other factors. Thereafter, the processing sequence is similar for each modal segment.
4. Traffic Assignment - The traffic represented by each origin-destination pair is assigned to the shortest or least cost route. The aggregate of all origin-destination pairs constitutes an assignment of all traffic to the links of the network.

When the above steps are complete, aggregated travel time, congestion and other factors can be used to judge the quality of the network under study.

Objectives

It is not the purpose of this study to revise and improve a complete computer package. Rather, this study probes three areas that appear to offer promise for improvement. These are:

1. A more realistic assignment of traffic to a transportation network;
2. A more efficient method for calculating the shortest path between all of the nodal pairs of a network; and
3. A method for approximating complex networks with much simpler networks.

Realistic Traffic Assignment

Because of its interest here, the traffic assignment problem warrants a more detailed description. Traffic assignment begins with a demand matrix that lists the trip volumes between all origin destination pairs. Given this demand information, the traffic assignment problem is to assign the traffic flows to the links of the network in a manner that closely represents the users' behavior. When considered in the aggregate, then, the assignments reasonably reflect or predict the real world traffic distribution.

Efficient Shortest Path Algorithm

A key to the computational efficiency of any traffic assignment is the ability to quickly and efficiently compute the shortest path between all of the origins and destinations of interest. An efficient shortest path algorithm is particularly critical for stepwise assignment, because a new set of shortest paths needs to be computed for each step.

Simplified Network

For several years, the transportation planning community has been seeking an adequate method for approximating the complex traffic analysis procedure. Several sketchy planning techniques have been proposed that approximate a complex network with a simple network for the purpose of

preliminary planning. The stepwise assignment procedure developed here offers a potentially useful approach to preliminary planning. In this approach, the more realistic assignment of traffic to the network compensates in part for the less detailed network. The result is a procedure that yields fairly useful results from much simplified input data.

CHAPTER II

TRAFFIC ASSIGNMENT ALGORITHMS

Large numbers of traffic assignment algorithms have been published in the literature. A number of these have been applied to transportation analysis computer programs and used in the investigation of urban transportation problems. The more widely used algorithms fall into five categories:

1. All-or-nothing assignment
2. Probabilistic assignment
3. Reiterative all-or-nothing assignment
4. Mathematical programming models
5. Incremental traffic assignment

Each of these classes of algorithms is discussed briefly below. Attention is focused on the shortcomings of existing algorithms as a guide to identifying useful changes.

All-Or-Nothing Assignment

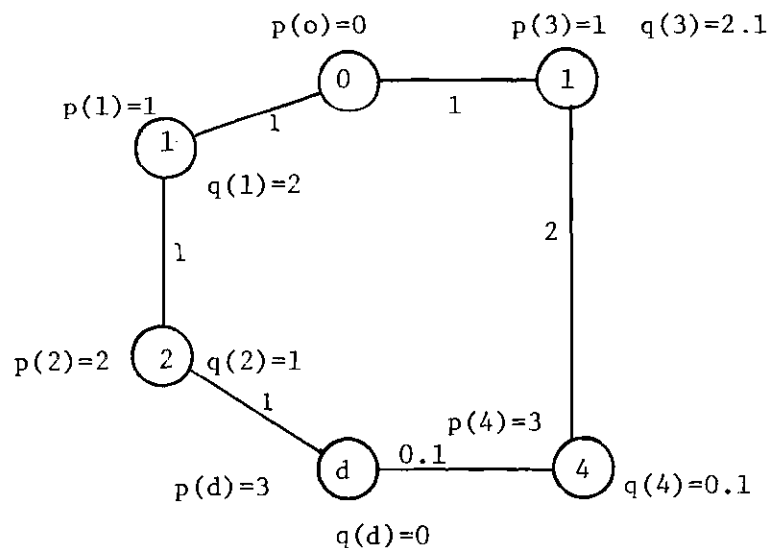
All-or-nothing assignment is a single path assignment. All trips between a fixed origin and destination are assigned to the links that constitute a single shortest connecting path. Since this algorithm can not consider the effect of capacity restraints, some links may be loaded beyond their capacity, while other links may have no flow because they are not included in any shortest path. The all-or-nothing assignment tends to offer unbalanced flow assignments. It accentuates congestion on

particularly attractive routes and does not reflect the efforts of individual travelers to avoid congestion.

Probabilistic Assignment

Dial's [5] probabilistic model assumes that there is random variation in the route selection among individual tripmakers. Without explicitly enumerating the different paths, trips are assigned to all reasonable paths simultaneously under certain probability assumptions. This algorithm progressively selects reasonable paths which always get further away from the origin and closer to the destination. The model has three principal drawbacks:

1. The set of reasonable paths is not complete enough to include all significant paths. This can be illustrated by an example of parallel paths, one of which is not included in the set of reasonable paths.



$p(i)$ = shortest path distance from origin node 0 to intermediate node i .

$q(i)$ = shortest path distance from intermediate node i to the destination node d .

$$\therefore p(4) = 3 \neq p(d) = 3$$

Path 0-3-4-d does not get further away from the origin and hence does not meet the criterion for a reasonable path. But path 0-3-4-d does move in the direction of the destination and will likely be randomly chosen by some trip makers. In fact, if for any path from 0 to d , the sum of the arcs excluding the last arc which is connected to d , is greater than or equal to the shortest path from 0 to d , then this path will not be included in Dial's set of reasonable paths.

2. The algorithm is not capacity restricted. The diversion probability is dependent only on the distance difference between the route being considered and the shortest route. $(e^{\theta(t^* - t)})$. The difference is only valid when the flow on each arc is close to the flow in the network from which times were estimated. Thus, the estimated arc travel times may be invalid when substantial network changes are introduced. It does not seem reasonable that the user supplied parameter θ for the diversion curve can be interpreted absolutely. Instead, adjustments are needed to reflect relative driver behavior.

Dial suggests that the probabilistic assignment model could be used to solve capacity restraint problems by applying an incremental

loading technique. If so, the algorithm is likely to lose its nice feature of being able to simultaneously assign trips to all reasonable paths. This could be a costly change because the probabilistic assignment algorithm takes more computer time than an ordinary stepwise assignment algorithm needs for each step.

3. The algorithm assumes that most trip makers are not aware of their best route choice; hence, they will tend to make route selections that are probabilistically distributed. This lack of concern is only present when the path times among alternatives are not very different. This situation is true in the examples of a downtown grid system that were used throughout Dial's paper. Many situations can be envisioned where the indifference criterion is not true. Taking a broader, or macro, point of view, nodes and links can be aggregated to form a condensed network in which nodes are critical intersections and links are significant arteries. In this simplified network, it is decisions that concern the main passageway that matter, not decisions about where to turn. Typically, there are not many alternative routes from which daily routine trip makers may choose, unless they seek detours to avoid congestion.

Reiterative All-Or-Nothing Assignment

The reiterative all-or-nothing algorithms have been most widely used in transportation planning. There are three important versions of this algorithm.

The first was developed as part of the Chicago Area Transportation

Study (CATS)^[2]. The second algorithm, developed by the U. S. Bureau of Public Roads (BPR) and incorporated into the "Highway Package," has received the greatest amount of use. The third algorithm, developed by the Traffic Research Corporation^[9, 10], goes back to the trip distribution phase and considers the impact of changes in travel time on trip distribution.

Algorithm Developed By Chicago Area Transportation Study (CATS)

The CATS assignment algorithm is as follows:

1. Randomly select one zone from the possible loading zones.
2. Determine the minimum path tree from the selected zone to all other zones.
3. Assign all trips from the selected zone to the minimum path defined by the tree.
4. Update the network with the new travel times calculated for the links in the minimum path tree according to a relationship between speed and traffic volume.
5. Repeat the procedure with the random selection of one of the remaining loading zones.

The basic drawback to this model is that the assignment is dependent on the sequence in which the different zone pairs are selected. The pairs selected early in the procedure are assigned to their shortest paths while the others may be assigned to the second or third shortest path because travel time has increased or flow has reached a link's capacity. For example, a link, which is common to the shortest paths of several zone pairs may be occupied by travelers from only a part of the origins of the

zone pairs. None of the travelers from some of the other zone pairs will appear on this link. The manner in which paths are selected to include some node pairs and exclude others is not realistic.

Algorithm Developed By The U. S. Bureau Of Public Roads (BPR)

The procedure of the UPR Traffic assignment algorithm is as follows:

1. Load the network using an all-or-nothing technique.
2. Determine the volume-capacity ratio for each link.
3. Revise link speed from a function relating volume-capacity ratio to speed.
4. Repeat the all-or-nothing assignment using the revised link speeds.
5. Repeat the revision of the link speeds and all-or-nothing assignment until the speeds at the beginning of an assignment approximately equal the speeds revised after the assignment.

This algorithm tries to scatter travelers from different zone pairs to different shortest paths but does not guarantee convergence. In practice, it tends to be stopped before a reasonable balance is achieved on critical congested links. The shortest path for each zone pair is switched from one to another, but eventually all flows are still assigned to a single path. The algorithm has most of the drawbacks of all-or-nothing assignment.

Algorithm Developed By The Traffic Research Corporation (TRC)

The TRC has developed a traffic model incorporating both the trip distribution and assignment phase of transportation planning, and proceeds as follows:

1. Compute initial set of minimum path trees based on an unloaded network.
2. Using a gravity model, compute the trip distribution from the travel time on minimum paths.
3. Assign trips to the network using an all-or-nothing technique.
4. Revise link travel times from the assigned link volume.
5. Compute minimum path trees based on the revised link travel time.
6. Repeat 2.
7. Assign trips to the original and revised minimum paths in proportion to the travel times over the two routes.
8. Repeat 4, 5 and 6.
9. Assign to the first, second and third revised minimum paths in proportion to the travel time over these routes.
10. Repeat the entire procedure until the desired number of iterations is reached. This algorithm tries to reflect how a traveler's choice of destinations might be affected by traffic congestion. It is an interesting approach, but does not guarantee convergence. In a typical application, only two or three iterations are performed.

The assignment part of this algorithm, which assigns flow to both the shortest path and the revised shortest path is an improvement over other reiterative all-or-nothing assignments. The flows are assigned in proportion to the travel time over these shortest paths. One problem arises from the definition of the travel time over a congested link, while another is that the proportion of travel time may be too high that each

time most of the travel flow on congested links are all assigned to the uncongested link to cause new congestion owing to "all" or "nothing" assignment.

All three assignment algorithms use the all-or-nothing technique to load a network, then use the results of the loading to modify the link times. These new link times are then used for a subsequent all-or-nothing loading in which presumably some new "shortest" paths are uncovered.

Mathematical Programming Models

The simplifying assumption of system-wide optimal user behavior allows the problem to be attacked with a variety of optimization techniques. These techniques usually are aimed at minimizing a function of the total travel time subject to flow conservation constraints and volume-speed relations. Charnes and Cooper^[1] have developed linear programming solutions. Yang and Snell^[15] formulate a non-linear equilibrium problem with fixed demand and develop a solution algorithm based on the maximum principle of Pontryagin. Tomlin^[13] formulates a quadratic programming problem.

It is the prohibitive solution cost that precludes most the mathematical programming approaches from solving real-sized problems.

Incremental Traffic Assignment

B. V. Martin and M. L. Manheim^[11] as well as M. A. Steel^[12] have proposed traffic assignment algorithms involving an incremental loading technique.

The Algorithm Of Martin And Manheim

Martin's procedure consists of five phases:

1. Random selection of a zone pair from all loading zones.
2. Determination of the minimum time path for this zone pair.
3. Use of a generation rate characteristic to determine the potential volume to be assigned for this zone pair.
4. Addition of a small increment of the potential volume to the minimum path.
5. Use of a volume-delay characteristic to update the travel time on the links in the minimum path due to the increase in volume.

The procedure is repeated with each zone pair picked randomly from the table of available zone pairs. As a zone pair become fully assigned, its entry is removed from the table of available zone pairs; therefore, the assignment is complete when the table is empty.

A generation rate characteristic indicates the percentage of the potential interzonal volume that will be realized as a function of the unit travel time between zone pairs. This allows the demand to be adjusted under different supply conditions. The problem is that neither theoretical formulation nor any empirical data are available at the present time. The algorithm is dependent on the sequence in which the different zone pairs are selected. Besides, after some zone pair has been selected, it still may be selected again, even when some other zone pairs have not yet been selected. It is not economical to revise the link travel time after each small increment assigned by one zone pair. It is also unrealistic to expect any volume-delay characteristic to be sensitive to small increments.

The Algorithm Of Steel

Steel's procedure is as follows:

1. Set all road speeds to maximum value (minimum cost).
2. Calculate minimum cost routes.
3. Assign l_i % of all traffic to these routes.
4. Calculate new road speeds from road traffic volumns.
5. Remove R_i % of all traffic from the road system.
6. Repeat step 2 till 100% of the trips are assigned.

The most obscure idea in this model is "the removal of flows from all roads in the network after each adjstment of the speeds"^[12]. There is no criterion for determing how much of the flow should be removed from the network. If the take-off percentage were set by the user, as suggested by Steel, "the most beneficial take-off percentage was assumed to be about one-third of the assigning percentage for the same iteration." Then we would be doing no more than assigning $(l_i - R_i)$ % of the total flow in each iteration. If the take-off percentage were calculated from, say, the difference between the previous road speeds and the new road speeds after assignment, then the following questions arise: "What is the criteria?" and "Of what significance is this?".

After surveying these classes of algorithms we may summarize as follows:

1. All-or-nothing assignment is an unbalanced, oversimplified, infinite capacity assignment.
2. In probabilistic assignment, the diversion probability for flow distribution on the routes is difficult to determine realistically.

3. The CATS reiterative all-or-nothing assignment has unequal opportunity for each traveler to use the network.
4. The UPR algorithm does not guarantee convergence. It is not very promising to balance traffic on congested links.
5. Traffic Research Corporation's algorithm assign flow to the shortest and the revised shortest route in proportion to the travel time. It is an interesting approach but it does not guarantee convergence.
6. Mathematical programming methods are not able to handle large problems.
7. Martin's incremental traffic assignment is far from economical. It updates the travel time and finds the shortest path for another zone pair each time a portion of demand for one zone pair is assigned to its shortest path.
8. Steel's procedure is very reasonable, except that it has a rather obscure removal process.

We therefore wish to find an improved assignment model in which:

1. Capacity is restricted on each link.
2. Diversion criteria are based on each traveler's behavioral route choice.
3. Each user has an equal opportunity to use uncongested links.
4. Computations are efficient.
5. A reasonable philosophical approach is taken.

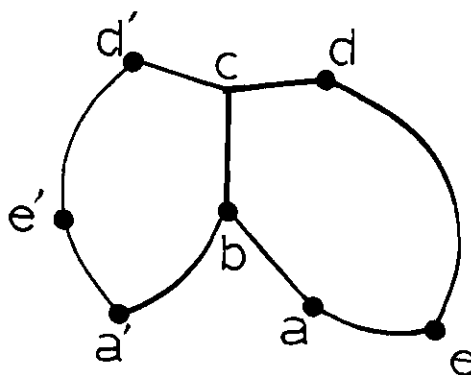
The stepwise assignment approach seems very promising with respect to these advantages and has been pursued here.

CHAPTER III

DEVELOPMENT OF THE STEPWISE ASSIGNMENT PROBLEM

Rationale

The stepwise assignment technique decomposes travel demand for a given time period into a set of incremental demands. The stepwise assignment procedure divides the travel between each origin-destination pair into a series of groups, or steps. When the assignment process begins, it treats the network as though it is empty. The groups or steps for each origin-destination pair are successively assigned. Each group selects its route through the network on the basis of a least travel time criterion. The travel time criterion is applied at the time that the group's travel is assigned to the network. In this fashion each group selects a route as though it knew the system state at the time of its departure. In practice, it is not reasonable to modify the system state after each group is assigned; therefore, after an entire step has been assigned, a picture is taken of the flow distribution that has been made up to that point and the travel time is adjusted on each network link to reflect the impact of the volume of traffic that has been assigned. The next step of travelers will then select the shortest path for their journeys on the basis of the revised travel times. The following example will illustrate the principle of route diversion.



x^{ij} is the travel demand from i to j .

f_{ij} is the travel time versus traffic flow relationship on link (i,j) .

x^{ad} and $x^{a'd'}$ are travel demands from a to d and from a' to d' .

Before any travel has been assigned, if

$$f_{ab}(0) + f_{bc}(0) + f_{cd}(0) < f_{ae}(0) + f_{ed}(0)$$

$$\text{and } f_{a'b}(0) + f_{bc}(0) + f_{cd'}(0) < f_{a'e'}(0) + f_{e'd'}(0)$$

then $abcd$ and $a'bcd'$ we know are the two shortest paths for the trips of interest. Initially, a portion of x^{ad} and a portion of $x^{a'd'}$ -- say ten percent -- are assigned through $abcd$ and $a'bcd'$. This results in a flow of $\frac{x^{ad} + x^{a'd'}}{10}$ on link bc . If the relations are

$$f_{ab}\left(\frac{x^{ad}}{10}\right) + f_{bc}\left(\frac{x^{ad} + x^{a'd'}}{10}\right) + f_{cd}\left(\frac{x^{ad}}{10}\right) < f_{ae}(0) + f_{ed}(0)$$

$$\text{and } f_{a'b}\left(\frac{x^{a'd'}}{10}\right) + f_{bc}\left(\frac{x^{ad} + x^{a'd'}}{10}\right) + f_{cd'}\left(\frac{x^{a'd'}}{10}\right) < f_{de'}(0) + f_{e'd'}(0)$$

then $abcd$ and $a'bcd'$ are still the shortest paths for the demand from a to d and demand from a' to d' . An additional $\frac{x^{ad}}{10}$ and $\frac{x^{a'd'}}{10}$ will be assigned through $abcd$ and $a'bcd'$. This results in a flow of $\frac{2x^{ad} + 2x^{a'd'}}{10}$ on link bc . Eventually, as the traffic on link bc increases and approaches the links' capacity, congestion will cause the travel time to increase rapidly.

At some step m , if

$$f_{ab} \frac{mx^{ad}}{10} + f_{bc} \frac{mx^{ad} + mx^{a'd'}}{10} + f_{cd} \frac{mx^{ad}}{10} > f_{ae}(0) + f_{ed}(0)$$

$$\text{or } f_{a'b} \frac{mx^{a'd'}}{10} + f_{bc} \frac{mx^{ad} + mx^{a'd'}}{10} + f_{cd'} \frac{mx^{a'd'}}{10} > f_{a'e'}(0) + f_{e'd'}(0)$$

then the next increments $\frac{x^{ad}}{10}$ of $\frac{x^{a'd'}}{10}$ will shift to the new path aed or $a'e'd'$. Since the travel increases on aed or $a'e'd'$, the travel time over aed or $a'e'd'$ may be greater than for path $abcd$ or path $a'bcd'$. Some traffic may be assigned to link bc again.

It is the principle of the stepwise assignment model to assign each trip to the shortest path available at the time of each user's decision. Whenever some link is common to more than one shortest path, each origin-destination pair has an equal opportunity to load this link.

After the travel times have increased, the users who have other shortest paths available may choose these alternative routes.

The postulate put forth here is that the trip decision made by each traveler is based on his knowledge of the system at the time that he chooses his route. This knowledge is based primarily on his previous

experience in the network. It is not unrealistic to assume that many trip makers are aware of the shortest paths available at different times of the day because they are making familiar trips at familiar times of the day. These travelers will naturally be alert to the problems of certain routes that may cause trouble.

The assumed sensitivity to even small differences in travel time between alternative routes at each step is not to be criticized. This overloading on the shortest path and the underloading on the slightly longer path is going to be rectified in each of the succeeding steps.

As an example, consider two routes, A and B, that connect the same origin and destination. If A is slightly shorter than B, then the stepwise assignment algorithm will assign the first portion of the demand to route A. After the first assignment, if the travel time on A is now higher than that on B, the next portion of demand will be assigned to B. As the assignment process continues. A relatively even flow will be assigned to the two routes. This balanced distribution is consistent with the distribution in a real situation and is a phenomenon which all-or-nothing assignment can never reflect.

Stepwise Assignment Procedure

The procedure for the stepwise assignment algorithm is as follows:

1. Determine the number of steps desired to load the total demand on the network.
2. Divide the demand matrix into as many submatrices as there are number of steps.
3. Based on the assigned travel speed on each link, calculate the

shortest path (shortest travel time) between all node pairs for which a travel demand exists.

4. Assign the first submatrix of the demand to the paths with the shortest travel time.

5. After each assignment, update the travel time on each link based on the expected speed for the flow rate on the links from all previous assignments. (Effective application of the stepwise assignment algorithm depends on the availability of reliable speed-flow relationships for the different links of the network.)

6. Recalculate the shortest paths between all node pairs for which travel demand remains.

7. Assign another submatrix of demand to each link of the shortest paths.

8. Repeat 5, 6 and 7 until all demand has been assigned to the network.

Experiments On A Small Network

The following example will illustrate the use of the stepwise assignment model on a small network.

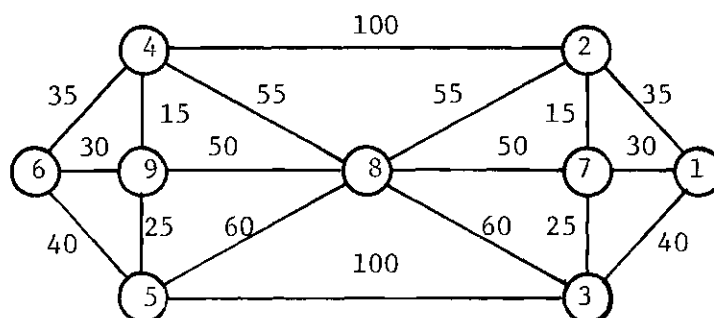


Figure 1. Network for the Experiment.

The example network consisting of 9 nodes and 36 links is illustrated in Figure 1. Four nodes (2, 3, 4 and 5) are origins and destinations. The trip matrix is shown in Figure 2.

	2	3	4	4
2	0	2000	2000	1000
3	200	0	1000	2000
4	200	100	0	1000
5	100	200	100	0

Figure 2. Trip Matrix For The Experiment.

The cost function used on each link is:

$$C_{ij}(x_{ij}) = 0.001 * x_{ij}^2 + l_{ij}$$

$$C_{ij} = \text{Cost on Link } (i,j)$$

$$x_{ij} = \text{Flow on Link } (i,j)$$

$$l_{ij} = \text{Length of Link } (i,j)$$

The flow distribution on the links with the 5 step assignments, the 10 step assignments and 20 step assignments are shown in Figure 3. The flow amount on links (2-4), (8.5) and (5-9) are increasing. The flow

amount on links (2-8), (8-4) and (8-9) are decreasing. The flow amount on links (7-2), (2-7), (3-7), (7-3), (3-8), (5-9) and (9.4) are making small adjustments. However, differences on these links are sufficient to suggest that the five step, ten step and twenty step solutions are significantly different. The paths to which flow is assigned at each step for every node pair are shown in Table 1 for both a five step assignment and a ten step assignment. Examination of Table 1 reveals that important differences in assignment occurs for origin-destination pairs (2-3), (2-4), (2-5), (3-2), (3-4), (3-5), and (4-5). These difference are substantial enough to account for the flow difference of Figure 3.

The total cost of $\sum_{ij} C_{ij} X_{ij}$ for this system is 3181984 for a five step assignment, 2903570 for a ten step assignment and 2873295 for a twenty step assignment. Since stepwise assignment is user-optimized, it is not necessary for a decreasing total system cost to accompany an increasing number of steps. However, this occurred in the example problem.

As the number of iterations increases, fewer trips are assigned to the network for each step and more users have the opportunity to make profitable decisions. One hopes that as more individuals in society make better decisions toward decreasing their own travel costs, the summation of user's cost will decrease. From Table 1 we can see that more route choice is available in ten step assignments than in five step assignments.

Table 1. Path Used On Each Step In 5 Step Assignments And 10 Step Assignments

Node Pair Path Used	(2.3)	(2.4)	(2.5)	(3.2)	(3.4)	(3.5)	(4.2)	(4.3)	(4.5)	(5.2)	(5.3)	(5.4)
Step 1	273	24	285	372	384	35	42	483	495	582	53	594
Step 2	273	24	285	372	384	35	42	483	495	582	53	594
Step 3	213	2894	284	372	37894	3784	42	483	495	582	53	594
Step 4	213	24	2465	312	384	35	42	483	495	582	43	594
Step 5	273	284	2895	312	384	3895	42	483	495	482	53	594
Step 1	273	24	285	372	384	35	42	483	495	582	53	594
Step 2	273	24	285	372	384	35	42	483	495	582	53	594
Step 3	273	24	285	372	384	35	42	483	495	582	53	594
Step 4	213	24	285	372	384	35	42	483	495	582	53	594
Step 5	213	24	285	372	37894	3785	42	483	495	582	53	594
Step 6	213	24	2895	372	3894	35	42	483	465	582	43	594
Step 7	273	27894	2784	372	3894	385	42	483	465	582	53	594
Step 8	213	24	2895	372	384	35	42	483	465	582	53	594
Step 9	283	284	285	372	3784	35	42	483	465	582	53	594
Step 10	213	24	2465	312	3894	35	42	483	465	582	53	594

CHAPTER IV

FORMULATION OF THE STEPWISE ASSIGNMENT ALGORITHM

Introduction

The stepwise assignment algorithm consists of four parts:

1. Input data - Since there are a bunch of data describing the network or the demand, an efficient way of inputting data is not only necessary in itself but also in supporting an efficient computation of shortest paths and route assignments. There are three groups of data that are used:

- (1) the length of links,
- (2) the travel demand between each origin - destination pair,
- (3) the type of roadway or guideway each link represents, and the number of lanes of each type of roadway or guideway that a link contains*. Since the data format and the handling methods have direct influence on the operation and efficiency of each procedure, input data form the basis for developing the whole algorithm.

2. Update - This part updates the cost of travel on each link after each step of flow has been assigned. It consists of a set of speed-volume relationships that can be applied to the guideway or types of each link to modify speed with respect to increasing

* In a simplified network that represents city streets, a single link may represent an arterial street or it may represent several parallel streets.

traffic volume.

3. Shortest path algorithm - With the knowledge of updated travel cost on each link, this algorithm finds the shortest path from each node to every other node. The output of this procedure is a back node matrix, which has each of its elements (i, j) representing the node preceding node j on the shortest path from node i to node j .

4. Assignment - The assignment part assigns a step of the demand to each link on the shortest path that has been identified for each origin destination pair.

The development of each of the four parts will be discussed in the following paragraphs, together with the significant contributions of this research.

Technique to Input Data

An Efficient Way To Input The Link Length Data

The objective of organizing link length data is to support efficient computation of shortest paths and route assignments. The simplest approach would be to input an $N \times N$ matrix, where N is the number of network nodes. Those node pairs that have no direct connection are assigned very large distances (9999 for example). Since in most transportation networks the number of links is very small compared to the number of possible node pairs, the $N \times N$ matrix method requires a large number of the 9999 distances to be input. This is not economical in either preparation or use. Another approach would be to designate all the link lengths in the network with individual defining statements.

This method would require as many statements as there are links in the network and is therefore an uneconomical approach for a large network.

The method selected for the stepwise assignment algorithm is to define three one-dimensional arrays: M_1 , M_2 and M_D .

The first array $M_1(N)$, which is 1 by N , indicates the number of nodes that are adjacent to or directly connected to each node of the network. We define $\underline{R}(j)$ as the set of nodes that are adjacent to or directly connected with node j . For example:

$M_1(1) = 2$ means there are two nodes adjacent to node 1,

$M_1(2) = 3$ means there are 3 nodes adjacent of node 2.

The second array $M_2(2\ell)$, which is 1 by 2ℓ , successively stores all of the adjacent node numbers for all nodes starting from node 1 and continuing through node N , where ℓ is the number of network links. Each link has 2 directions, thus 2ℓ storage positions are needed. For example:

$M_2(1) = 23$. Node 23, which is the first node adjacent to node 1, is in the first position in M_2 .

$M_2(2) = 47$. Node 47, which is the second node adjacent to node 1, is in the second position in M_2 .

$M_2(3) = 15$. Node 15, which is the first node adjacent to node 2, is in the third position in M_2 . In this example there must be only two nodes adjacent to node 1, that is, $M_2(1) = 2$.

The list is ordered serially and indexed by array $M_1(N)$. Thus, there is no indication in M_2 that $M_2(1)$ and $M_2(2)$ are associated with node 1 and that $M_2(3)$ is associated with node 2, unless one goes back to

array $M1(N)$ and notes that node 1 has only 2 adjacent nodes. Thus, the third entry in $M2$ must be associated with the node that follows node 1 --- node 2.

The third array $MD(2\ell)$ which is also 1 by 2ℓ , successively indicates the length of each link. The process begins with node 1 by first identifying all links adjacent to node 1, i.e. those nodes directly connected to node 1. Node 1 is the head node of each of these links. The process continues to node N .

The three one-dimensional arrays when taken together completely define the link lengths in the network under investigation. For example:

$MD(1) = 2.9$. 2.9 is the link length for node pair $(M2(1), 1)$, where $M2(1)$ identifies the number of the first node adjacent to node 1.

$MD(2) = 4.4$. 4.4 is the link length for node pair $(M2(2), 1)$, where $M2(2)$ identifies the number of the second node adjacent to node 1.

$MD(3) = 1.7$. 1.7 is the link length for node pair $(M2(3), 2)$, where $M2(3)$ identifies the number of the first node adjacent to node 2.

An example exhibiting the relationship between the elements of the three arrays $M1$, $M2$, and MD is shown in Figure 4.

	1	2			N
M1	2	3	...		
	1 3	5			2ℓ
M2	23	47	15		...
	1 2	3 4 5			2ℓ
MD	2.9	4.4	1.7		...

Figure 4. Relation Among Arrays M1, M2 and MD.

Element $M1(1)$ tells how many nodes are adjacent to node 1; $M1(1)$ tells how many nodes are adjacent to node 2. The first $M1(1)$ positions in $M2$ store the node numbers of the nodes adjacent to node 1. The next $M1(2)$ positions in $M2$ store the node numbers of the nodes adjacent to node 2, and so forth. Now, consider node 1 as the reference node. The first $M1(1)$ positions in MD store the length of the links from each adjacent node of node 1 to node 1, e.e. $AA(M2(II), 1) = MD(II)$, $II = 1, \dots, M1(1)$. Next, consider node 2 as the reference node. Since the next $M1(2)$ positions in $M2$ store the node number of all the nodes adjacent to node 2, the next $M1(2)$ positions in MD store the length of the links from each adjacent node of node 2 to node 2, i.e. $AA(M2(II), 2) = MD(II)$, $II = M1(1) + 1, M2(1) + 2, \dots, M1(1) + M1(2)$. The rest of the nodes are considered successively in the same manner. The only problem associated with constructing AA is the need to identify when the reference node changes. This task is facilitated by a parameter K that is used as a counter. When K is equal $M1(i)$, then all adjacent nodes of i have been considered; therefore, $i + 1$ is the node to be scanned. The process

starts with node 1 and continues until all N nodes have been scanned.

The flow chart illustrating the method of preparing the distance matrix from the three linear arrays is given in Figure 5.

Linear arrays M1, M2 and MD provide an efficient basis for inputting the link length data and provide a key concept for supporting the shortest path algorithm, which will be presented in Chapter V.

An Efficient Method For Preparing The Travel Demand Data

Each element (i, j) in a trip matrix represents the number of trips that originate at node i and terminate at node j . Typically, most of the entries in the matrix are zero. Non-zero entries tend to cluster about the diagonal of the trip matrix. It is not economical to prepare a sparse matrix as an $N \times N$ array, nor to require the computer memory to retain it. An $N \times N$ array is also awkward to use because when we assign demand to the network we must search for the O-D pairs that have non-zero entries. This is a potentially wasteful process. Demand data are completely analogous to link length data; a demand datum consists of the origin of the demand, the destination of the demand and the volume of the demand. Similarly, a link length datum consists of the head node of the link, the tail node of the link and the length of the link. This similarity suggests an approach similar to that used for the link length. Accordingly, we define a set of three one-dimensional arrays M4, M5 and M6 to efficiently store demand data.

The first array $M4(N)$, which is 1 by N, indicates the number of nodal destinations that originate from each node of the network. For example:

$M4(1) = 64$ means there are non-zero trips from node 1 to 64 other

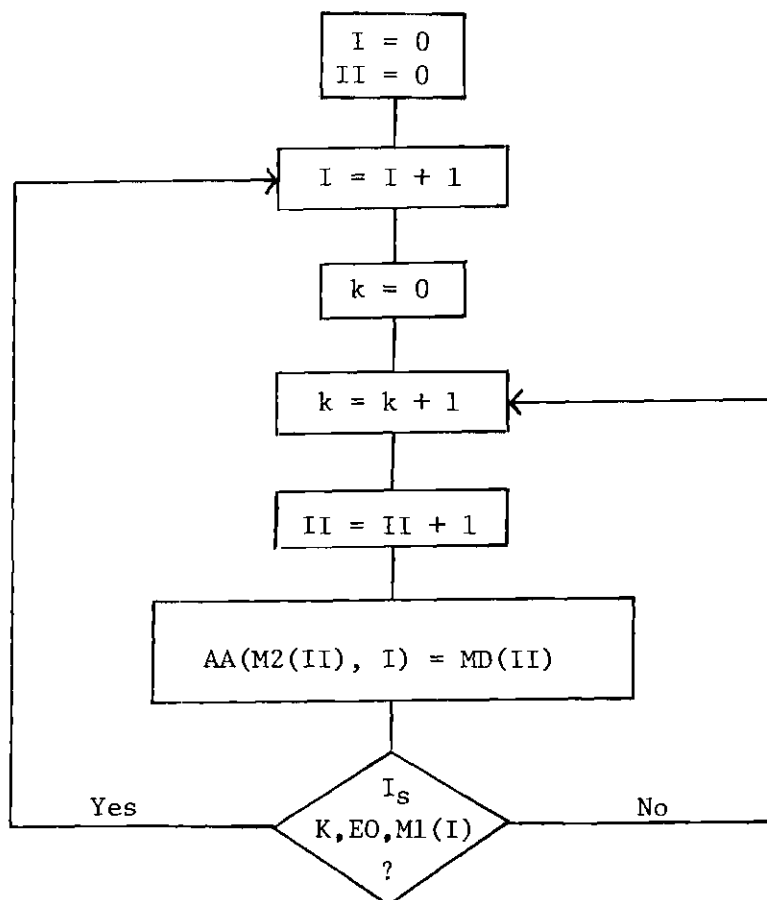


Figure 5. Flow Chart Of The Construction Of The Distance Matrix From Arrays M1, M2 and MD.

nodes

$M4(2) = 51$ means there are non-zero trips from node 2 to 51 other nodes.

The second array $M5(T)$, which is 1 by T , successively stores all of the destination node numbers originating from all nodes starting from node 1 and continuing through node N . The total number of origins and destinations in the system is designated by T . For example:

$M5(1) = 17$. The first position in $M5$ identifies node 17 as the first destination node for travel originating from node 1.

$M5(64) = 73$. The 64th position in $M5$ identifies node 73 as the last destination node for travel originating from node 1.

$M5(65) = 8$. The 65th position in $M5$ identifies node 8 as the first destination node for travel originating from node 2.

The third array $M6(T)$, which is 1 by T , successively stores all of the demand volumes to the destination nodes starting with travel originating at node 1 and continuing to travel originating at node N . The sequence of destination nodes is the same as that in $M5$. For example:

$M6(1) = 700$. 700 travelers wish to move between node pair (1, $M5(1)$).

$M6(64) = 300$. 300 travelers wish to move between node pair (1, $M5(64)$).

$M6(65) = 450$. 450 travelers wish to move between node pair (2, $M5(65)$).

An example showing the relationship between the elements of the three arrays M4, M5 and M6 is shown in Figure 6.

	1		2		N	
M4	64		51		...	
	1	64	65			T
M5	17	...	73	8
	1	64	65			T
M6	700	...	300	450

Figure 6. Relation Among Arrays M4, M5 and M6.

Each element in M5 is a destination node for a demand from some origin node. The same position in M6 gives the volume of this demand. As with the link length data, a count, k , is used to identify the change of the origin node. When k is equal to $M4(i)$, then all the demand originating from node i has been considered and $i + 1$ is the next origin node to be considered. The process starts with the node 1 and continues until all N nodes have been scanned.

The flow chart showing the logical procedure for developing the demand matrix from the three linear arrays follows:

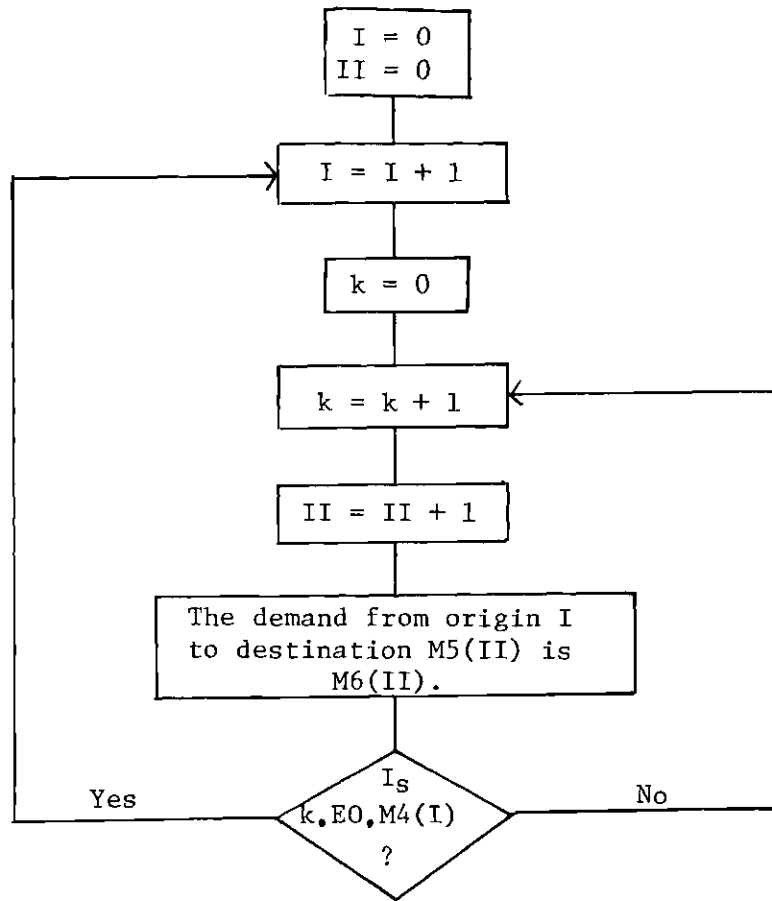


Figure 7. Flow Chart Of The Obtaining Of The Demand Information From Arrays M4, M5 and M6.

The total computer storage needed in this method is $N + T + T$, while that for the matrix method would be $N \times N$. Thus, if $T < \frac{N^2 - N}{2}$, then the proposed method is more efficient with respect to the storage used. Because the demand between origins and destinations can be explored systematically from the linear arrays, it is better not to prepare the full demand matrix for most purposes.

Link Type Data

The last remaining list of input data is given in a one-dimensional array, $M8(2\ell)$, which contains link or road type data. This array successively indicates the capacity character for each link beginning with the nodes adjacent to node 1 and continuing to node N in exactly the same sequence as the link lengths given in MD. Array M8 contains a code that identifies one of a set of possible configurations.

For a highway network, array A9 is used to indicate the number of highway lanes for the different types of road that appear. Element A9 (i) indicates the number of lanes of road type i.

Technique To Assign Flow To The Links

Although demand can occur from any origin to any destination, flows will only be distributed on the links of the network. Therefore, the flow distribution can be expressed as a one-dimensional array with the same dimension as MD, the link length array, rather than as a two-dimensional array. This approach saves both storage space and processing time over the use of an $N \times N$ matrix. Both are important for large networks because flow is assigned to each link on the shortest path for every step of every demand pair. Therefore, the operation of assigning

flows to a link will be executed many times.

In searching for an efficient assignment algorithm, three approaches were explored. Each is of some interest here because of their contributions to the evolutionary development of the final algorithm.

The First Method

The flows on each link are stored in a one-dimensional array, $AF(2\ell)$, where ℓ is the number of links in the network. Since each link has two directions, 2ℓ storage positions are needed. Following exactly the approach used to define M2 and MD, AF successively indicates the volume of flow on each link beginning with the nodes adjacent to node 1 and continuing to nodes adjacent to node N. For example, the mth element in AF represents the flow volume on the link whose length is MD(m) and whose head node is M2(m).

Once we know how much flow to assign to link (k,j), the position for the flows on link (k,j) in the array AF is determined by adding the number of adjacent nodes of each node smaller than j, plus the ordinal position of k in $K(j)$ which is represented by the notation $MN1_{k \rightarrow K(j)}$.

The sum

$$M1(1) + M1(2) + \dots + M1(j - 1) + MN1_{k \rightarrow K(j)}$$

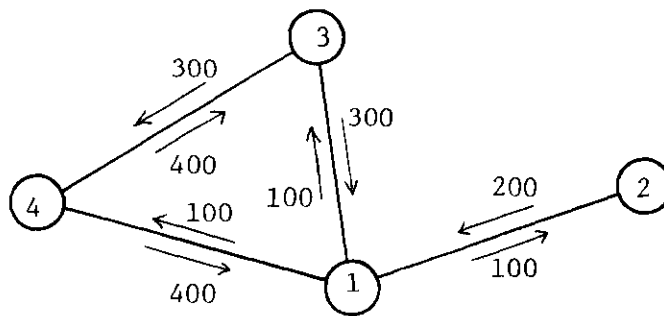
is the position in array AF where the flow on link (k,j) is stored. Whenever additional flow is to be assigned to link (k,j) it is necessary to search $K(j)$ to find the ordinal position of k in $K(j)$, that is $MN1_{k \rightarrow K(j)}$, and then perform the necessary additions. Because flows are to be

assigned to every link on the basis of its shortest path for every step of every origin destination pair, it is not economical to search, compare and add to find the necessary position in AF to store the flow. Therefore, a more efficient storage method was sought out.

The Second Method

One way to avoid both a two-dimensional array and cumbersome additions is to put the flow on link (k,j) into position $a*(j-1) + MN1_{k \rightarrow K(j)}$ in the one-dimensional array AF, where "a" is the maximum number of links which are connected to any node in the network. Therefore, we are sure that $a*N$ storage positions are sufficient to store the flows on all links. Positions $MN1_{k \rightarrow K(j)}$ is again the ordinal position of k in the set of the nodes adjacent to j.

Consider the following example which illustrates this method of assigning flow distributions. A simple network with flows assigned to each link is shown below.



The maximum number of links that are connected to any node in the network is 3. Therefore, $a=3$ in this case.

Considering node 2 as the end node:

$$K(2) = \{1\} \quad \therefore MN1_{1 \rightarrow \{1\}} = 1$$

$$a^*(J - 1) + MN1_{1 \rightarrow K(2)} = 3^*(2 - 1) + 1 = 4 \quad \therefore AF(4) = 100$$

the flow on link (1,2) is assigned to the 4th position in AF array.

Considering node 1 as the end node:

$$K(1) = \{2,3,4\} \quad \therefore MN1_{2 \rightarrow \{2-3-4\}} = 1$$

$$MN1_{3 \rightarrow \{2-3-4\}} = 2$$

$$MN1_{4 \rightarrow \{2-3-4\}} = 3$$

$$a^*(J - 1) + MN1_{2 \rightarrow K(1)} = 3^*(1 - 1) + 1 = 1 \quad \therefore AF(1) = 200$$

the flow on link (2,1) is assigned to the first position in AF array.

$$a^*(J - 1) + MN1_{3 \rightarrow K(1)} = 3^*(1 - 1) + 2 = 2 \quad \therefore AF(2) = 300$$

the flow on link (3,1) is assigned to the second position in AF array.

$$a^*(J - 1) + MN1_{4 \rightarrow K(1)} = 3^*(1 - 1) + 3 = 3 \quad \therefore AF(3) = 400$$

the flow on link (4,1) is assigned to the 3rd position in AF array.

Considering mode 3 as the end node:

$$K(3) = \{1,4\} \quad \therefore \text{MN1}_{1 \rightarrow \{1,4\}} = 1$$

$$\text{MN1}_{4 \rightarrow \{1,4\}} = 2$$

$$a*(J - 1) + \text{MN1}_{1 \rightarrow K(3)} = 3*(3 - 1) + 1 = 7 \quad \therefore \text{AF}(7) = 100$$

the flow on link (1,3) is assigned to the 7th position in AF array.

$$a*(J - 1) + \text{MN1}_{4 \rightarrow K(3)} = 3*(3 - 1) + 2 = 8 \quad \therefore \text{AF}(8) = 400$$

the flow on link (4,3) is assigned to the 8th position in AF array.

Considering node 4 as the end node:

$$K(4) = \{1,3\} \quad \therefore \text{MN1}_{1 \rightarrow \{1,3\}} = 1$$

$$\text{MN1}_{3 \rightarrow \{1,3\}} = 2$$

$$a*(J - 1) + \text{MN1}_{1 \rightarrow K(4)} = 3*(4 - 1) + 1 = 10 \quad \therefore \text{AF}(10) = 100$$

the flow on link (1,4) is assigned to the 10th position in AF array.

$$a*(J - 1) + \text{MN1}_{3 \rightarrow K(4)} = 3*(4 - 1) + 2 = 11 \quad \therefore \text{AF}(11) = 300$$

the flow on link (3,4) is assigned to the 11th position in AF array.

The positions in the one-dimensional array AF occupied by the link flows on each link are shown in Figure 8.

	1	2	3	4	5	6	7	8	9	10	11	12
AF	200	300	400	100			100	400		100	300	

Figure 8. Array AF With Elements Representing Link Flows.

Consider next the procedure for assigning flow to a shortest path of $i-k_2-k_3-j$. First, we search for the ordinal position of k_3 in $K(j)$, $MN1_{k_3 \rightarrow K(j)}$, then we assign the flow on link (k_3, j) to the position of $a^*(j-1) + MN1_{k_3 \rightarrow K(j)}$ in the AF array. Next, we search for the ordinal position of k_2 in $K(k_3)$, $MN1_{k_2 \rightarrow K(k_3)}$, then assign the flow on link (k_2, k_3) to the position of $a^*(k_3-1) + MN1_{k_2 \rightarrow K(k_3)}$ in the AF array. Finally we search for the ordinal position of i in $K(k_2)$, $MN1_{i \rightarrow K(k_2)}$, then assign the flow on link (i, k_2) to the position of $a^*(k_2-1) + MN1_{i \rightarrow K(k_2)}$ in the AF array. That is to say, we have to search and compare as many times for "MN1" as there are links in the shortest path for each demand pair. Because the nodes connecting a path are in random order, it is difficult to find the set of nodes adjacent to these nodes if they are out of order.

This approach resolves the difficulty of adding the number of nodes adjacent to each node starting from node 1 up to node $j-1$, that is $M1(1) + M1(2) + M1(3) + \dots + M1(j-1)$. However, the problem of searching for the ordinal position of the head node of a link in the set of nodes

adjacent to the tail node of the link, that is, "MN1", still remains

The Third Method

The demand between each origin-destination pair (i,j) is to be assigned to every link on the shortest path connecting node i to node j .

Since the back node matrix MN stores all of the intermediate nodes of the shortest paths, we explain below the method for identifying a shortest path from node i to node j as in Figure 9 from elements in the back node matrix.

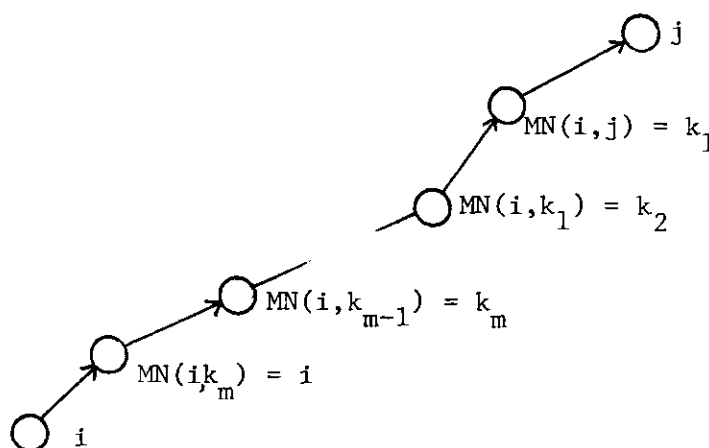


Figure 9. A Shortest Path From Node i To Node j Identified By Back Nodes.

Element $MN(i,j)$, which is the back node of node pair (i,j) , represents the preceding node to node j on the shortest path from node i to node j . Let $MN(i,j)$ be expressed as k_1 . We know that link (k_1,j) is a link on the shortest path from node i to node j . The shortest path from node i to node j thus consists of the shortest path from node i to node k_1 and the link (k_1,j) .

Element $MN(i, k_1)$, which is the back node of node pair (i, k_1) , represents the node preceding node k_1 on the shortest path from node i to node k_1 . Let $MN(i, k_1)$ be expressed as k_2 . We know that link (k_2, k_1) is on the shortest path from node i to node k_1 . So this link is on the shortest path from node i to node j . Up to now, the shortest path from node i to node j consists of the shortest path from node i to node k_2 , the link (k_2, k_1) and the link (k_1, j) . Each link on the shortest path is identified in this way. At the time the back node of some node pair (i, k_m) equals the origin node i . The shortest path from node i to node j is found and consists of the links

$(i, k_m) (k_m, k_{m-1}) \dots (k_2, k_1)$ and (k_1, j) , where

$$k_m = MN(i, k_{m-1})$$

⋮

$$k_2 = MN(i, k_1)$$

$$k_1 = MN(i, j)$$

The head nodes of each of the links in the shortest path are always expressed as the back nodes of the node pairs which are formed from the origin node to each of the tail nodes of the links. It is the ordinal position of the head node in the set of nodes adjacent to the tail node that determines the position in AF to which the flow is assigned. Therefore, in order to avoid searching the set of nodes adjacent to some nodes which are out of order, we search the set of nodes adjacent to each

of the destination nodes, starting with node 1 and continuing to node N. Whenever a set $k(j)$ is known, we compare each $MN(i,j)$, $i \in N$, with the elements in $K(j)$ to determine the ordinal position of each $MN(i,j)$, $i \in N$ in the set of $K(j)$, and store this ordinal position in the left half of the storage position of $MN(i,j)$. After all destination nodes have been scanned, we possess a new MN array which contains the back node of every node pair (i,j) in the right half storage position of $MN(i,j)$ and the ordinal position of this back node in the set of nodes adjacent to the end node j of node pair (i,j) in the left half storage position of $MN(i,j)$. See Figure 10.

Based on the new back node matrix MN, we can assign flow to the one-dimensional array AF in the following manner. At the time of assigning flow to the O-D pair (i,j) , we shift the right half of the memory word at $MN(i,j)$ to get $bn(i,j)$. Next, shift the left half of the memory word at $MN(i,j)$ to get the ordinal position of $bn(i,j)$ in $K(j)$, that is, the value of $MN1_{bn(i,j) \rightarrow K(j)}$. The flow on the link $(bn(i,j), j)$ can now be assigned to the position of $a^*(j-1) + MN1_{bn(i,j) \rightarrow K(j)}$ in the one-dimensional array AF. $Len\ bn(i,j) = k_1$. Now shift the right half of the memory word at $MN(i, k_1)$ to get the $bn(i, k_1)$, then shift the left half of the memory word at $MN(i, k_1)$ to get the ordinal position of $bn(i, k_1)$ in $K(k_1)$, that is, the value of $MN1_{bn(i, k_1) \rightarrow K(k_1)}$.

The flow on the link $(bn(i, k_1), k_1)$ can then be assigned to the position of $a^*(k_1-1) + MN1_{bn(i, k_1) \rightarrow K(k_1)}$ in the one-dimensional array AF. The flows are assigned in this way until we have $bn(i, k_m) = i$ for some node pair (i, k_m) . This means link (i, k_m) is the last link in the

A shortest path: i, k_1, k_2, k_3, j .

Back node matrix: $MN(N \times N)$

$$i \left[\begin{array}{cccc} k_1 & k_2 & k_3 & j \\ MN(i, k_1) = i & MN(i, k_2) = k_1 & MN(i, k_3) = k_2 & MN(i, j) = k_3 \end{array} \right]$$

New back node matrix: $MN(N \times N)$

Left side of each word stores the information of the position of this back node in the set of nodes adjacent to its end node.

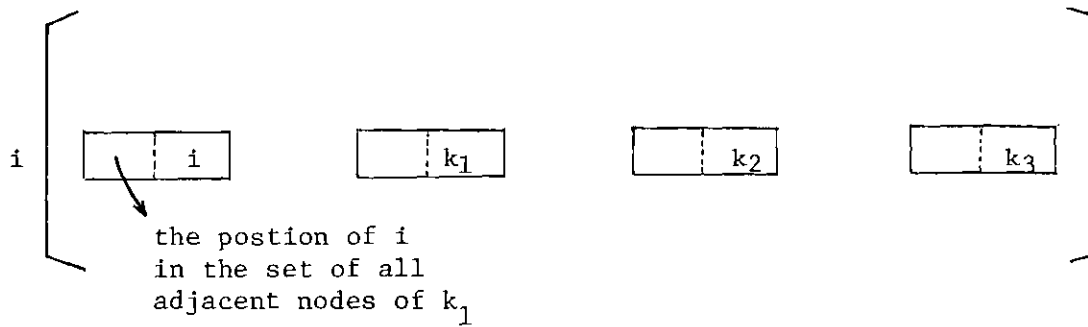


Figure 10. Back Node Matrix And New Back Node Matrix.

shortest path of O-D pair (i,j) to be assigned.

The detailed procedure is as follows:

1. Start with the destination node j, j=1.
2. Find the set of nodes adjacent to j, K(j), using linear arrays M1 and M2.
3. Compare all MN(i,j), $i \in N$, with the elements in K(j) to determine the ordinal position of each MN(i,j), $i \in N$, in K(j).
4. Store this information from step 3 in the left half of the memory word of MN(i,j) for all $i \in N$.
5. $j = j + 1$. Repeat 2, 3, 4, until $j=N$.

The element (i,j) of the new MN(NxN) matrix consist of the back mode of the shortest path (i,j), that is $bn(i,j)$, and the position of this back node in the set of nodes adjacent to node j, that is

$${}^{MN1}_{bn(i,j) \rightarrow K(j)}$$

6. Consider the first origin and destination pair (i,j).
 7. "Shift" the right half of the information in the memory word at MN(i,j) to get the back node of the node pair (i,j), that is, $bn(i,j)$.
 8. "Shift" the left half of the information in the memory word at MN(i,j) to get the position of $bn(i,j)$ in K(j), that is
- $${}^{MN1}_{bn(i,j) \rightarrow K(j)}$$
9. Assign flow on link (i,j) to the position of $ax(j-1) + {}^{MN1}_{bn(i,j) \rightarrow K(j)}$ in the AF array.
 10. Let $j = MN(i,j)$. Repeat 7, 8, and 9 until $MN(i,j) = i$.
 11. Consider another origin-destination pair (i',j'). Go to 7,

8, 9 and 10 until all demand pairs have been scanned.

The advantages of this method are as follows:

1. We need to search the nodes adjacent to some node only N times for the assignment procedure, and the nodes to be explored are in order. In using the second method, much time spent searching for the nodes adjacent to some node as there are links in the path for each demand pair. This is complicated because the nodes to be explored are out of order.
2. After we search the nodes adjacent to some destination node j , $K(j)$, we determine the ordinal position of each $MN(i,j)$, $i \in N$ in $K(j)$. Thus, N^2 times of determining the value of $MN1$ are necessary. In using the second method, the value of $MN1$ is determined as many times as there are links in the path for each demand pair.

If T is the total number of demand pairs, and if we assume that the average number of links in the shortest path connecting each demand pair is b , then the total number of times we search the set of nodes adjacent to the tail node of some link, and the number of times we determine the position of some head node in this set is $b*T$, as in the second method. If $b*T$ is not too small compared with N^2 , the third method is apparently better.

Consider now the case where a link $(MN(i,j), j)$ is the common link to many shortest paths originating from node i , as in Figure 13.

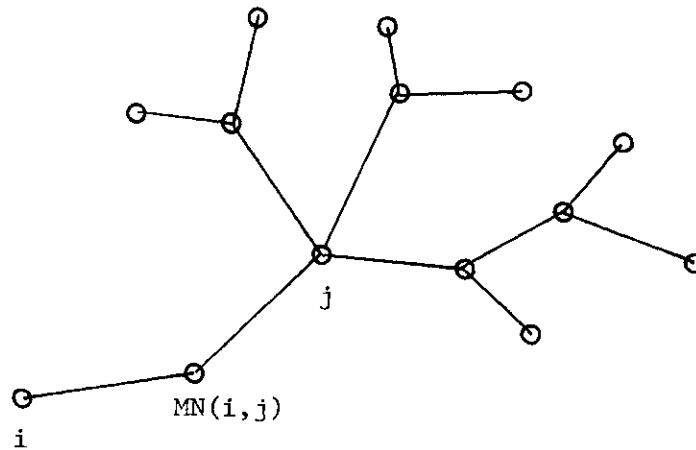


Figure 13. A Network Contains A Link Which Is Common To Many Shortest Paths.

Using the second method, we will have to search for $K(j)$ and determine the position of $MN(i,j)$ in $K(j)$ as many times as there are paths containing this link. However, in the improved third method, the position of $MN(i,j)$ in $K(j)$ is calculated and stored in left half of the memory word at $MN(i,j)$. Only a shift operation is needed to retrieve it whenever necessary.

3. The storage space is not increased, only the computation time of "shift" is additional. However, "shifting" is a very efficient operation compared with searching or comparing.

Technique To Update The Cost On Each Link

The total flow assigned to a link (i,j) in each step is stored in position

$$a*(j - 1) + MN1_{i \rightarrow K(j)}$$

of a one-dimensional array AF. We will demonstrate the efficiency of retrieving the flow data from the one-dimensional array AF to update the cost matrix for finding the shortest paths.

Even though the dimension of AF is $a*N$ instead of 2ℓ , the non-zero entries representing the flows on the links in AF are in the same sequence as the distance of each link in MD. For example, the m th non-zero element in AF represents the flow volume on the link whose length is $MD(m)$ and whose head node is $M2(m)$.

Using the linear arrays M1 and M2, we can relate each non-zero flow in AF to its corresponding link by identifying the head node and the tail node of the link.

Consider first the node j , $j=1$, as the reference node. At first the $M1(1)$ positions in M2 store the node number of all nodes adjacent to node 1. Position $MN1_{i \rightarrow K(1)}$ is the ordinal position of node i , one of the nodes adjacent to node 1, in $K(1)$. Therefore, $MN1_{i \rightarrow K(1)} = 1, \dots, M1(1)$.

$$\begin{array}{ll}
 j = 1, MN1 = 1, & \therefore L = a*(j - 1) + MN1 = 1. \quad AF(1) = AF'(M2(1), 1). \\
 MN1 = 2, & \therefore L = a*(j - 1) + MN1 = 2. \quad AF(2) = AF'(M2(2), 1). \\
 & \vdots \\
 & \vdots \\
 MN1 = M1(1) & L = a*(j - 1) + MN1 = M1(1). \quad AF(M2(1)) = AF' \\
 & (M2(M1(1)), 1).
 \end{array}$$

Array AF' is a two-dimensional array with elements representing the flow on each link.

Next, consider the node j , $j=2$, as the reference node. The next $M1(2)$ positions in $M2$ store the node number of all the nodes adjacent to node 2. Position $MN1_{i \rightarrow K(2)}$ is the ordinal position of node i , one of the nodes adjacent of node 2, in $K(2)$. Therefore, $MN1_{i \rightarrow K(2)} = 1, \dots, M1(2)$.

$$j = 2. \quad MN1 = 1 \quad \therefore L = ax(j - 1) + MN1 = a + 1. \quad AF(a + 1) = AF' \\ (M2(m1(1) + 1), 2).$$

$$M2(M1(1) + 1) K(2)$$

$$MN1 = 2. \quad \therefore L = ax(j - 1) + MN1 = a + 2. \quad AF(a + 1) = AF' \\ (M2(m2(1) + 2), 2).$$

$$M2(M2(1) + 2) K(2)$$

$\begin{matrix} \cdot & & \cdot & & \cdot \\ \cdot & & \cdot & & \cdot \\ \cdot & & \cdot & & \cdot \end{matrix}$

$$MN1 = M1(2) \quad \therefore L = ax(j - 1) + MN1 = a + M1(2). \quad AF(a + M2(2)) = \\ M](1), 2).$$

$$M2(M2(1) + M1(2)) K(2)$$

In this manner we sequentially transfer the flow on the one-dimensional array AF to the two-dimensional array AF' .

The problem is now to identify when the reference node changes. This task is facilitated by a parameter k that is used as a counter. When k is equal to $M1(j)$, all direct links to node j have been considered and $j+1$ is the next reference node to be scanned. The process starts with

node 1 and continues until all N nodes have been considered as reference node.

In this manner, we can relate the flow distribution from a one-dimensional array AF to a corresponding two-dimensional expression as in Figure 14.

We know the flow on each link from the one-dimensional array. From M8, we know the type of the roadway or guideway each link represents. The speed on the link for the present flow can be found by using the speed-flow relationship for this type of road. The distance of each link divided by the travel speed plus the delay in the intersection is the travel time for using this link. Now, using the knowledge of the correspondence between the one-dimensional expression and two-dimensional expression, we transfer the travel time to a two-dimensional array which is our new updated cost matrix AA. The procedure comes from using the linear arrays M1, M2 and MD, and will be clearly demonstrated by the flow chart in Figure 15.

Technique To Find The Shortest Paths

This will be discussed separately on the Chapter of Shortest Path Algorithm.

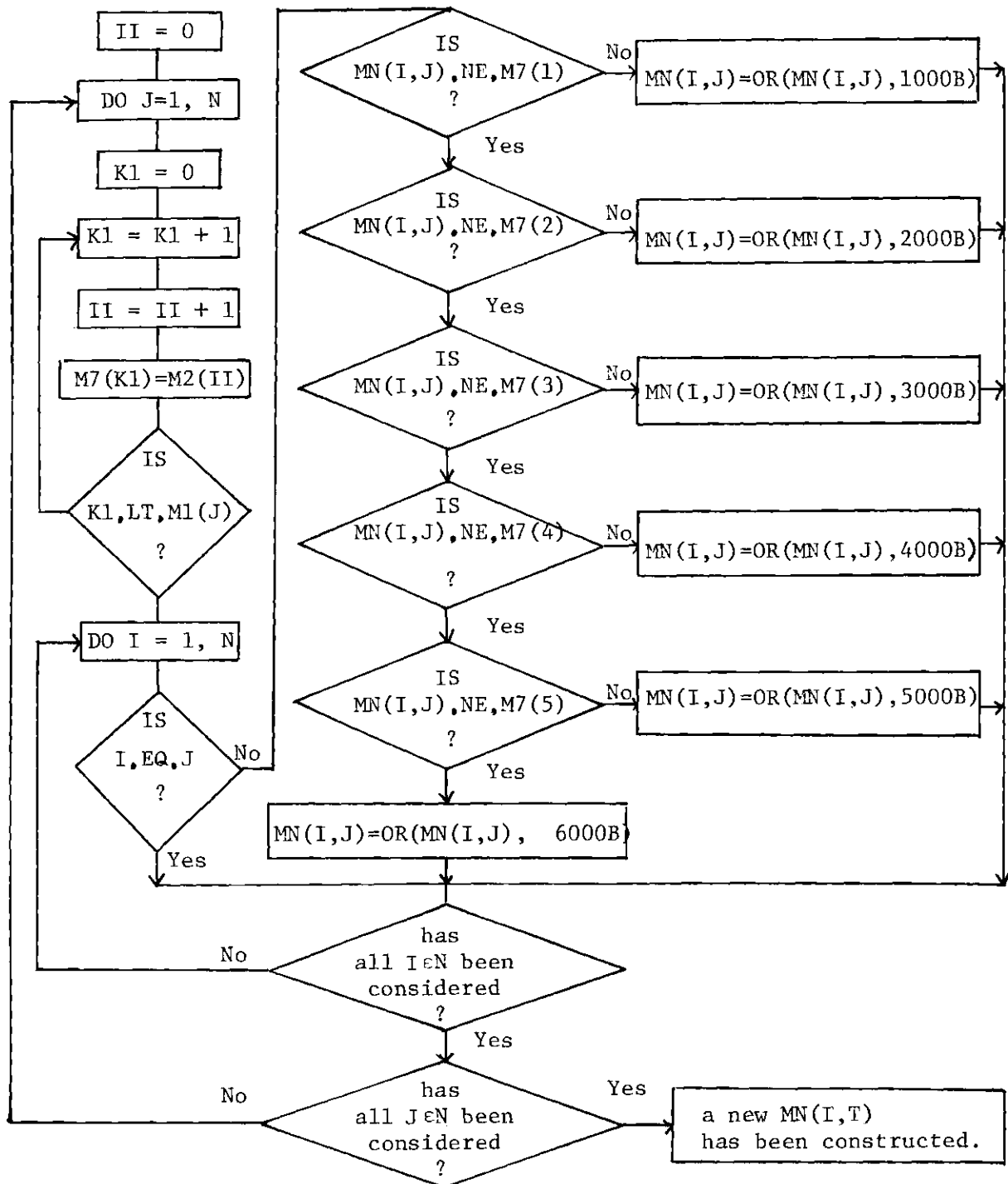


Figure 11. Flow Chart Of The Construction Of A New MN Array.

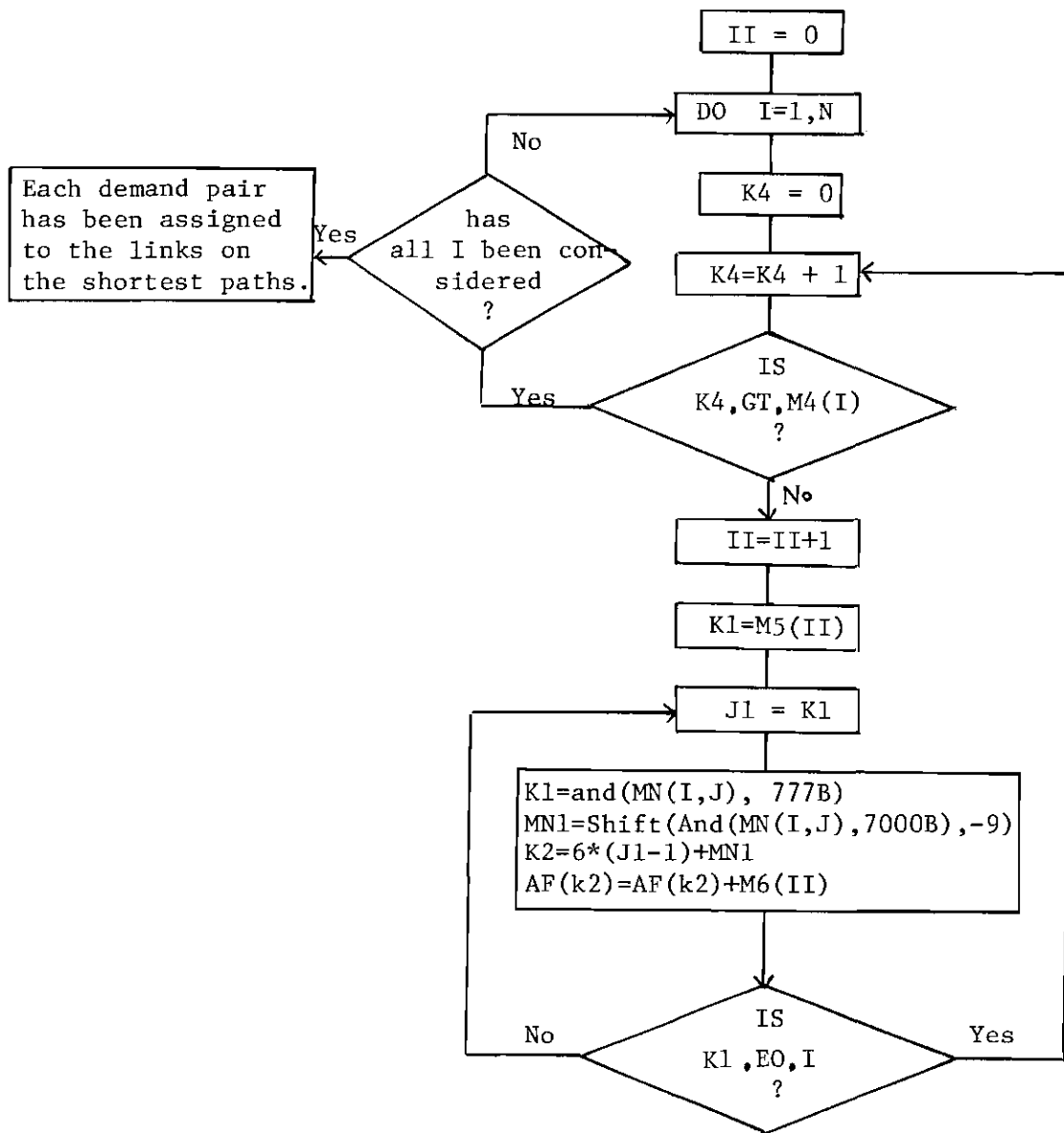


Figure 12. Flow Chart Of The Assignment Of Demand To The Links On The Shortest Path.

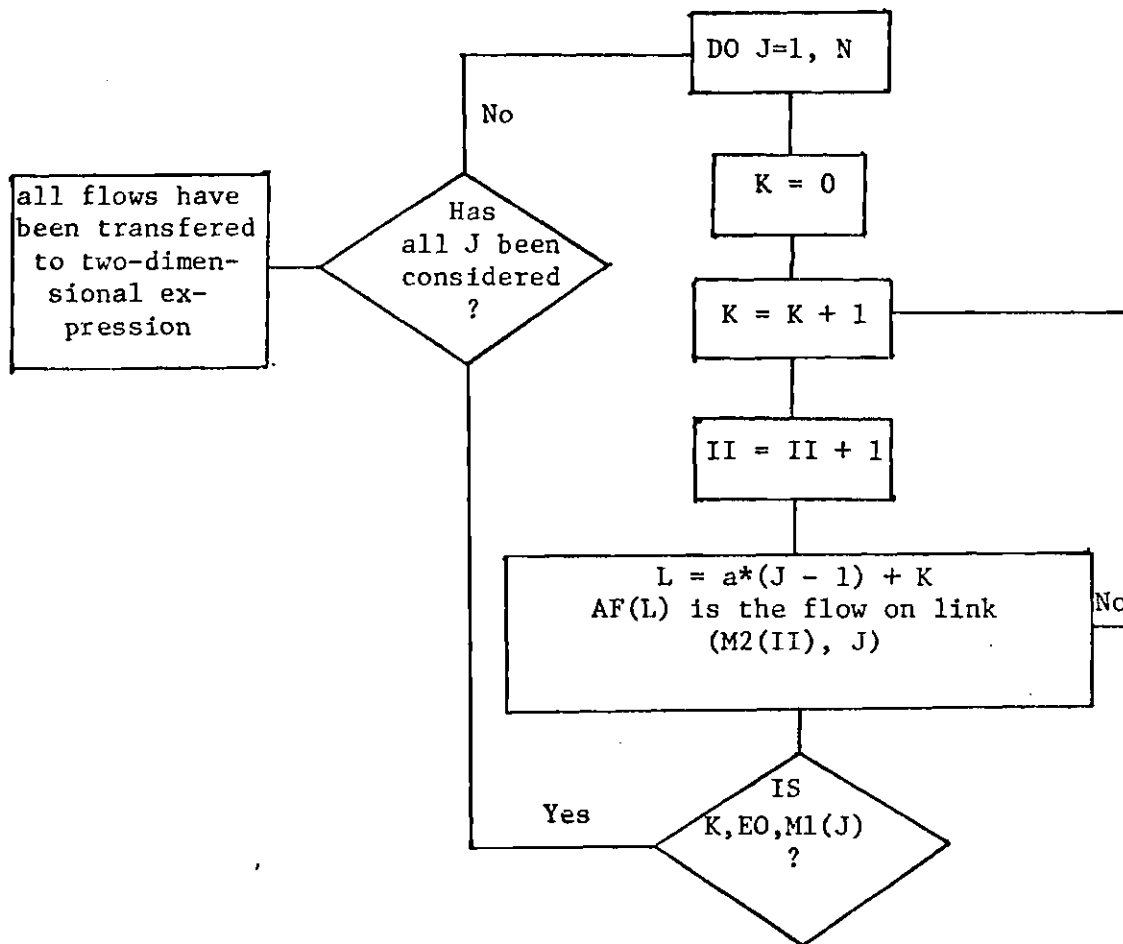


Figure 14. Flow Chart Of The Conversion Of Flow Distribution From A One-Dimensional Array AF To A Two-Dimensional Array.

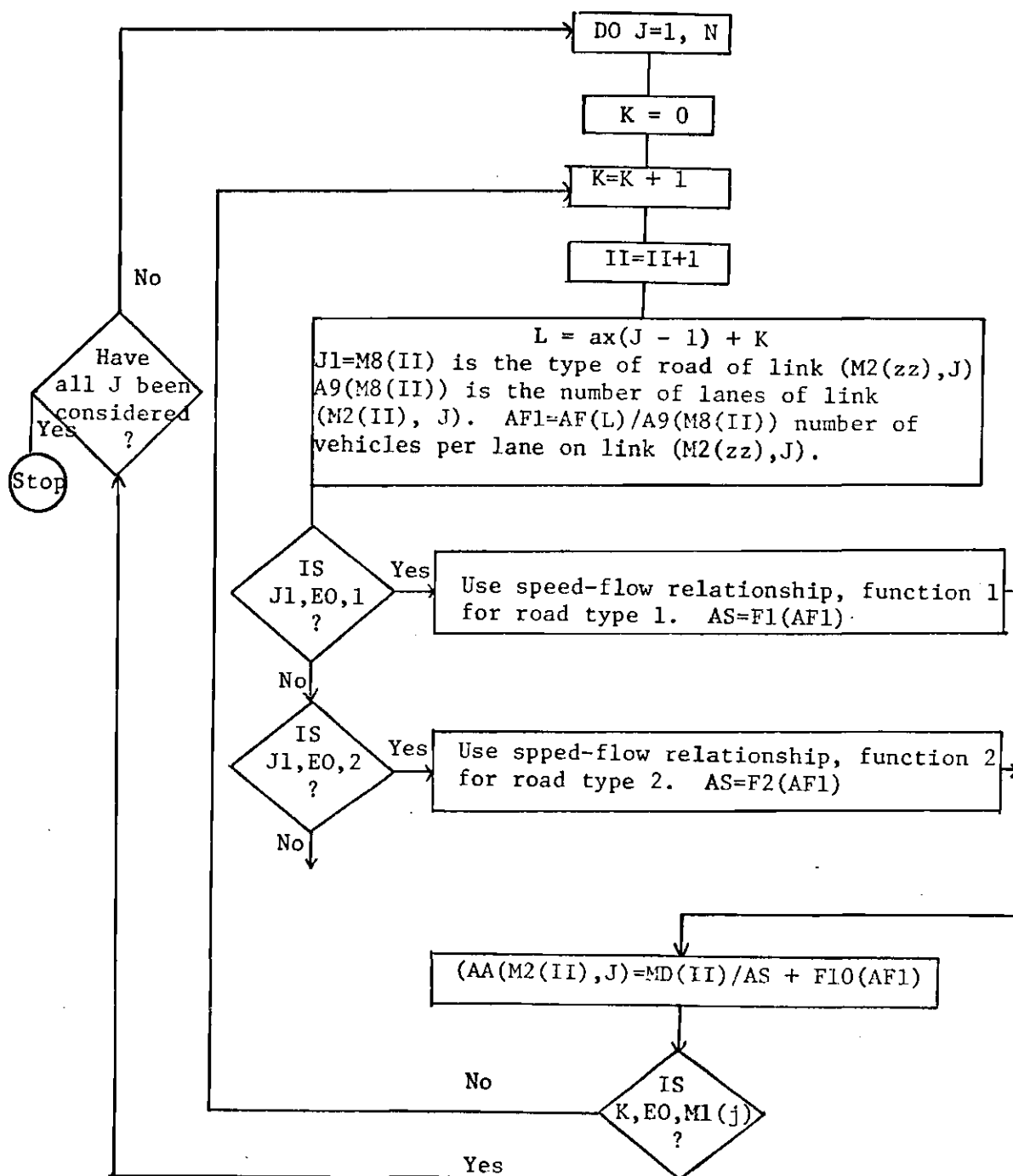


Figure 15. Flow Chart Of The Updating Of Cost Matrix AA Based On The Flow Distribution Stored In A One-Dimensional AF Array.

CHAPTER V

SHORTEST PATH ALGORITHM

Introduction

Shortest path algorithms are of vital importance in solving numerous transportation problems (especially traffic assignment problems). A great deal of efforts have been made to make the algorithm as efficient as possible. In this thesis, a new algorithm is presented based on the improvement of three other algorithms devised during this investigation. This algorithm will be shown to be more efficient than other matrix methods of shortest path algorithms.

The shortest path algorithm may be summarized in a number of ways:

1. Find the shortest path from one node to another node.
2. Find the shortest path from one node to all other nodes.
3. Find the shortest paths between all pairs of nodes.

Following will be a brief discussion of definitions peculiar to the shortest path problem.

A path of node pair (i,j) is a sequence of links i,k_1,k_2,k_3,k_4,j , starting with node i and ending with node j .

The length of a path is the summation of the lengths of all links in the path.

$$d_{ij} = d_{ik_1} + d_{k_1k_2} + d_{k_2k_3} + d_{k_3k_4} + d_{k_4j}$$

The shortest path for node pair (i,j) is one of the possible paths from i to j with the smallest length. This can be defined with the following recursive relationship:

$$d_{ij}^* = \min_{k \in K(j)} (d_{ik}^* + d_{kj}^o)$$

A back node is the node preceding the end node on a shortest path. In order to designate nodes along the shortest path, the following example will illustrate how the shortest path is indicated.

Consider the following information:

$bn(i,j) = k_2$ means that the shortest path from i to j passes node

$$k_2 \text{ before reaching } j. \quad d_{ij}^* = d_{ik_2}^* + d_{k_2j}^o.$$

$bn(i,k_2) = k_1$ means that the shortest path from i to k_2 passes

$$\text{node } k_1 \text{ before reaching } k_2, \quad d_{ik_2}^* = d_{ik_1}^* + d_{k_1k_2}^o.$$

$bn(i,k_1) = i$ means that the shortest path from i to k_1 passes node

$$i \text{ before reaching } k_1, \quad d_{ik_1}^* = d_{ii}^o + d_{ik_1}^o.$$

Knowing this, the shortest path from node i to node j must be i- k_1 - k_2 -j.

Previous Research

There are two main types of algorithms for computing the shortest paths between many nodes:

1. Tree building algorithms, and
2. Matrix algorithms.

In a tree building algorithm, the shortest path from one node to all other nodes is generated as a tree. It can be used to find the shortest paths from N origins to many destinations by repeating the

algorithm N times. The advantage is that it is only necessary to keep one tree at a time in storage.

In a matrix algorithm the lengths of each link or the shortest distance for each node pair are stored in the form of a matrix. The shortest paths between all nodes are obtained simultaneously once the final distance matrix has been updated.

Selection of a particular algorithm usually depends on the specific characteristics of each problem.

Two of the most efficient matrix algorithms are those devised by Floyd^[7] and Dantzig^[4].

The Algorithm Of Floyd

Given a network of N nodes, numbered arbitrarily from 1 to N , we let an $N \times N$ matrix represent the distances between each node pair. If an arc exists between node i and node j , d_{ij} represents the length of the arc. If no arc exists between node i and node j , $d_{ij} = \infty$. For all $i \in N$, $d_{ii} = 0$. Starting with the $N \times N$ matrix of direct distances, the Floyd procedure builds optimal paths by improving the matrix N times. In each iteration it considers a new node as an intermediate node. The $(k+1)$ th matrix is constructed from the k th matrix by considering node $(k+1)$ as the intermediate node and satisfying the following formula:

$$d_{ij}^{(k+1)} = \min_{k+1} [d_{ij}^{(k)}, d_{i, k+1}^{(k)} + d_{k+1, j}^{(k)}] \text{ for all } i \in N, j \in N$$

If a shorter path is detected by considering node $(k+1)$ as the intermediate node, the back node for the shortest path between node pair (i, j)

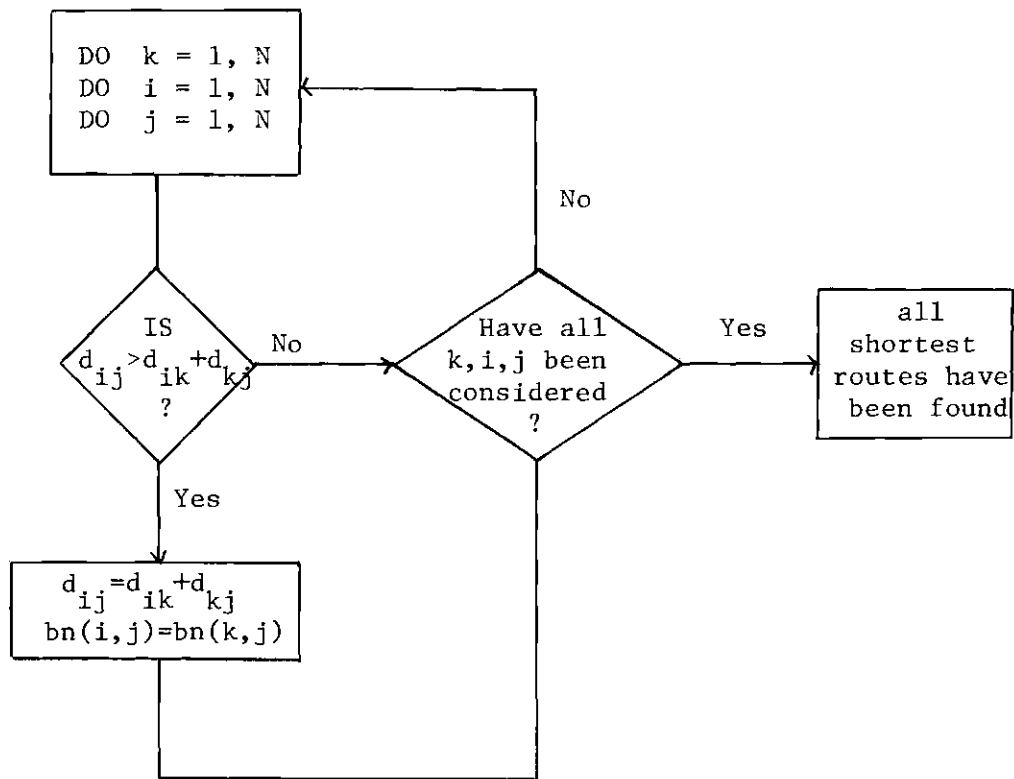


Figure 16. Flow Chart Of The Floyd's Algorithm.

is the same as the back node of node pair $(k+1, j)$. The shortest distance for node pair (i, j) is $d_{ij}^{(N)}$. Since there are N intermediate nodes to be considered, for each intermediate node n , when $i=n$, $j=n$ or $j=i$, no inspection is needed. Therefore, the total number of additions and comparisons is $N(N-1)(N-2)$.

The Algorithm Of Dantzig

Dantzig's scheme generates successive matrices of increasing size. The k th iteration produces a $K \times K$ matrix whose elements are the shortest distances connecting node i and node j , $i=1 \dots k$, $j=1 \dots k$, with possible intermediate node k , $k=1 \dots k$. The $(k+1)$ th iteration considers node $(k+1)$ as the intermediate node. The elements of the $(k+1) \times (k+1)$ matrix are updated as follows:

1. Compute $d_{i, k+1}^{(k+1)}$ for $i=1, \dots, k$ by

$$d_{i, k+1}^{(k+1)} = \min_{m=1 \dots k} [d_{i, m}^{(k)} + d_{m, k+1}^{(0)}].$$

2. Compute $d_{k+1, j}^{(k+1)}$ for $j=1, \dots, k$ by

$$d_{k+1, j}^{(k+1)} = \min_{m=1 \dots k} [d_{k+1, m}^{(k)} + d_{m, j}^{(0)}]$$

3. Compute $d_{ij}^{(k+1)}$ for $i=1, \dots, k$, $j=1, \dots, k$ by

$$d_{ij}^{(k+1)} = \min_{k+1} [d_{ij}^{(k)}, d_{i, k+1}^{(k+1)} + d_{k+1, j}^{(k+1)}]$$

The shortest distance for node pair (i, j) is $d_{ij}^{(N)}$. There are k $d_{i, k+1}^{(k+1)}$'s, each one needing k additions and comparisons; therefore, k^2 additions and comparisons are needed for the determination of $d_{i, k+1}^{(k+1)}$, $i=1, \dots, k$ or

$d_{k+1, j}^{(k+1)}$, $j=1, \dots, k$.

When $j=i$, $d_{i, j}^{(k+1)}$ does not need to be inspected. There are $(k) \times (k-1)$ additions and comparisons for the evaluation of $d_{i, j}^{(k+1)}$, $i=1 \dots k$, $j=1 \dots, k$. Therefore, the total number of additions and comparisons is

$$\sum_{k=0}^{N-1} \{2k^2 + (k)(k-1)\} = N(N-1)^2$$

Hence, the algorithms of Floyd and Dantzig have the same efficiency.

A New Shortest Path Algorithm

The objective is to find the shortest paths between all pairs of nodes such that:

1. Distances are nonnegative, $d_{ij} \geq 0$.
2. Distances are not necessarily symmetric, $d_{ij} \neq d_{ji}$.

Instead of defining the shortest distance as:

$$d_{ij}^* = \min_{k \in N} [d_{ik}^* + d_{kj}^*],$$

in the new algorithm it will be defined as follows:

$$d_{ij}^* = \min_{k \in K(j)} [d_{ik}^* + d_{kj}^{(0)}] \text{ for every } i, j, i \neq j$$

$$d_{ij} = 0 \text{ if } i = j.$$

This definition also forms the basic operation for computing the shortest path.

Following will be a brief discussion of the development of four algorithms, each one being an improvement over the previous one.

The First Algorithm

Start with the $N \times N$ distance matrix whose element d_{ij} represents the length of the arc. If there is a direct arc connecting node i and node j , $d_{ij} < \infty$. If there is no arc connecting node i and node j , $d_{ij} = \infty$ and $d_{ii} = 0$, for all $i \in N$. For each node pair (i, j) , try all k , $k \in N$ as intermediate nodes. Let $d_{ij}^1 = \min_{k \in N} (d_{ik}^0 + d_{kj}^0)$. When every node pair has been examined once, we claim that an iteration has been performed. If any $d_{ij}^1 \neq d_{ij}^0$ another iteration is necessary. Let $d_{ij}^2 = \min_{k \in N} (d_{ik}^1 + d_{kj}^1)$. If any $d_{ij}^2 \neq d_{ij}^1$, perform another iteration. By the time we have $d_{ij}^{\ell+1} = d_{ij}^{\ell}$ for all $i, j \in N$, the $d_{ij}^* = d_{ij}^{\ell}$ for all $i, j \in N$. The shortest distance for each node pair has thus been determined.

This may be illustrated in the following example, which shows that the shortest path for any node pair (i, j) can be obtained in a finite number of iterations.

Suppose the shortest path for node pair (i, j) is $i-k_1-k_2-k_3-j$. In the first iteration: the examination of node pair (i, k_2) , with k_1 as intermediate node, is one of the $M(n-1)(N-2)$ examinations. Therefore, we obtain $d_{ik_2}^* = d_{ik_1}^* + d_{k_1k_2}^*$.

Continuing the first iteration: The examination of node pair (k_2, j) with k_3 as intermediate node is also one of the $N(N-1)(N-2)$ examinations. Therefore, we obtain $d_{i_2j}^* = d_{k_2k_3}^* + d_{k_3j}^*$.

In the second iteration: The examination of node pair (i,j) with k_2 as intermediate node is one of the $N(N-1)(N-2)$ examinations. Since $d_{ik_2}^*$ and $d_{k_2j}^*$ are known from the first iteration, we obtain $d_{ij}^* = d_{ik_2}^* + d_{k_2j}^*$.

In other words, the shortest path for node pair (i,j) can be obtained in 2 iterations.

The procedure for the first method is as follows:

1. The sequence of node pairs being examined is in rowwise order:
 $(1,1)(1,2)(1,3) \dots (1,N), (2,1)(2,2)(2,3) \dots (2,N)(3,1) \dots$
 $(3,N) \dots (N,1)(N,2) \dots (N,N)$.

2. Select the first node pair (i,j) in the sequence.

3. Compare d_{ij} with $d_{ik} + d_{kj}$ for all $k \in N$.

If $d_{ij} > d_{ik} + d_{kj}$ for some $k \in N$, let $d_{ij} = d_{ik} + d_{kj}$,
 $bn(i,j) = bn(k,j)$.

If $d_{ij} \leq d_{ik} + d_{kj}$, d_{ij} remains unchanged.

4. Select the next node pair (i',j') in the sequence.

5. Do the comparisons as in 3 for all $k \in N$.

6. After every node pair (i,j) has been examined, one iteration is completed. Determine if any shorter distance was found for any node pair. If so, go back to 2. Examine every node pair once again. If a shorter distance can not be found, the d_{ik} 's in the present matrix are the shortest distances for each node pair.

The flow chart of this method is shown in Figure 17. From Figure 16 and Figure 17 we can see that Floyd's algorithm examines all (i,j) pairs for each k as an intermediate node. This algorithm examines all k intermediate nodes for each (i,j) pair.

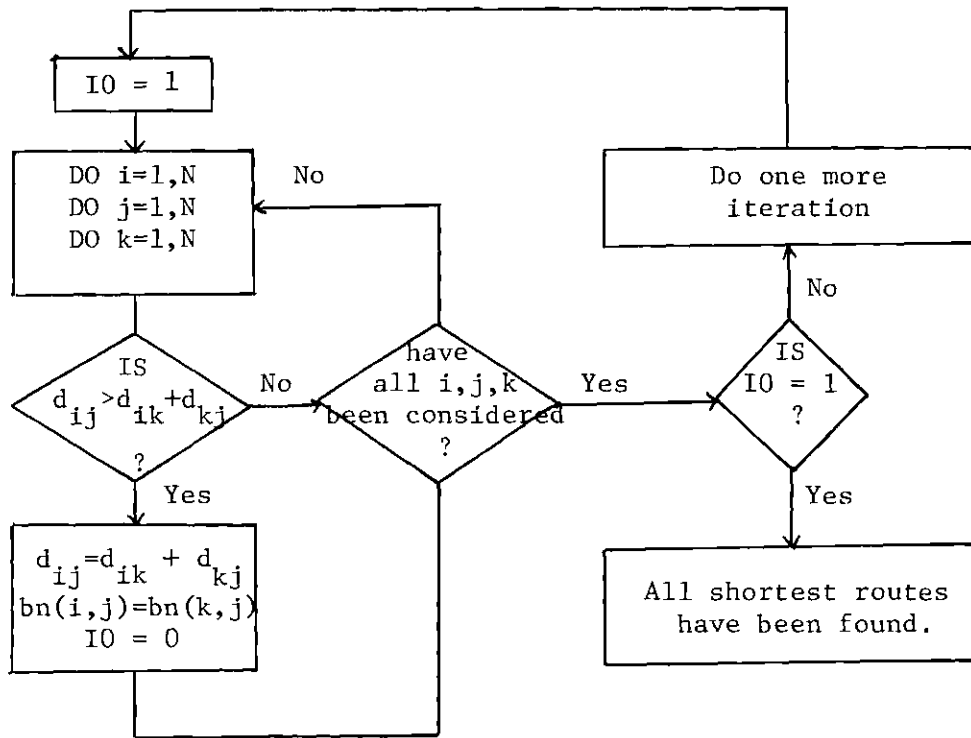


Figure 17. Flow Chart Of The First Shortest Path Algorithm.

An example for finding the shortest path is as follows: suppose the shortest path for node pair (1,2) is 1-3-4-5-2.

The sequence of node pairs to be examined is fixed as (1,1), (1,2), (1,3), (1,4), (1,5), (2,1), (2,2), (2,3), (2,4), (2,5), (3,1), (3,2), (3,3), (3,4), (3,5), (4,1), (4,2), (4,3), (4,4), (4,5), (5,1), (5,2), (5,3), (5,4), (5,5). Since the examination of node pair (1,2) comes before the examination of node pairs (1,3), (3,2), (1,4), (4,2), (1,5), (5,2) in each iteration, d_{13}^* , d_{32}^* , d_{14}^* , d_{42}^* , d_{15}^* , d_{52}^* is not known at the time the node pair (1,2) is examined in the first iteration.

The shortest path for (1,2) is either equal to $d_{11}^* + d_{12}^*$, $d_{12}^* + d_{22}^*$, $d_{13}^* + d_{32}^*$, $d_{14}^* + d_{42}^*$ or $d_{15}^* + d_{52}^*$. Therefore, d_{12}^* is not obtained in the first iteration.

From this example, we can see that both d_{ik}^* and d_{kj}^* for any $k \in N$ must be known in advance before d_{ij}^* can be evaluated, which is the main reason why it is usually not possible to get the shortest path for every node pair in the first iteration. The node number in the path affects the sequence of examination, and is thus a factor in determining the number of iterations needed to find the shortest path. The total number of additions and comparisons in the algorithm is

$$N(N - 1)(N - 2) \text{ (number of iterations).}$$

It is clear that the first method is definitely inferior to the method of Floyd and Dantzig.

The Second Algorithm

Start with the $N \times N$ distance matrix. For each node pair (i,j) , try only the nodes adjacent to j as intermediate nodes; that is, all $k \in K(j)$. Check to see if $d_{ij} > d_{ik} + d_{kj}$ for each $k \in K(j)$. If any shorter route is found in this iteration, another iteration is needed. Otherwise, the shortest path for all node pairs has been obtained. In fact, this algorithm combines some of the ideas of tree building algorithm and matrix algorithm.

The advantage of this algorithm depends on an efficient way of searching the nodes adjacent to each destination node j . Three one-dimensional arrays, $M1$, $M2$ and MD are used to represent the input data describing the network. The first $M1(1)$ positions in $M2$ and MD store the nodes adjacent to node 1, and the link lengths between these nodes to node 1. The next $M1(2)$ positions in $M2$ and MD store the nodes adjacent to node 2 and the link lengths between these nodes to node 2. The sequence of node pairs to be examined is as follows: $(1,1)$, $(1,2)$, $(1,3)$..., $(1,N)$, $(2,1)$..., $(2,N)$, $(3,1)$..., $(3,N)$, $(N,1)$, $(N,2)$..., (N,N) . The destination nodes in this sequence come out in order for each origin node i . Therefore, the nodes adjacent to these destination nodes can be picked out one by one in $M2$. The distance between the adjacent node to the destination node can also be picked out successively in MD . An index II is used to designate the position of an adjacent node in $M2$ and its corresponding link length in MD . Each time the index II is increased by 1, another adjacent node and the link length between this node to the destination node is obtained by $M2(II+1)$ and $MD(II+1)$. A parameter k is used to tell when the nodes adjacent to a destination have been considered.

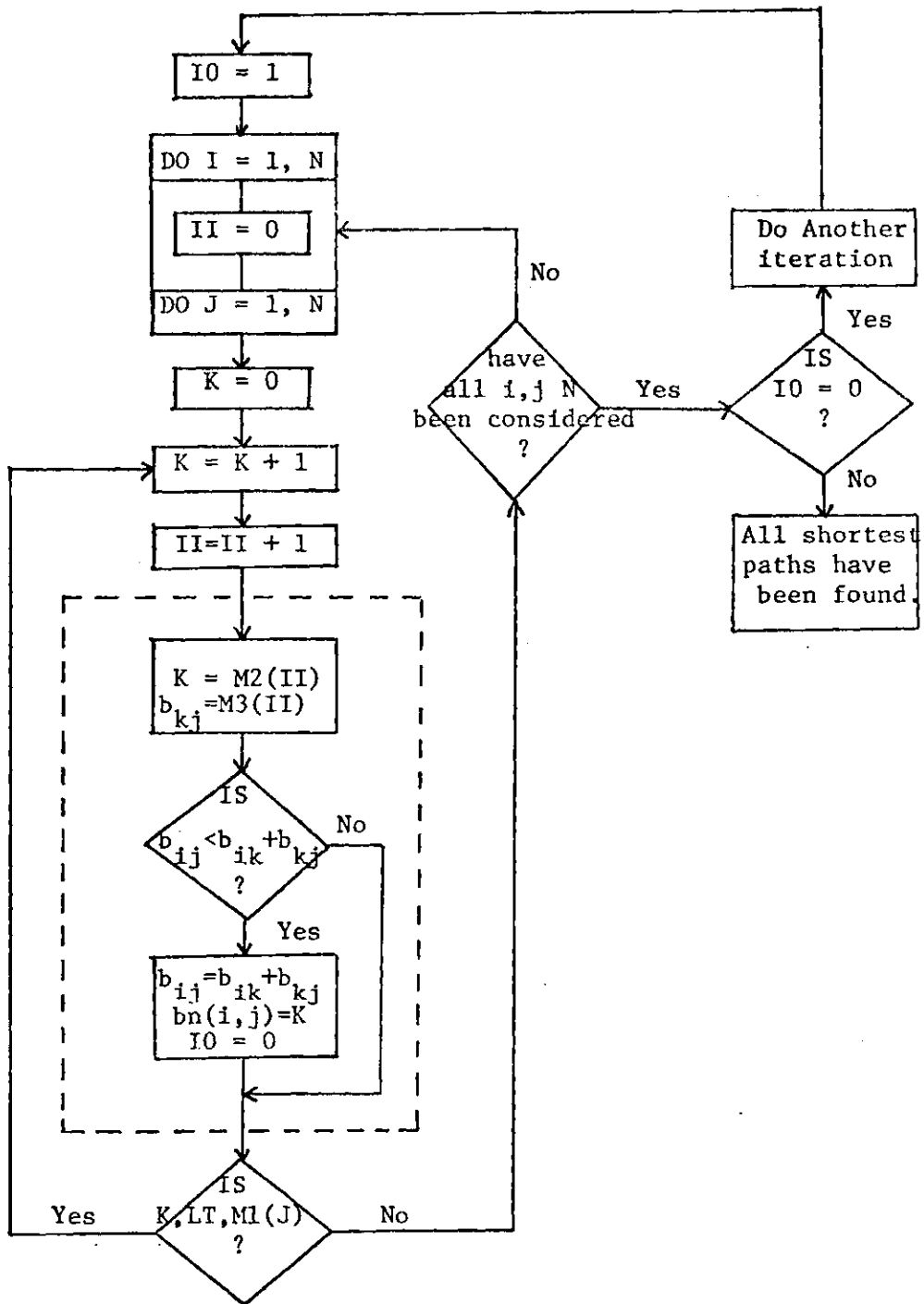


Figure 18. Flow Chart Of The Second New Shortest Path Algorithm.

When k is equal to $M1(j)$, then all nodes adjacent to node j have been considered, and $j+1$ is the next destination node, where $M1(j+1)$ adjacent nodes are stored in positions from

$$M1(1) + M1(2) + \dots + M1(j) + \underline{1} \text{ to}$$

$$M1(1) + M1(2) + \dots + M1(j) + \underline{M1(j+1)} \text{ in } M2.$$

The procedure for the second method is as follows:

1. The sequence of node pairs being examined is in rowwise order:
 $(1,1), (1,2), (1,3) \dots (1,N), (2,1), (2,2) \dots (2,N), (3,1) \dots$
 $(3,N) \dots (N,1), (N,2) \dots (N,N).$

2. For each node pair (i,j) , search all the adjacent nodes of j ; that is, every $k \in K(j)$.

3. Compare d_{ij} with $d_{ik} + d_{kj}^{(0)}$ for each $k \in K(j)$.

If $d_{ij} > d_{ik} + d_{kj}^{(0)}$ for some $k \in K(j)$, let $d_{ij} = d_{ik} + d_{kj}^{(0)}$
 $bn(i,j) = k.$

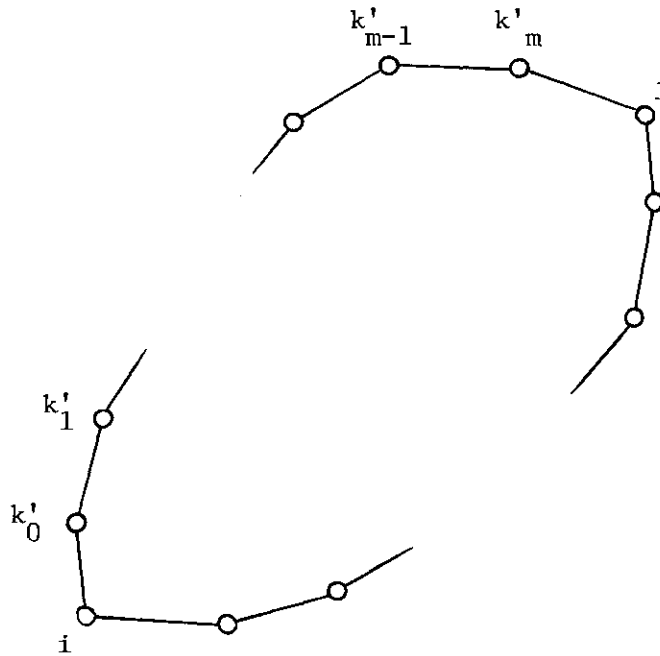
If $d_{ij} \leq d_{ik} + d_{kj}^{(0)}$, d_{ij} remains unchanged.

4. After every node pair (i,j) $i, j \in N$ $i \neq j$ has been examined, determine if any shorter distance was found for any node pair. If so, another iteration is necessary. If not, the d_{ij} 's in the present distance matrix are the shortest distances for each node pair.

The flow chart of this method is shown in Figure 18.

Proof Of Optimality. The algorithm stops when $d_{ij} \leq d_{ik} + d_{kj}$ $k \in K(j)$ for every node pair (i,j) .

1. First, we prove that the shortest paths found in this method were really the shortest paths.



Suppose a shortest path for node pair (i, j) has been found by our algorithm and this distance is d_{ij}^* . Suppose there is another path for node pair (i, j) , $(i-k'_0-k'_1-k'_2-\dots-k'_m-k)$ which is shorter but has not been found by our algorithm. The distance is

$$d_{ij}' = d_{ik'_0} + d_{k'_0k'_1} + d_{k'_1k'_2} + \dots + d_{k'_m j}$$

According to the stopping criteria, the following relation must be true.

$$d_{ij} = \text{Min}_{k \in K(j)} \{d_{ik} + d_{kj}\} \text{ for all node pair } (i,j), i \in N, j \in N.$$

$$\text{Since, } d_{ij} \leq d_{ik'_m} + d_{k'_m j}$$

$$\therefore d_{ij} - d_{ik'_m} \leq d_{k'_m j} \text{ ----- 1.}$$

$$\text{Since } d_{ik'_m} \leq d_{ik'_{m-1}} + d_{k'_{m-1} k'_m}$$

$$\therefore d_{ik'_m} - d_{ik'_{m-1}} \leq d_{k'_{m-1} k'_m} \text{ ----- 2.}$$

⋮

$$\text{Since } d_{ik'_1} \leq d_{ik'_0} + d_{k'_0 k'_1}$$

$$\therefore d_{ik'_1} - d_{ik'_0} \leq d_{k'_0 k'_1} \text{ ----- m-1.}$$

$$\text{Since } d_{ik'_0} \leq d_{ii} + d_{ik'_0}$$

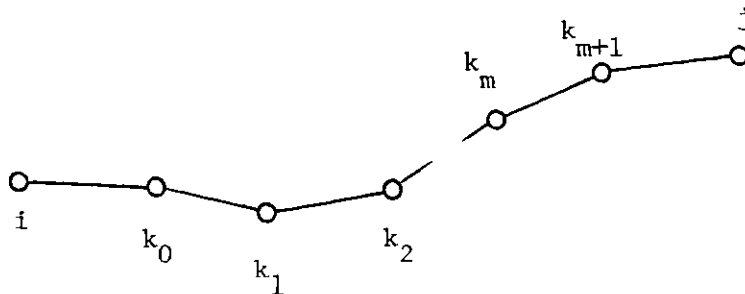
$$\therefore d_{ik'_0} - 0 \leq d_{ik'_0} \text{ ----- m.}$$

Then adding equations 1 through m, and canceling like terms, we have

$$d_{ij} \leq d_{ik_0} + d_{k_0k_1} + d_{k_1k_2} + d_{k_2k_3} + \dots + d_{k_mj} = d'_{ij}$$

Therefore, no path connecting node i and node j is shorter than the shortest path found by our algorithm. Hence, the paths found by our algorithm are indeed the shortest paths.

Second, we prove that every shortest path is able to be found by our algorithm. For any node pair (i,j) in the network, suppose the shortest path for this node pair is $(i, k_0, k_1, k_2, \dots, k_m, j)$. We now prove that it will be found before our algorithm stops.



The shortest path from node i to node k_0 is d_{ik_0} . The shortest path from node k_0 to node k_1 is $d_{k_0k_1}$, etc. If the previous statements are not true then our assumption that $(i, k_0, k_1, \dots, k_m, j)$ is the shortest path for node pair (i, j) will be wrong.

By mathematical induction the proof follows:

$$\therefore d_{ik_1} = \text{Min}_{k \in K(k_1)} \left\{ d_{ik} + d_{kk_1} \right\}, \text{ and } k_0 \in K(k_1)$$

$\therefore d_{ik_1}^* = d_{ik_0}^* + d_{k_0k_1}$ is the shortest path from node i to node k_1 that

can be found in the first iteration.

Since there is a shorter route found in this iteration, there will be another iteration following this iteration.

When $n=2$:

$$\therefore d_{ik_2} = \text{Min}_{k \in K(k_2)} \left\{ d_{ik} + d_{kk_2} \right\}$$

and $k_1 \in K(k_2)$.

Since $d_{ik_1}^*$ has been found, $d_{ik_1}^* + d_{k_1k_2}$ (the shortest distance from node i to node k_2) will be found before the end of the next iteration.

After the shortest route $d_{ik_2}^*$ is found, the algorithm will perform one more iteration.

When $n=m$, $m > 2$:

We know that the number of iterations needed to find $d_{ik_m}^*$ is less than one plus the number of iterations needed to find $d_{ik_{m-1}}^*$.

When $n=m+1$:

We know that the algorithm will perform one more iteration after $d_{ik_m}^*$ is found.

$$\therefore d_{i,k_{m+1}} = \min_{k \in K(k_{m+1})} d_{ik_m} + d_{k_m k_{m+1}}$$

$$\text{and } k_m \in K(k_{m+1}).$$

Since d_{i,k_m}^* is found, $d_{ik_m} + d_{k_m k_{m+1}}$ (the shortest distance from node i to node k_{m+1}) will be found before the end of the next iteration. That is to say, $d_{i,k_{m+1}}^*$ can be found before the algorithm stops. Therefore, the shortest path from i to j will be found by this algorithm.

The improvement of this algorithm eliminates the necessity of making additions and comparisons for many useless k 's. Although there may be an increase in the number of iterations, the decrease in the number of additions and comparisons makes this algorithm more beneficial than the previous algorithm. $(N)(N-1)(N-2)$ (number of iterations) additions and comparisons for the previous method is far more than $(N)(N-1)(4)$ (increased number of iterations) for this method. It is a good approximation to assume that the average number of nodes adjacent to some node is 4. Rarely will any path contain as many nodes as one fourth of the total number of nodes in a typical transportation network, besides, the number of iterations cannot exceed the maximum number of links in any path in the network; therefore, $(N)(N-1)(4)$ (number of iterations) additions and comparisons is less than $(N)(N-1)(N-2)$ additions and comparisons.

The extra computer time incurred by the search of $K(j)$ for each destination node j is very economical with our well coded program, therefore, this algorithm is likely to be more efficient than Floyd's or Dantzig's algorithms.

The Third Algorithm

For some destination j search for one of its adjacent nodes, k_1 , $k_1 \in K(j)$, as an intermediate node. Evaluate the shorter distance from every node i to node j . Next, use $k_2 \in K(j)$ as an intermediate node and evaluate for all $i \in N$. When all $k \in K(j)$ have been used, another destination node j is considered. After every destination node j has been examined, do another iteration if any improvement is detected in this iteration.

The procedure for the third method follows:

1. Set the sequence of node pairs being examined in columnwise order: $(1,1), (2,1), (3,1) \dots (N,1), (1,2), (2,2) \dots (N,2) \dots (1,N), (2,N) \dots (N,N)$.
2. Start with the destination node $j=1$.
3. Find the first adjacent node to j , $k_1 \in K(j)$.
4. Compare d_{ij} with $d_{ik_1} + d_{k_1j}$ for all $i \in N$.

If $d_{ij} > d_{ik_1} + d_{k_1j}$ let $d_{ij} = d_{ik_1} + d_{k_1j}$,

$bn(i,j) = k_1$.

If $d_{ij} \geq d_{ik_1} + d_{k_1j}$, d_{ij} remains unchanged.

5. Find the second node adjacent to j , $k_2 \in K(j)$. Compare d_{ij} with $d_{ik_2} + d_{k_2j}$ for all (i,j) pairs $i \in N$ as in step 4 until every node in $k(j)$ has been used as an intermediate node.

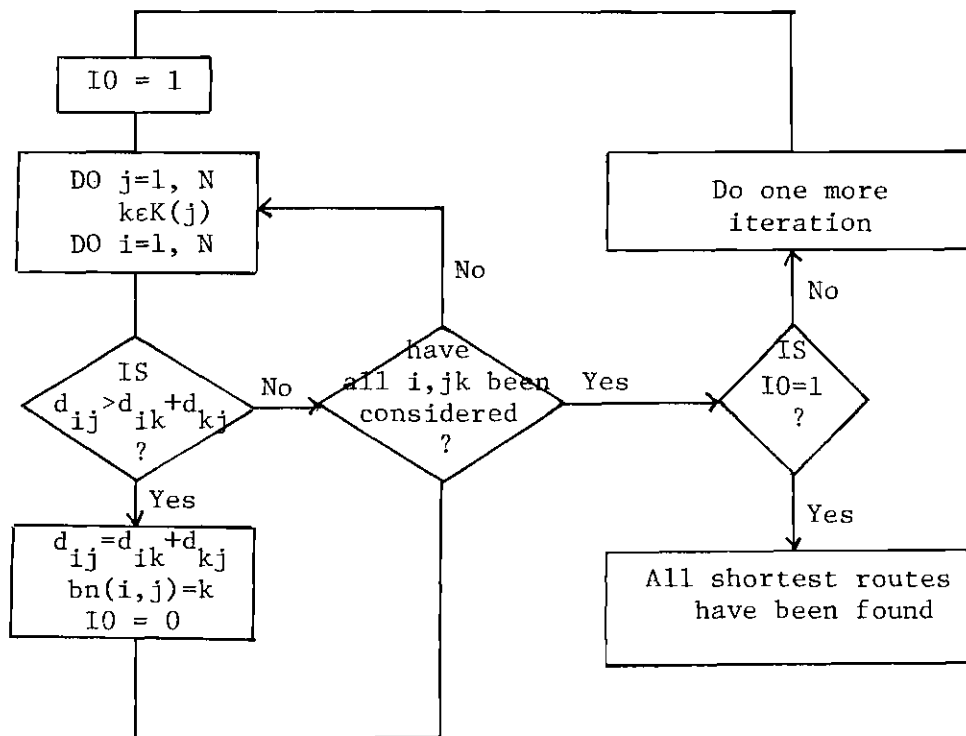


Figure 19. Flow Chart Of The Third New Shortest Path Algorithm.

6. Change destination node to $j+1$.
7. Repeat steps 3, 4, 5 and 6 for every node pair in the sequence.
8. After every node pair (i,j) , $i,j \in N$ has been examined, determine if any shorter distance has been found for any node pair. If so, another iteration is necessary. If not, the d_{ij} 's in the present matrix are the shortest distances for each node pair. The flow chart for this method is shown in Figure 19.

The advantage of this algorithm is that we need not search the nodes adjacent to node j for every (i,j) pair, i.e., there are as many as $(N)(N-1)$ searches per iteration. We search $K(j)$ for each $j \in N$ only once and check all (i,j) , $i \in N$ for each $k \in K(j)$; therefore, only N searches of $K(j)$ are necessary in each iteration.

The Fourth Algorithm

So far, the total number of additions and comparisons needed to determine the shortest paths in the previous algorithm is $(N)(N-1)(4)$ (number of iterations), and the total number of searches for the nodes adjacent to each destination node is $(N) \times (\text{number of iterations})$. Now an attempt is made to improve the algorithm by trying to reduce the number of iterations. Suppose K is an intermediate node from node i to node j . If node pair (i,j) is in front of node pair (i,k) in the examining sequence of node pairs, then d_{ij} can only be determined one iteration later than d_{ik} is determined. That is, one iteration is needed to evaluate d_{ik} ; another iteration is needed to evaluate d_{ij} . Therefore, it would be ideal if the node pairs being evaluated were in such a sequence that each time d_{ik}^* is known prior to the evaluation of d_{ij}^* .

The way the nodes are numbered has a significant influence on the number of iterations needed to find the shortest paths. In a proposed ideal network as in Figure 21, the node numbers are getting bigger as they are farther away from the central node.

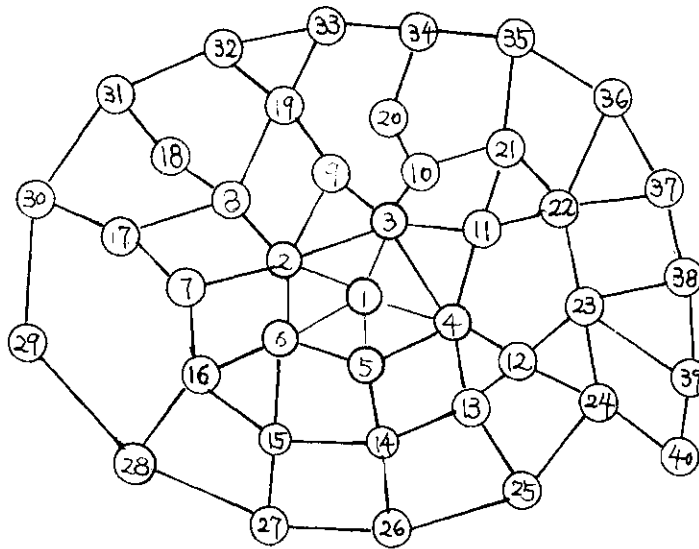


Figure 20. Proposed Ideal Network.

The shortest path from any origin to any destination in this network is very likely to be in one of these patterns:

1. Node numbers on the shortest path form a descending series. For example, the shortest path for (31,1) is 31-18-8-2-1.
2. Node numbers on the shortest path form an ascending series. For example, the shortest path for (2,37) is 2-3-11-22-37.
3. Node numbers on the shortest path form a series that descends first, then ascends. For example, the shortest path for (31,40) is 31-18-8-2-1-4-12-24-40.

We take this case as an example using the previous algorithm to find the shortest path.

The order of evaluating each node pair is set in the matrix in columnwise form, as in Figure 21.

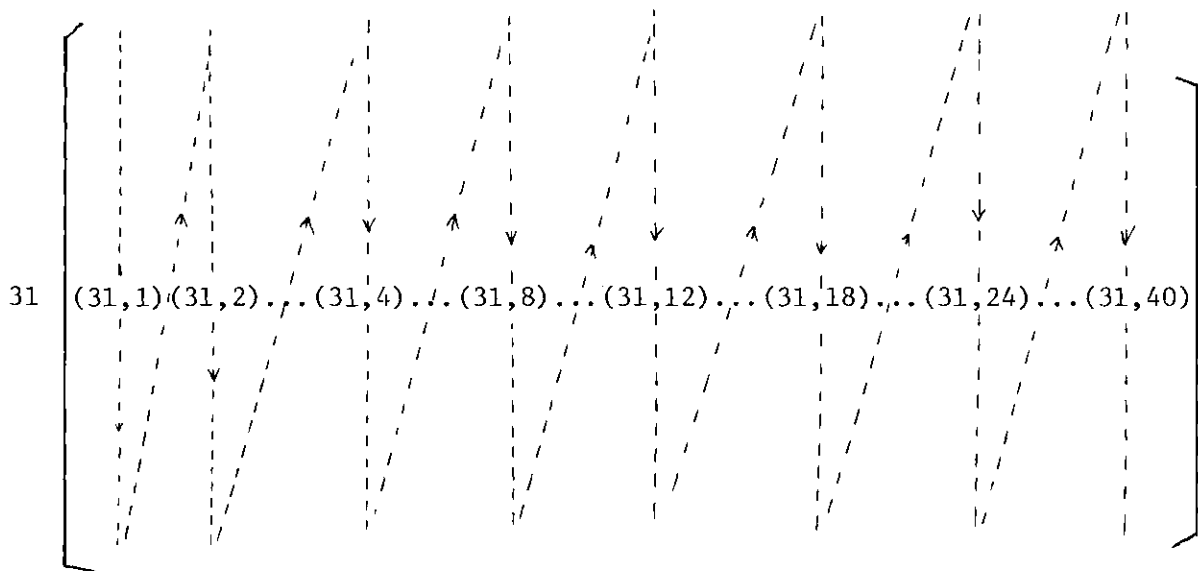


Figure 21. The Positions Of The Node Pairs In The Examining Sequence Of All Node Pairs.

In iteration 0:

$$d_{31,18} = d_{31,18}^*, d_{18,8} = d_{18,8}^*, d_{8,2} = d_{8,2}^*, d_{2,1} = d_{2,1}^*,$$

$$d_{1,4} = d_{1,4}^*, d_{4,12} = d_{4,12}^*, d_{12,24} = d_{12,24}^*, d_{24,40} = d_{24,40}^*.$$

In iteration 1:

1. $d_{31,1} = \infty$, since $d_{31,2}^*$ is not yet known
2. $d_{31,2} = \infty$, since $d_{31,8}^*$ is not yet known
3. $d_{31,4} = \infty$, since $d_{31,1}^*$ is not yet known
4. $d_{31,8}^* = d_{31,18}^* + d_{18,8}^*$
5. $d_{31,12} = \infty$, since $d_{31,4}^*$ is not yet known
6. $d_{31,18} = d_{31,18}^*$
7. $d_{31,24} = \infty$, since $d_{31,12}^*$ is not yet known
8. $d_{31,40} = \infty$, since $d_{31,24}^*$ is not yet known.

In iteration 2:

1. $d_{31-1} = \infty$, since $d_{31,2}^*$ is not yet known
2. $d_{31,2}^* = d_{31-8}^* + d_{8,2}^*$
3. $d_{31,4} = \infty$, since $d_{31,1}^*$ is not yet known
4. $d_{31,8} = d_{31,8}^*$
5. $d_{31,12} = \infty$, since d_{31-4}^* is not yet known
6. $d_{31-18} = d_{31,18}^*$
7. $d_{31-24} = \infty$, since d_{31-12}^* is not yet known
8. $d_{31-40} = \infty$, since d_{31-24}^* is not yet known

In iteration 3:

1. $d_{31-1}^* = d_{31,2}^* + d_{2,1}^*$
2. $d_{31,2} = d_{31,2}^*$

$$3. \quad d_{31,4}^* = d_{31,4}^*$$

$$4. \quad d_{31,8} = d_{31,8}^*$$

$$5. \quad d_{31,12} = d_{31,4}^* + d_{4,12}^*$$

$$6. \quad d_{31,8} = d_{31,18}^*$$

$$7. \quad d_{31,24}^* = d_{31,12}^* + d_{12,24}^*$$

$$8. \quad d_{31,40}^* = d_{31,24}^* + d_{24,40}^*$$

It takes three iterations to find the shortest path.

We now modify the order of evaluating node pairs in such a way that:

1. The node pairs in the lower triangle of the matrix (which are node pairs from large node numbers to small node numbers) are evaluated from right to left, top to bottom.
2. The node pairs in the upper triangle of the matrix (which are node pairs from small node numbers to large node numbers) are evaluated from right to left, top to bottom.
3. This completes one iteration.
4. In the next iteration the order of evaluation is switched to be the opposite of the first.

Using this algorithm to find the shortest path for (31,40), the order of evaluating the node pairs in the lower triangle of the matrix is from right to left for the first iteration; that is (31,1), (31,2), ... (31,4) ... (31,8) ... (31,12) ... (31,18) ... (31,24) ... (31,40).

In iteration 0:

$$d_{31,18} = d_{31,18}^*, d_{18,8} = d_{18,8}^*, d_{8,2} = d_{8,2}^*, d_{2,1} = d_{2,1}^*,$$

$$d_{1,4} = d_{1,4}^*, d_{4,12} = d_{4,12}^*, d_{12,24} = d_{12,24}^*, d_{24,40} = d_{24,40}^*.$$

In iteration 1:

$$1. d_{31,24} = \infty, \text{ since } d_{31,12}^* \text{ is not yet known}$$

$$2. d_{31,18} = d_{31,18}^*$$

$$3. d_{31,12} = \infty, \text{ since } d_{31,4}^* \text{ is not yet known}$$

$$4. d_{31,8} = d_{31,18}^* + d_{18,8}^*$$

$$5. d_{31,4} = \infty, \text{ since } d_{31,1}^* \text{ is not yet known}$$

$$6. \quad d_{31,2} = d_{31,8}^* + d_{31,2}^*$$

$$7. \quad d_{31,1} = d_{31,2}^* + d_{2,1}^*$$

$$8. \quad d_{31,40} = \infty, \quad \text{since } d_{31,24}^* \text{ is not yet known.}$$

In iteration 2:

The order for examining the node pairs is switched; that is:

(31,1) ... (31,2) ... (31,4) ... (31,8) ... (31,12) ... (31,18)
 ... (31,24) and (31,40).

$$1. \quad d_{31,1} = d_{31,1}^*$$

$$2. \quad d_{31,2} = d_{31,2}^*$$

$$3. \quad d_{31,4} = d_{31,1}^* + d_{1,4}^*$$

$$4. \quad d_{31,8} = d_{31,8}^*$$

$$5. \quad d_{31,12} = d_{31,4}^* + d_{4,12}^*$$

$$6. \quad d_{31,18} = d_{31,18}^*$$

$$7. \quad d_{31,24} = d_{31,12}^* + d_{12,24}^*$$

$$8. \quad d_{31,40} = d_{31,24}^* + d_{24,40}^*$$

It takes only 2 iterations to find the shortest path. Thus, we can obtain a new algorithm using a different order to evaluate the node pairs as suggested in this example.

In this algorithm, the procedure for the 1st, 3rd or $(2n+1)$ th iteration is as follows:

1. For node pairs in the lower triangle, fix the sequence of node pairs being examined as $[(N, N-1)], [(N-1, N-2), (N, N-2)], [(N-2, N-3), (N-1, N-3), (N, N-3)], [(N-3, N-4), \dots, (N, N-4)], [(2,1), (3,1), \dots, (N,1)]$. See Figure 22.

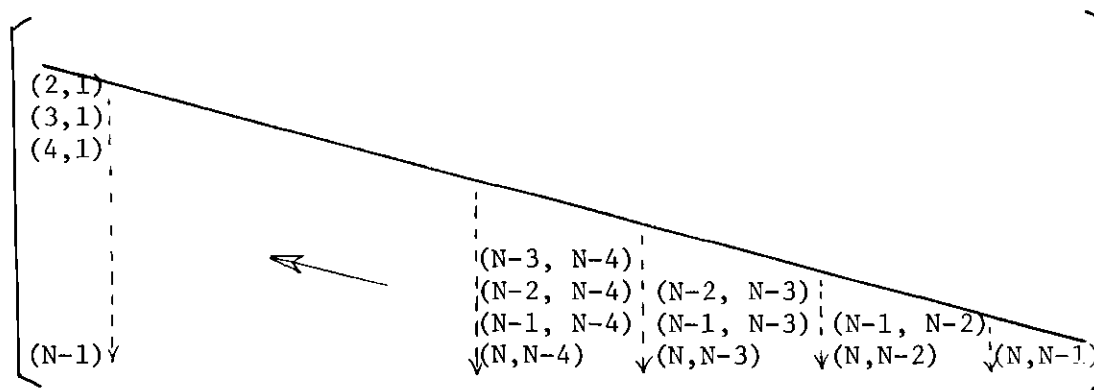


Figure 22. Sequence Of Node Pairs Being Examined (I).

2. Start with the destination node $j=N-1$.
3. Search for one of its adjacent nodes, $k_1 \in K(j)$.

4. Compare d_{ij} with $d_{ik_1} + d_{k_1j}$ for all (i,j) pairs, $i > j$.

If $d_{ij} > d_{ik_1} + d_{k_1j}$, let $d_{ij} = d_{ik_1} + d_{k_1j}$

$bn(i,j) = k_1$.

If $d_{ij} \leq d_{ik_1} + d_{k_1j}$, d_{ij} remains unchanged.

5. Search for another of its adjacent nodes $k_2 \in K(j)$ and compare d_{ij} with $d_{ik_2} + d_{k_2j}$ for all (i,j) pairs $i > j$ as in 4 until every node in $K(j)$ has been used as an intermediate node.

6. Change destination node to $j-1$.

7. Repeat steps 3, 4, 5 and 6 for every node pair from the largest node number to the smallest node number.

8. For the node pairs in the upper triangle, fix the sequence of node pairs being examined as $[(1-N)(2-N) \dots, (N-1, N)] [(1, N-1) (2, N-1) \dots, (N-2, N-1)] [(1, N-2), (2, N-2) \dots, (N-3, N-2)] \dots, [(1,3), (2,3)] [(1,2)]$. See Figure 23.

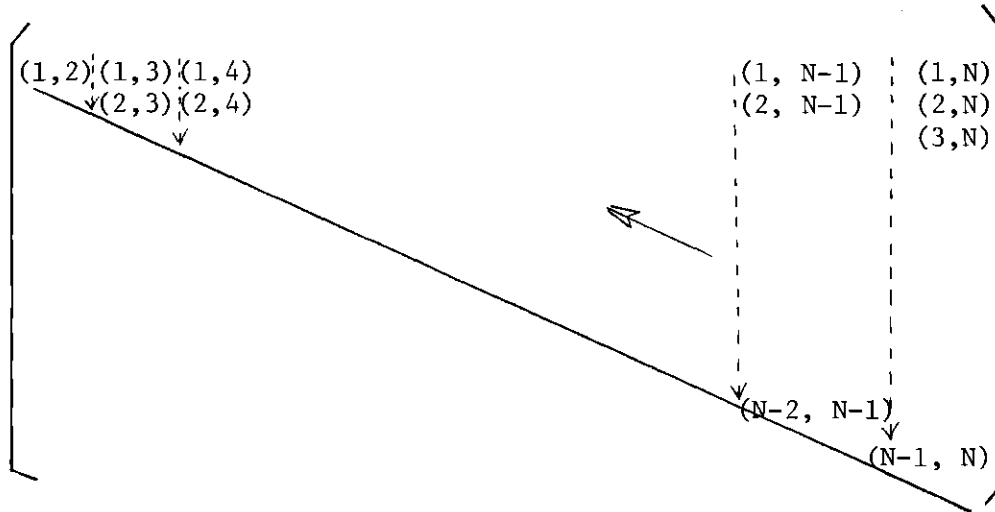


Figure 23. Sequence Of Node Pairs Being Examined (II).

9. Start with the destination node $j=N$.
10. Search for one of its adjacent nodes $k_1 \in K(j)$.
11. Compare d_{ij} with $d_{ik_1} + d_{k_1j}$ for all $i < j$.

If $d_{ij} > d_{ik_1} + d_{k_1j}$, let $d_{ij} = d_{ik_1} + d_{k_1j}$,

$bn(i,j) = k_1$.

If $d_{ij} \leq d_{ik_1} + d_{k_1j}$, d_{ij} remains unchanged.

12. Search for another of its adjacent nodes $k_2 \in K(j)$ and compare d_{ij} with $d_{ik_2} + d_{k_2j}$ for all $i < j$ as in 11 until every node in $K(j)$ has been used as an intermediate node.

13. Change destination node to $j-1$.
14. Repeat steps 9, 10, 11 and 12 for every node pair in the upper triangle.
15. After all node pairs have been examined, determine if any shorter distance has been found for any node pair. If so, another iteration is necessary. If not, the d_{ij} 's in the present matrix are the shortest distances for each node pair.

The procedure for the 2nd, 4th or 2Nth iteration is as follows:

1. For node pairs in the lower triangle fix the sequence of node pairs being examined as $[(2,1), (3,1), (4,1) \dots, (N,1)], [(3,2), (4,2), (5,2), \dots, (N-1, 2), (N,2)], [(4,3), (5,3) \dots, (N,3)] \dots [(N-1, N-2)] [(N, N-1)]$. See Figure 24.

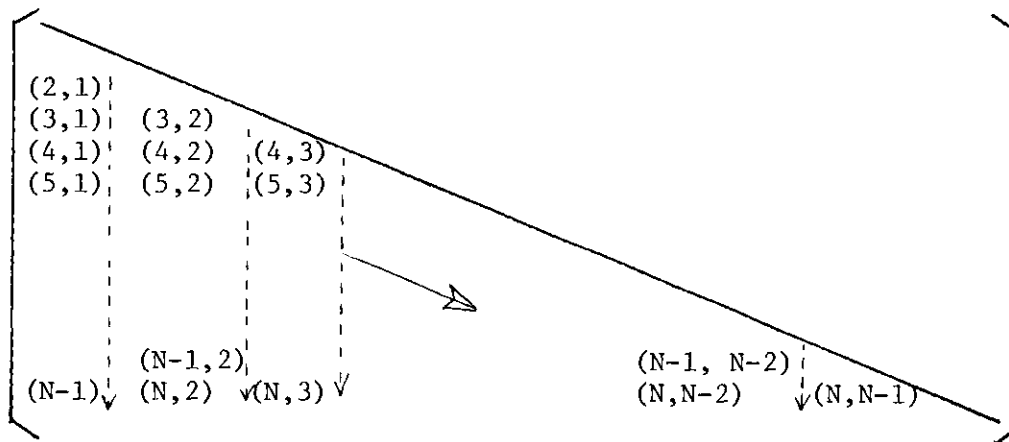


Figure 24. Sequence Of Node Pairs Being Examined (III).

2. Start with the destination node $j=1$.
3. Search for one of its adjacent nodes $k_1 \in K(j)$.
4. Compare d_{ij} with $d_{ik_1} + d_{k_1j}$ for all (i,j) pairs, $i > j$.

If $d_{ij} > d_{ik_1} + d_{k_1j}$, let $d_{ij} = d_{ik_1} + d_{k_1j}$,

$bn(i,j) = k_1$.

If $d_{ij} \leq d_{ik_1} + d_{k_1j}$, d_{ij} remains unchanged.

5. Search for another of its adjacent nodes $k_2 \in K(j)$ and compare d_{ij} with $d_{ik_2} + d_{k_2j}$ for all (i,j) pairs for all $i > j$ as in 4 until every node in $K(j)$ has been used as an intermediate node.
6. Change destination node to $j+1$.
7. Repeat steps 3, 4, 5 and 6 for every node pair in the lower triangle.
8. For the node pairs in the upper triangle, fix the sequence of node pairs being examined as $[(1,2)]$, $[(1,3)(2,3)]$, $[(1,4)(2,4)(3,4)] \dots$, $[(1, N-1), (2, N-1) \dots, (N-2, N-1)] [(1,N), (2,N), (3,N) \dots (N-1, N)]$. See Figure 25.

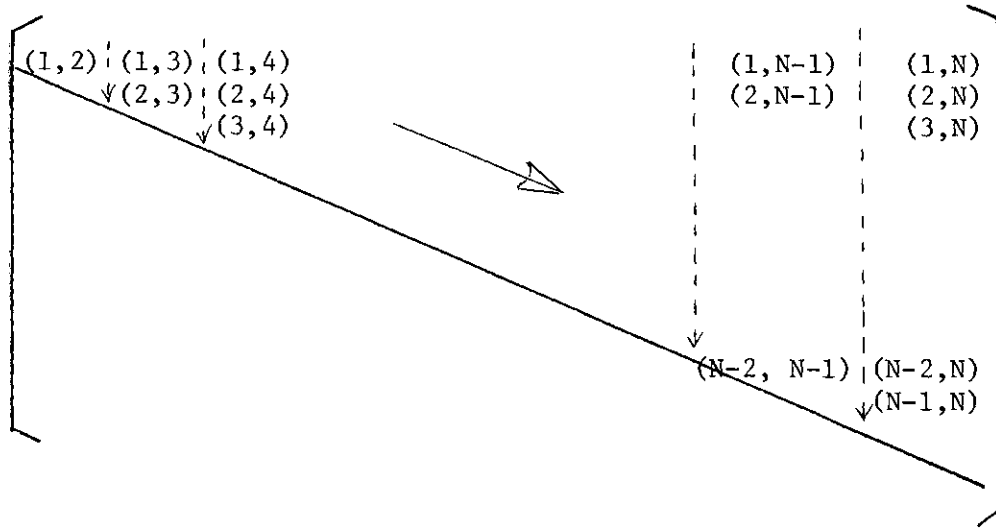


Figure 25. Sequence Of Node Pairs Being Examined (IV).

9. Start with the destination node $j=1$.
10. Search for one of its adjacent nodes $k_1 \in K(j)$.
11. Compare d_{ij} with $d_{ik_1} + d_{k_1j}$ for all $i < j$.

If $d_{ij} > d_{ik_1} + d_{k_1j}$, let $d_{ij} = d_{ik_1} + d_{k_1j}$,

$bn(i,j) = k_1$.

If $d_{ij} \leq d_{ik_1} + d_{k_1j}$, d_{ij} remains unchanged.

12. Search for another of its adjacent nodes $k_2 \in K(j)$ and compare

d_{ij} with $d_{ik_2} + d_{k_2j}$ for all i, j as in 11 until every node in $K(j)$ has been used as an intermediate node.

13. Change destination node to $j+1$.

14. Repeat steps 9, 10, 11 and 12 for every node pair in the upper triangle.

15. After all node pairs have been examined, determine if any shorter distance has been found for any node pair. If so, another iteration is necessary. If not, the d_{ij} 's in the present matrix are the shortest distance for each node pair.

The flow chart of the algorithm is given in Figure 26.

The node numbers in a shortest path can be divided into several ascending and descending sequences, as in Figure 27.

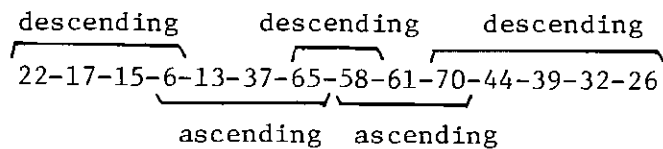


Figure 27. Descending And Ascending Sequences
Of The Node Numbers In A Shortest Path.

The advantage in switching the examination sequence of node pairs is that the distances from the origin node to the nodes in the same sequence can be obtained in the same iteration. Therefore, the number of iterations needed to find the shortest path for all node pairs is about the maximum number of sequences in any of the shortest paths. In this manner generally we reduce the number of iterations necessary to find the shortest paths.

Conclusion

Both this algorithm and the algorithm of Floyd have been programmed to run on a test network of 97 nodes with 342 one-way links. The node numbers in this network have been arranged according to our proposed rule. Only 6 iterations are needed to find the shortest path for each node pair. It takes 6 cpu seconds for this new algorithm and 15 cpu seconds for Floyd's algorithm. This supports the claim that the new algorithm is likely to be more efficient than other matrix algorithms that find the shortest path for all node pairs.

CHAPTER VI

APPLICATION OF THE STEPWISE ASSIGNMENT MODEL TO
DERIVE THE TRAFFIC DISTRIBUTION OF COLUMBUS - PHENIX CITY

The stepwise assignment model has been described to reasonably reflect traffic distribution. An efficient algorithm has also been devised for the implementation of this model. In order to apply the stepwise assignment to a real-world transportation system, the following information about the system has to be determined:

1. the network of the system
2. the trip demands
3. the cost functions.

The Network

The expressways, city arteries and streets are to be transformed into the network in abstract form. The resulting network is described in a number of links and nodes. A node may represent an intersection or an aggregation of a transport zone; a link represents the main traffic artery in the road network, sometimes representing the combination of several roads serving the same direction. Since the number of nodes or links has direct influence on the computation time, it is desired to have as few nodes and links as possible, while still being able to reflect the traffic behavior realistically.

The aggregated network of Columbus, Georgia and Phenix City, Alabama consists of 97 nodes and 342 one-way links, as shown in Figure

28. The major arterial routes of Columbus-Phenix City are shown in Figure 29. The arterial routes represented by our aggregated network are shown in Figure 30. Table 2 lists the names of the roads on each arterial route and the corresponding links in the network. Based on the particular road system of Columbus-Phenix City, we have grouped the real roads, represented by each link in the network, into eleven categories. The number of lanes in each category is represented by a one-dimensional array, A9, which has eleven elements. This relationship is displayed in Table 3. The one-dimensional array, M8, with 342 elements, represents the category of road for each link in the network. The road types of all links are given graphically in Figure 31.

The nodes in the network have been numbered according to our proposed principle, [See Figure 28]. Node 1 is assigned to the area about the center of the network where many arteries converge. As the node number increases the nodes are assigned farther away from node 1. The assignment of the node number is determined by the following rule: the sequence of node numbers in any path are either ascending or descending, but do not alternate too many times. See Figure 1. For example, the shortest path from node 96 to node 86 in the network may be:

96-50-28-27-19-8-2-1-6-15-45-46-80-86 the shortest path from node 84 to
 |-----|-----|
 descending ascending

91 may be: 84-74-73-62-42-41-30-16-6-5-4-11-13-25-35-52-91. It is
 |-----|-----|
 descending ascending

believed that these efforts will reduce the number of iterations needed to find the shortest paths.

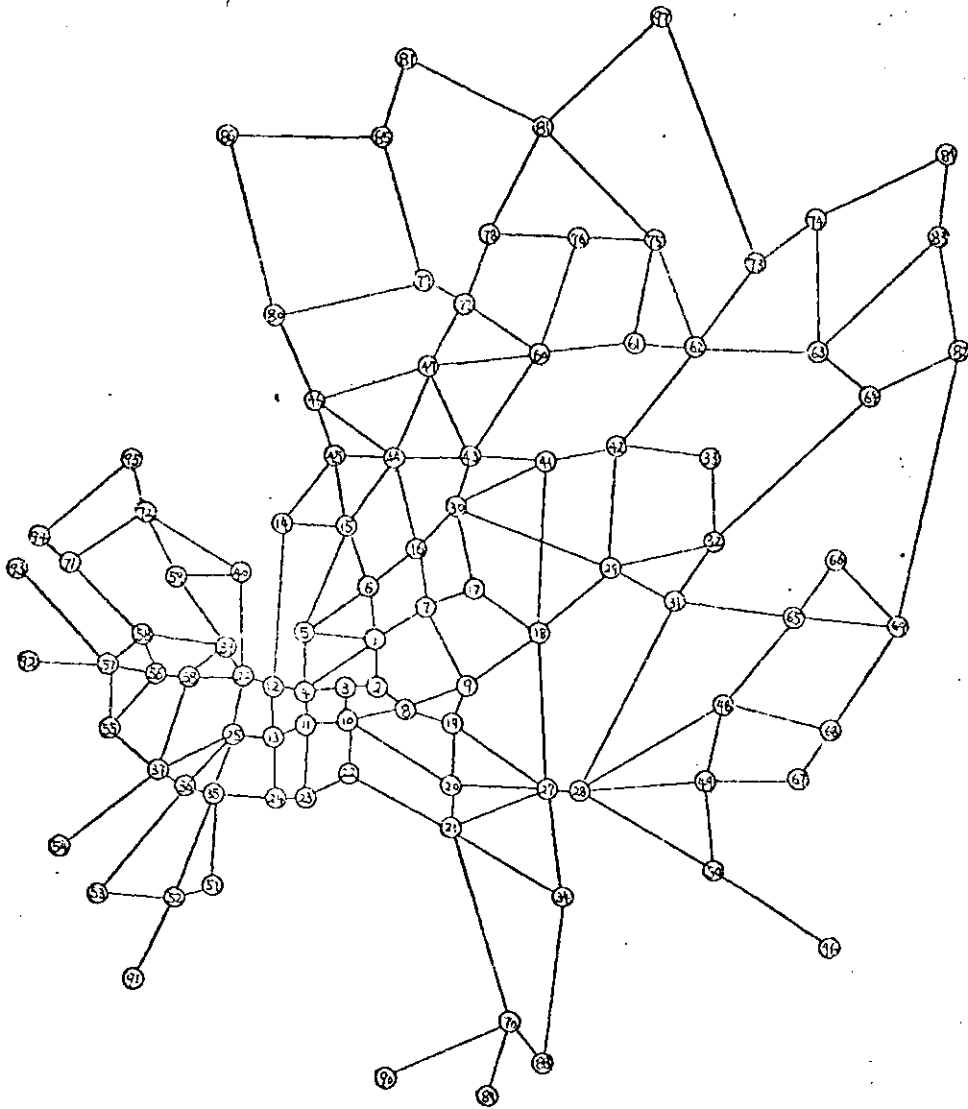


Figure 28. The Aggregated Network Of Columbus-Phenix City.

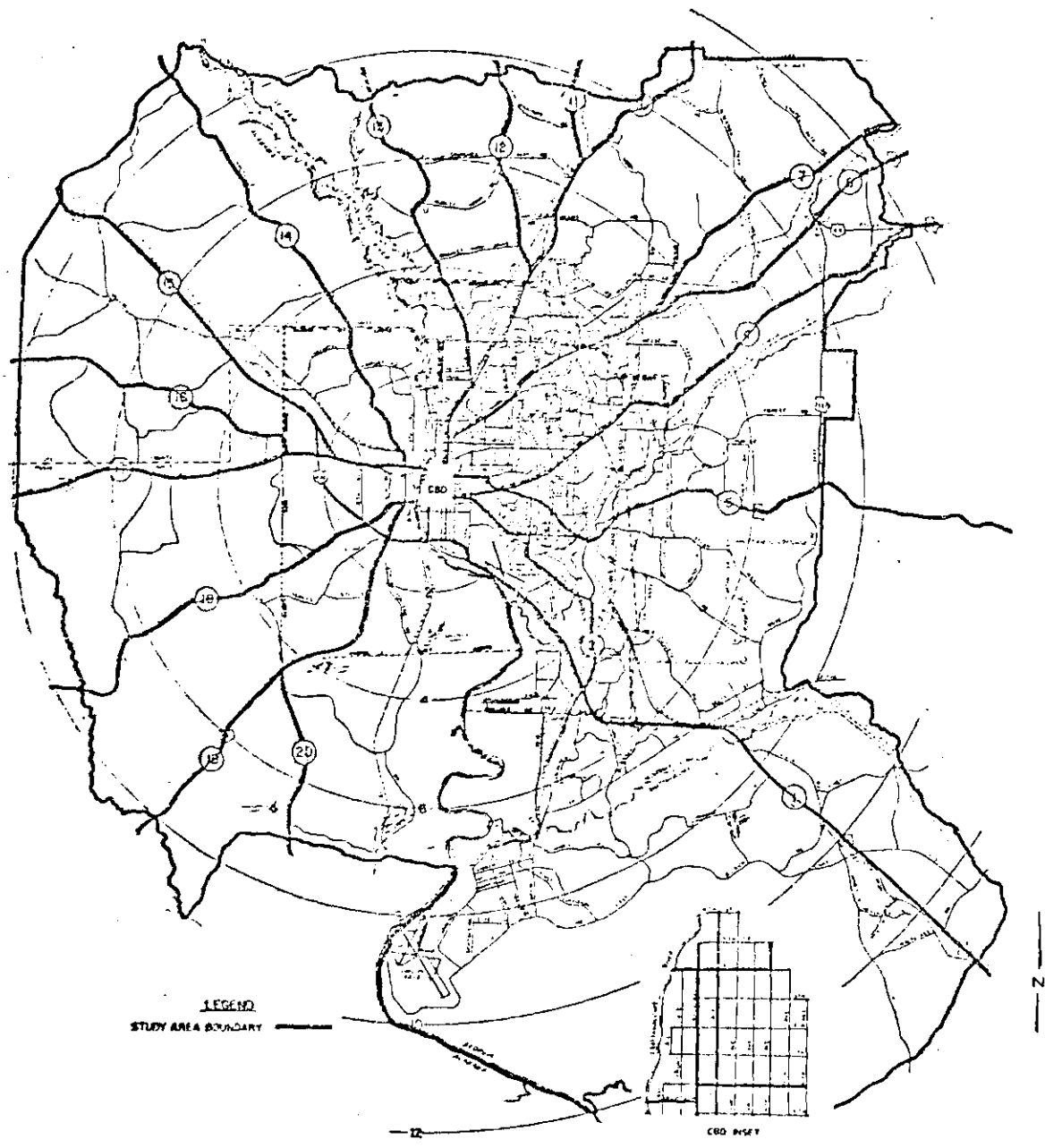


Figure 29. Major Arterial Routes Of Columbus-Phenix City.

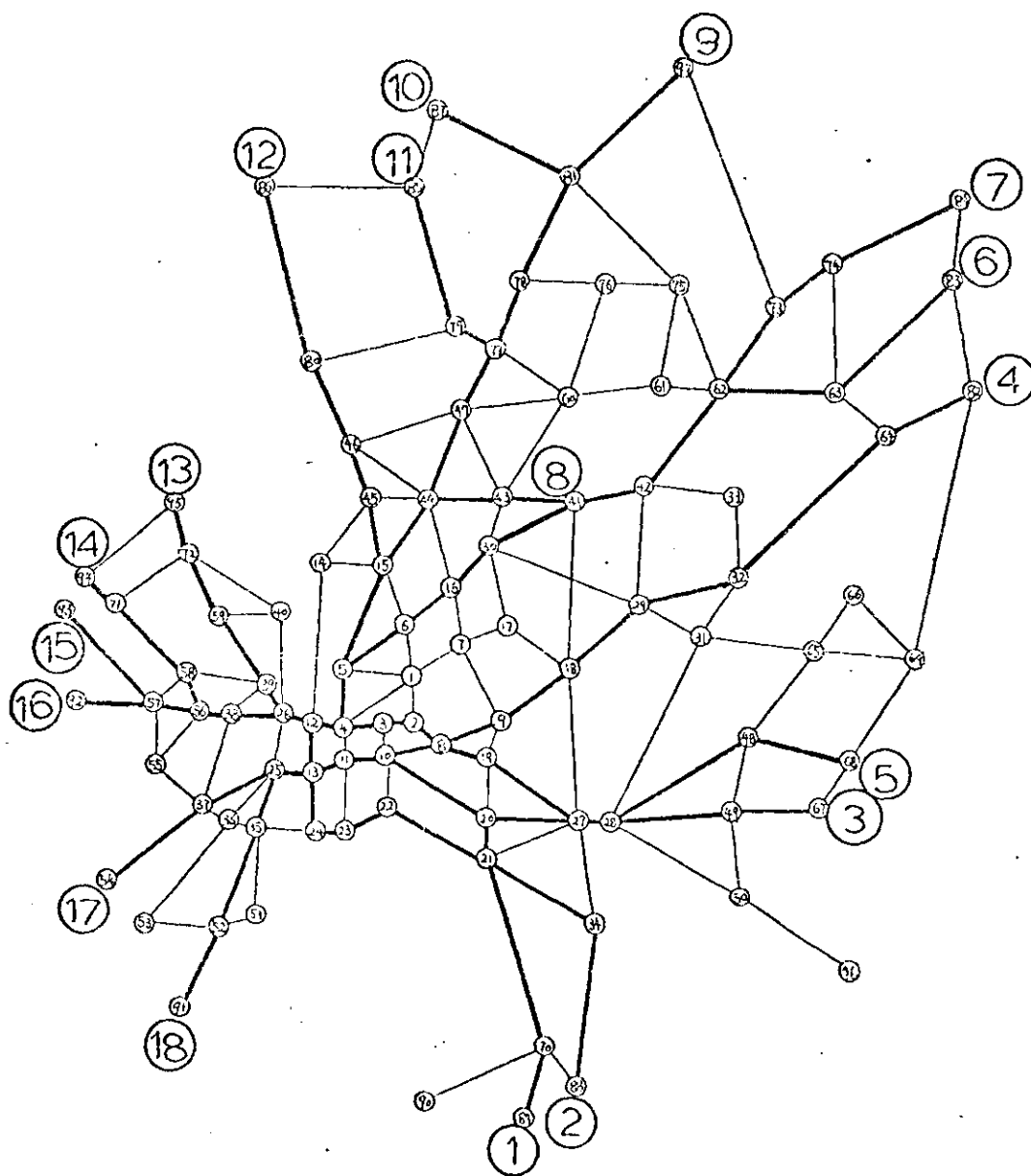


Figure 30. Major Arterial Routes Represented By The Networks.

Table 2. Route Description And Node Sequences
Representing The Major Arterial Routes Of Columbus-Phenix City

Route No.	Route Description From Broadway & 12th (Mi.)	Node Sequence
1.	Broadway - Victory Dr.	12-13-24-23-22-21-70-89
2.	Broadway - 10th St. - Brookhaven Blvd. - Cusseta Rd. - Fort Benning Rd.	12-13-11-10-20-21-34-88
3.	Broadway - 10th St. - Brookhaven Blvd. - St. Mary's Rd.	12-13-11-10-20-27-28-49-67
4.	Broadway - 10th St. - 10th Ave. - Wynnton Rd. - Macon Rd.	12-13-11-10-8-9-18-29-32-64-82
5.	Broadway - 13th St. - Buena Vista Rd.	12-4-3-2-8-11-27-28-48-68
6.	Broadway - 14th St. - 4th Ave. - Warm Springs Rd. - Ga. 85	12-4-5-6-16-30-41-42-62-63-83
7.	Broadway - 14th St. - 4th Ave. - Warm Springs Rd.	12-4-5-6-16-30-41-42-62-73-74-8
8.	Broadway - 14th St. - 4th Ave. - Hamilton Rd. - Columbus	12-4-5-15-44-43-41
9.	Broadway - 14th St. - 4th Ave. - Ga. 85 - U.S. 27 (Ga. 1)	12-4-5-15-44-47-77-78-81-97
10.	Broadway - 14th St. - 4th Ave. - Ga. 85 - U.S. 27 (Ga. 1) - S633	12-4-5-15-44-47-77-78-81-87
11.	Broadway - 14th St. - 4th Ave. - Ga. 85 - U.S. 27 (Ga. 1) - S633	12-4-5-15-44-47-77-79-85
12.	Broadway - 14th St. - 4th Ave. - Ga. 85 - Ga. 103	12-4-5-15-45-46-80-86
13.	Broadway - 14th St. - Sommer- ville Rd.	12-26-39-49-72-95
14.	Broadway - 14th St. - U.S. 431	12-26-38-56-58-71-94
15.	Broadway - 14th St. - Crawford Rd. - Stafford	12-26-38-56-67-93

Table 2 (Cont.)

Route No.	Route Description From Broadway & 12th (Mi.)	Node Sequence
16.	Broadway - 14th St. - Crawford Rd.	12-26-38-56-57-92
17.	Broadway - Dillingham St. - Sandfort Rd.	12-13-25-37-54
18.	Broadway - Dillingham St. - Seale Rd.	12-13-25-35-52-91

Table 3. Types Of Roads, Numbers Of Zones And Names Of Speed-Flow Functions Of Each Category Of Roads.

Category	Type of Roads	Number of Lanes	Name of Speed-Flow Functions
1	2	A9(1) = 1	F2
2	2B	A9(2) = 1	F2B
3	4	A9(3) = 2	F4
4	4B	A9(4) = 2	F4B
5	6	A9(5) = 3	F6
6	2, 2	A9(6) = 2	F2
7	2B, 2B	A9(7) = 2	F2B
8	4, 4	A9(8) = 4	F4
9	6, 2, 2	A9(9) = 5	F6
10	4, 6	A9(10) = 5	F6
11	2, 4	A9(11) = 3	F6

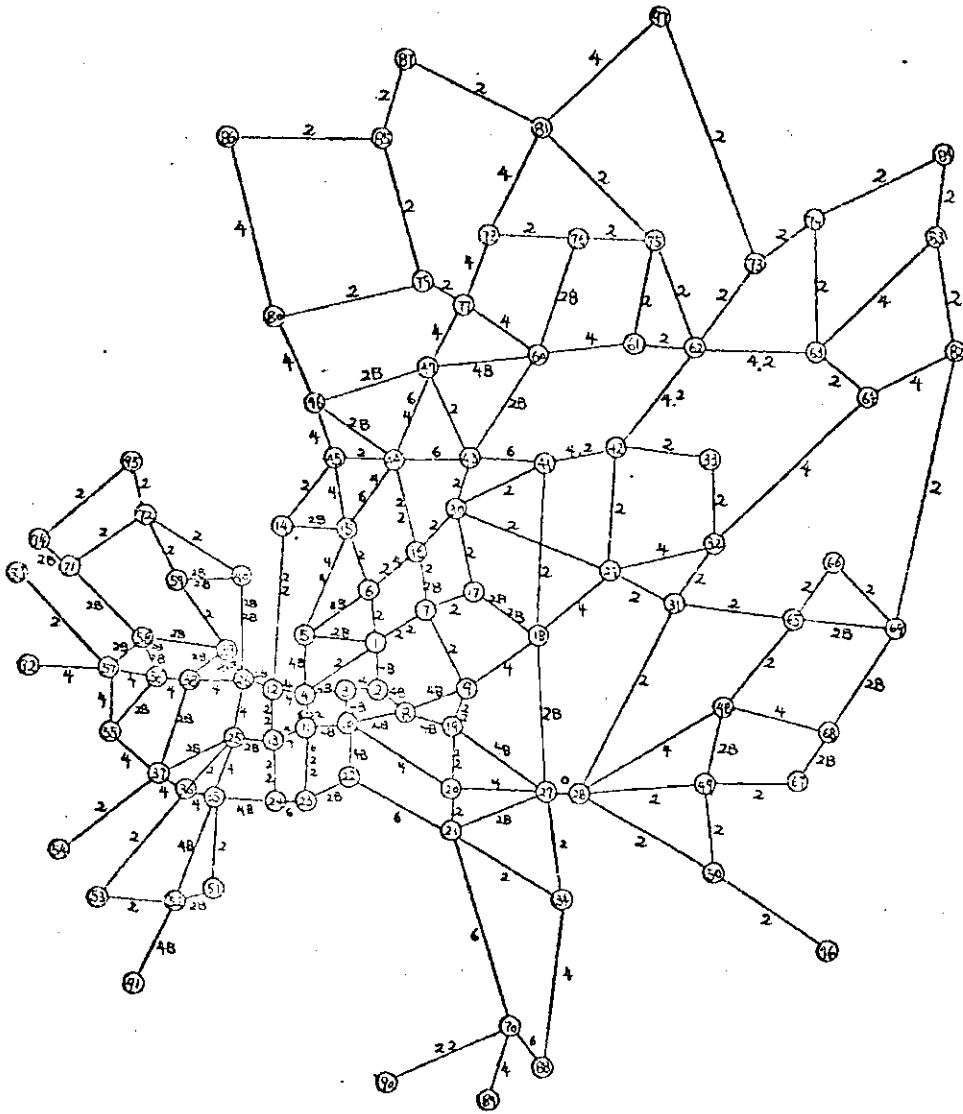


Figure 31. Type of Roads That Each Link Represents.

The Trip Demands

Since the flow pattern is different at different periods of the day, describing the flow distribution in terms of total daily demand would not be logical. Assuming that 1/10 of the total daily trip is uniformly distributed in the morning peak hour at a steady state flow rate, we can use the stepwise assignment algorithm to find the steady state flow pattern of this system. Since the flow rate on the link is accumulating at each step, and not the flow volume, the relationships between travel speed and flow rate on various links must be determined.

The Cost Functions

Travel time, operating expenses or simply the ignorance or prejudice of the travelers to the routes are possible factors involved in the route decision-making process. The cost defined in this study represents only the travel time, which is believed to be the predominant factor affecting traffic distribution.

Cost On Nodes

The delay caused by traffic signals in intersections is one of the most important features in urban transportation. The "average delay per vehicle" passing through an intersection is heavily dependent on the green light period and the flow rate.

Our abstract network represent aggregated roads and intersections. Many intersections which really exist in the road network may have been neglected by our "link." Sometimes, several intersections are aggregated to form a "node." In order to reflect the actual delay effect in the intersections the following assumptions are made:

1. All nodes in the network represent intersections.
2. The average delay for each vehicle in passing through an intersection depends on the red period in his direction and the rate of flow coming into the node in his direction.
3. Progressive signal systems are assumed for the intersections being omitted on the links; therefore, their effects are negligible.
4. All intersections have thirty-second red and green light periods.
5. In order to increase the effect of the node upon traffic flow, 10% right turns and 10% left turns are assumed for each intersection.
6. For a 30 second red period traffic system, 550 vehicles per hour per lane is about the capacity of the link. The delay for flow rates beyond 550 vehicles per hour per lane is assumed to be infinite.
7. Webster's delay equation^[14] is used to relate delay and flow rate. The graphical representation of this relationship for various flow rates under 10% right and 10% left turns is in Figure 32.

Cost On Links

Unlike the case of uninterrupted flow, the data about flow-speed relationships for urban streets are very rare. According to the research by L. C. Edie and R. S. Foote^[6] in the Holland Tunnel, a large sample of approximately 24,000 vehicles was analyzed to establish a relationship between speed and spacing, as in Table 4. The rationale is that

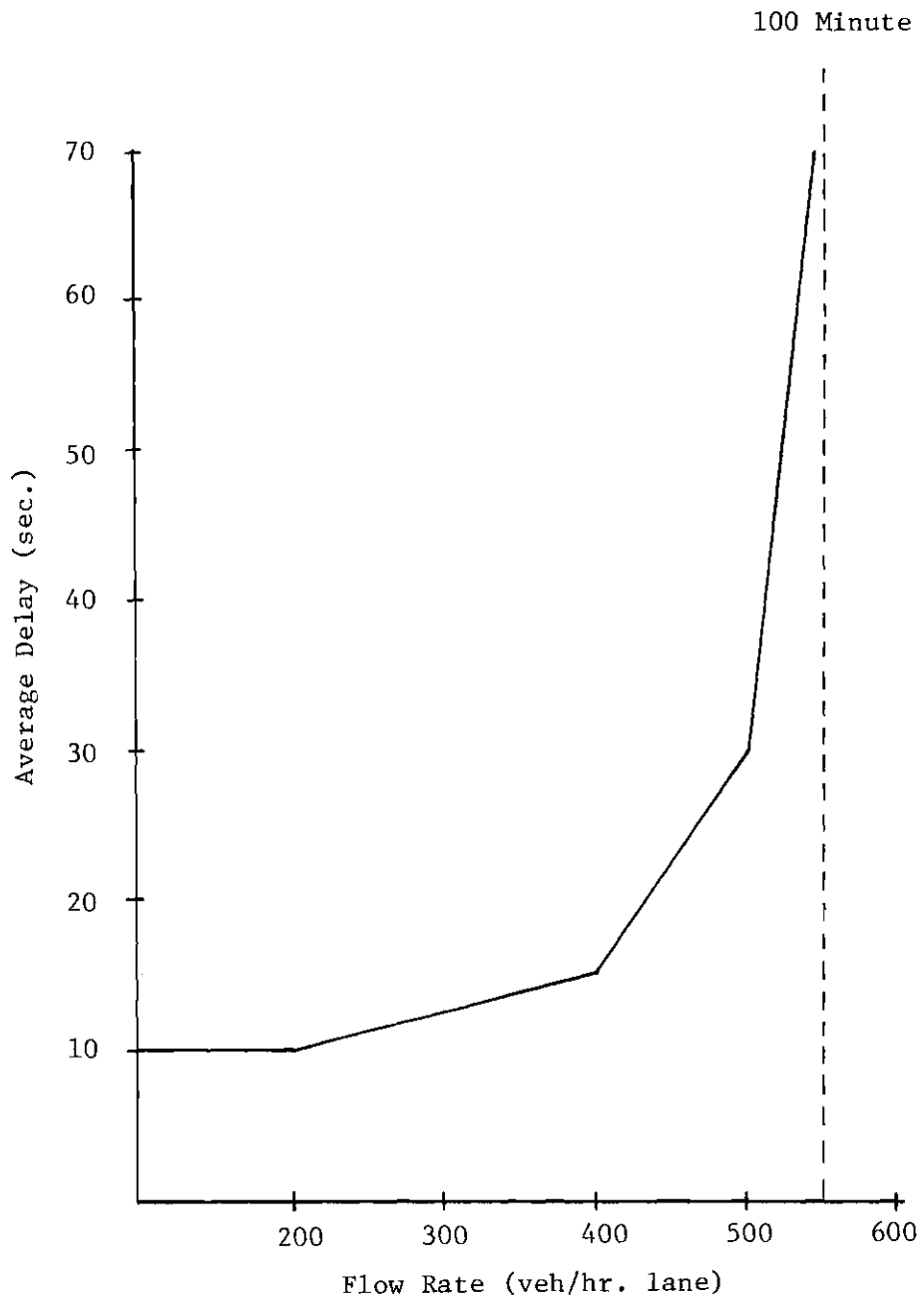


Figure 32. Delay On Intersection For 10%
Right - 10% Left Turns.

Table 4. Relationship Among Speed Spacing,
Concentration And Flow Rate For Single-Lane Uninterrupted Flow.

Speed L(ft./sec.)	Speed V(mile/hr.)	Average Spacing S(ft./car)	Concentra- tion K(car/mile)	Flow Rate q(car/hr.)
21	14.3	65.6	80.4	1151
25	17.0	74.0	70.5	1201
29	19.8	84.6	62.4	1233
33	22.5	92.4	57.2	1287
37	25.2	107.2	49.3	1243
41	27.9	112.6	43.1	1205
45	30.7	143.4	36.8	1129
49	33.4	181.5	29.1	972
53	36.1	201.5	26.2	947
57	38.9	250.4	21.1	820
61	41.6	257.7	23.5	853
65	44.3	334.7	15.8	700

the steady state mean spacing selected by a driver is determined primarily on the basis of the driver's speed. The average spacing (ft/car) gives the average distance between each car; therefore, we may determine the concentration (car/mile). After the speed and the concentration are known, we may determine the flow rate (car/hr.). Thus, we have established the relationship between speed and flow rate, which is plotted in Figure 33. The relationship between speed and concentration is given in Figure 34. The relationship between flow rate and concentration is given in Figure 35. These relationships are usually true for flows on any single lane, urban street under uninterrupted ideal conditions. The curves assumed for the speed-flow relationship, for multilane roads are given in Appendix 1.

According to the 1965 Highway Capacity Manual^[8], "Red periods reduce the amount of traffic that can be accommodated in a clock hour in approximate proportion to their percentage of the total time." We assumed that each intersection has thirty second red periods; therefore, for interrupted speed-flow relationships, the scale for the uninterrupted ideal speed-flow relationship is reduced by one-half, while the shape remains the same.

Results And Discussion Of Results

Computation Time

The cpu time for a single step assignment is about 15 seconds, for 5-step assignments about 45 seconds, and for 10-step assignments about 95 seconds.

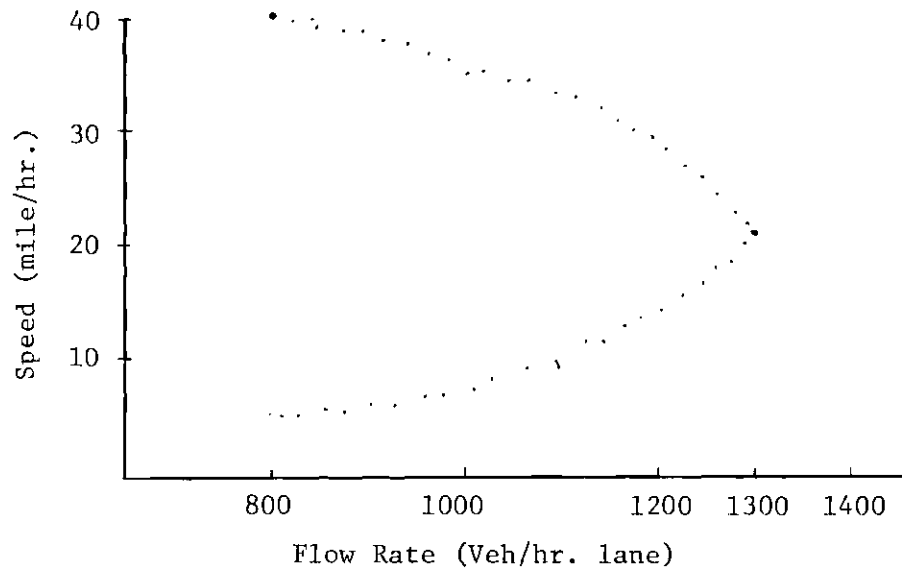


Figure 33. Speed Versus Flow Rate For Single-Lane Uninterrupted Flow.

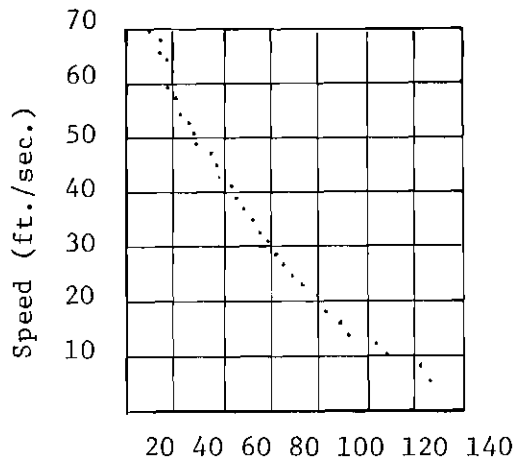


Figure 34. Speed Versus Concentration For Single-Lane Uninterrupted Flow.

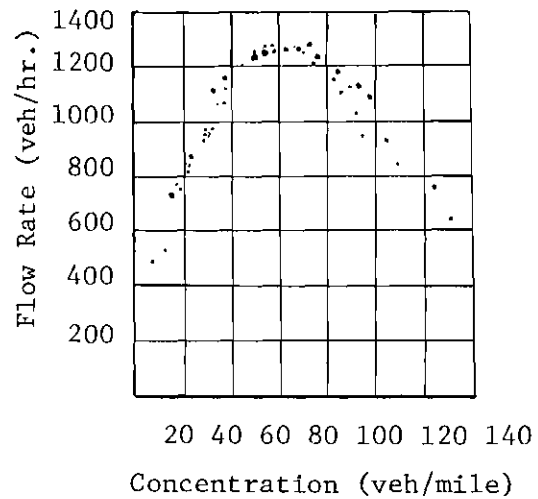


Figure 35. Flow Rate Versus Concentration For Single-Lane Uninterrupted Flow.

Memory Requirements

There are two (97x97) arrays for the cost matrix and back node matrix. There is one (1x97) array for M1, one (1x342) array for M2, and one (1x342) array for MD to describe the network data. Similarly, there is one (1x97) array for M4, one (1x4128) array for M5, and one (1x4128) array for M6 to describe the demand data. There is also one (1x578) array for assigning flow to the links and one (1x342) array for indicating the type of road in each link.

Comparison Of Flow Distribution For 1-Step, 5-Step And 10-Step Assignments

The flow distribution of 1-step, 5-step and 10-step assignments are presented in Figures 36, 37 and 38, respectively.

The number of links in each range of flow for 1-step, 5-step and 10-step assignments are given in Table 5.

Table 5. Number of Links in Each Range of Flow For 1-Step, 5-Step and 10-Step Assignments.

Range of Flow	1-Step Assignment	5-Step Assignment	10-Step Assignment
800<x	8	2	0
700<x<800	2	5	0
600<x<700	7	12	18
500<x<600	16	29	29
400<x<500	19	18	25
x<400	288	274	268

* Link (27,28) is a dummy link, the flow on it is not considered.

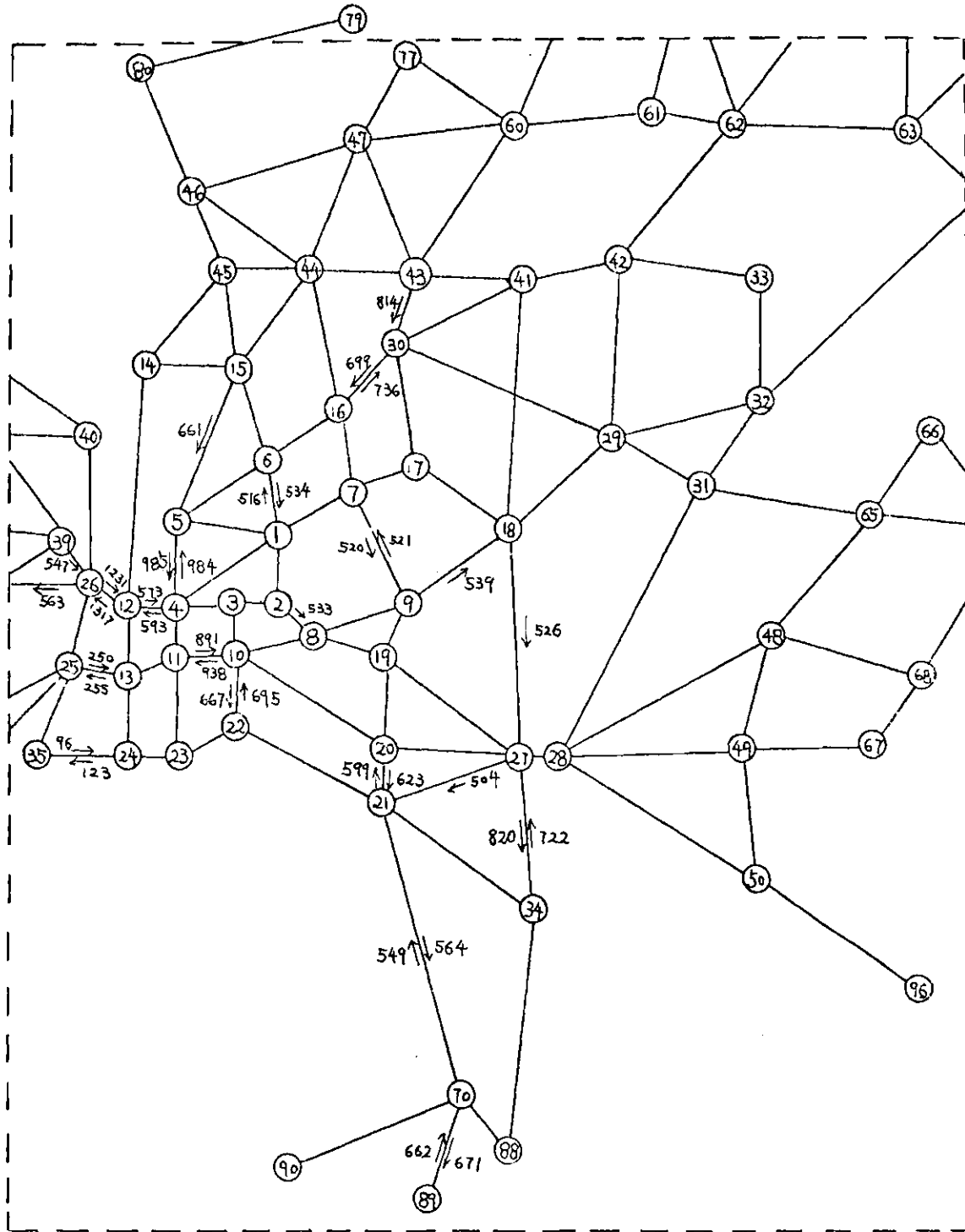


Figure 36. Flow Distribution of 1-Step Assignment.

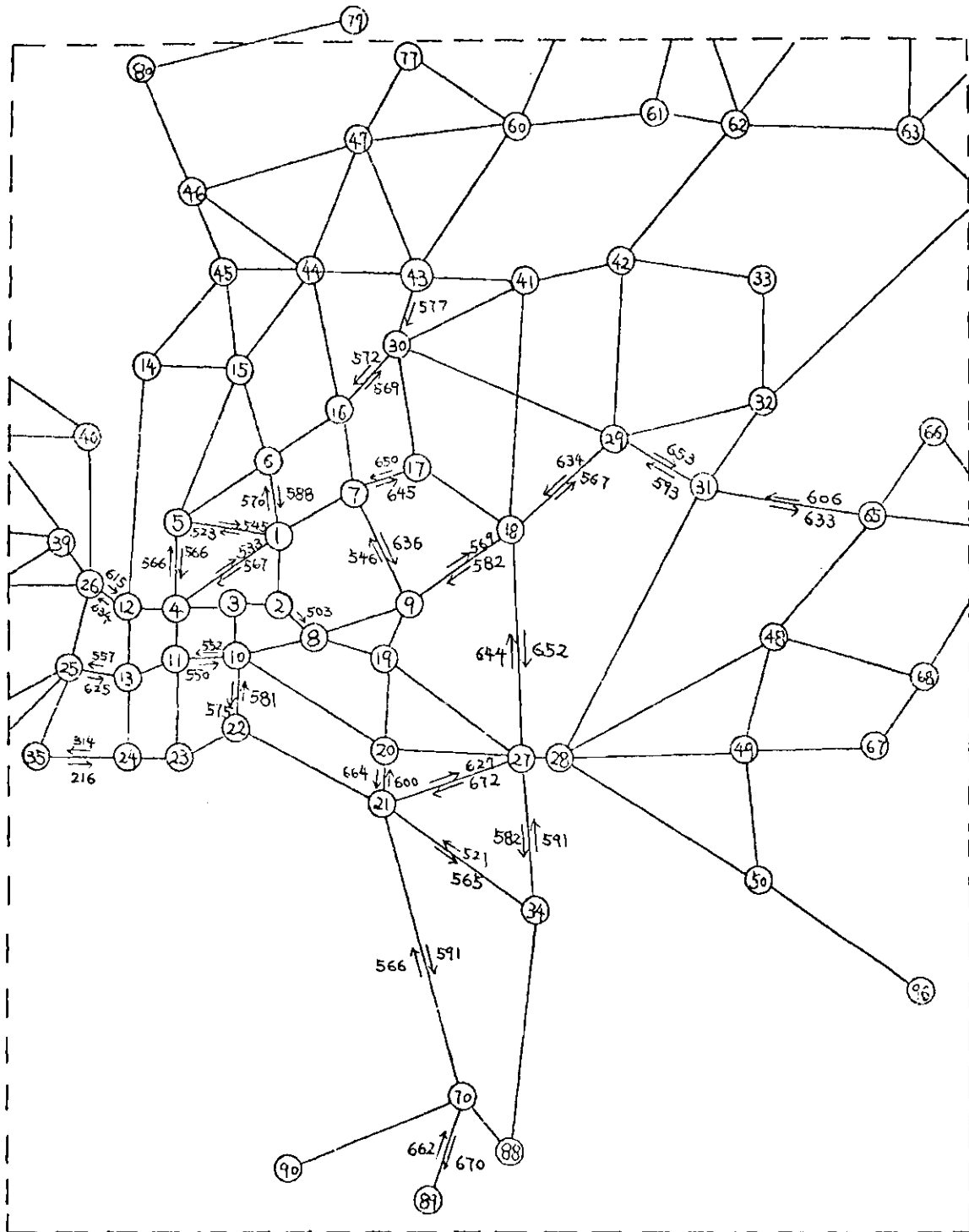


Figure 38. Flow Distribution Of 10-Step Assignment.

For 1-step assignment, eight links exceed 800 (veh/hr. lane). For 5-step assignments, only two links are over 800 (veh/h4. lane). For 10-step assignments, the flow rate on each link is below 700 (veh/hr. lane). The results show that the effect of capacity restraint works well as the steps increase.

The number of links in the range $600 < x < 700$ for 10-step, 5-step and 1-step assignments satisfy $18 > 12 > 7$. For $500 < x < 600$, the relation is $29 > 29 > 16$. For $400 < x < 500$, the relation is $25 > 18, 25 > 19$. For the 10-step assignments more links are to be assigned to the range of $400 < x < 700$, which is approximately the capacity of each link.

The number of links in the range $x < 400$, which is far below the capacity, satisfies the relation $268 < 274 < 288$. For 10-step, 5-step and 1-step assignments the number of links in this range is getting less as the number of steps increase. It is the purpose of stepwise assignments to assign flows to the network within the limit of each link's capacity. Flow which is over the capacity of some links is assigned to those uncongested links; therefore, more links are in the range near capacity and fewer links in the range far below or far above the capacity.

Discussion Of The Trip Distribution Among The Three Bridges Connecting Columbus and Phenix City

In the morning peak hour, traffic demand included a total of 1672 trips from Phenix City to Columbus and 1818 trips from Columbus to Phenix City. Only three bridges cross the Chattahoochee River: located at Fourteenth Street, Dilling Ham and Fourth Street. Fourth Street is a four-lane road but with flow restrictions. Fourteenth Street and Dilling Ham are two-lane roads with flow restrictions. All trips are distributed

on the three links of (26,12), (25, 13) and (35,24) for each step of assignment.

The trips from Phenix City to Columbus are distributed among links (26,12), (25,13) and (35,24) as given in Table 6. The trips from Columbus to Phenix City are distributed among links (12,26), (13,25) and (24,35) as given in Table 7.

Table 6. Flow Distribution Among Link (26,12), (25,13) And (35,24) Under 1-Step, 5-Step And 10-Step Assignments.

Link	1-Step Assignment		5-Step Assignment		10-Step Assignment	
	veh/hr. lane	veh/hr.	veh/hr. lane	veh/hr.	veh/hr. lane	veh/hr.
(26,12)	1231	1231	737	737	615	615
(25,13)	250	250	687	687	625	625
(35,24)	96	192	124	248	216	432
Total Trip		1673		1672		1672

Table 7. Flow Distribution Among Links (12,26), (13,25) and (24,35) Under 1-Step, 5-Step And 10-Step Assignments

Link	1-Step Assignment		5-Step Assignment		10-Step Assignment	
	veh/hr. lane	veh/hr.	veh/hr. lane	veh/hr.	veh/hr. lane	veh/hr.
(12,26)	1317	1317	740	740	634	634
(13,25)	255	255	705	705	557	557
(24,35)	123	246	186	372	314	628
Total Trip		1818		1817		1819

For the one-step assignment, most travelers would use link (26,12) because it is on the shortest path of many O-D pairs. As the steps increase, the trips are suppressed to use link (25,13) and (25,24), which makes the flow on these three links within the reasonable range of their capacity.

Number of Iterations To Find The Shortest Path

The number of iterations needed to find the shortest paths for each step of the assignment is listed in Table 8.

Table 8. The Number of Iterations Needed To Find The Shortest Paths For Each Step Of The 1-Step, 5-Step And 10-Step Assignments.

1-Step Assignment		5-Step Assignments					
Step	1st	Step	1st	2nd	3rd	4th	5th
No. of Iteration	7	No. of Iteration	7	7	6	8	8

10-Step Assignments										
Step	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
No. of Iteration	7	7	7	7	6	8	8	8	9	9

Based on their geographic relationships, the nodes are numbered in accordance with our proposed rule. Only a few additional iterations are needed after the cost on the links have been changed.

Criticism Of Assumptions

It is not a realistic result to have so many links assigned to the margin of their capacity. One factor for this result is that our assumption of 1/10 total daily demand for the flow rate in the morning peak hour is overestimated. Another factor can be that our assumption of thirty second red and green periods for each intersection reduces the capacity of many arteries, which usually have red periods less than thirty

seconds, and increases the capacity of many secondary streets, which usually have red periods more than thirty seconds.

Comparison Of The Results From Ten-Step Assignment
And The Columbus-Phenix City Transportation Study^[3]

This study has been modeled as a network of 440 nodes and 3050 one-way links.

Comparison Of Congestion Conditions. Most congested links in the Columbus-Phenix City Transportation Study (CPTS) are congested in the ten-step assignment, e.g., two-way links (12,26), (13,25), (4,5), (1,4), (6,1), (17,7), (18,27), (29,21) and (18,29).

Those links which are congested in our results but are not congested in CPTS are two-way links such as (29,24), (20,21), (7,9), (18,9) and (30,16). We believe that an important factor might be that our demand data are slightly different from those used in CPTS. For example, links (29,34), (20,21) and (30,16) are over-assigned in the one-step assignment, which means that these links are on the shortest paths of many O-D pairs. Therefore, they are very likely to be congested under our demand information.

Links (47,44), (44,15), (15,5), (45,15), (44,43) and (47,18) are somewhat congested in CPTS but not congested in our results.

Comparison of Travel Time. In our algorithm, when traffic flow is more than 550 (veh/hr. lane) on some link, an infinite cost is incurred in this link to prevent any further assignment. Therefore, the travel time on these congested links is not significant. A few non-congested links are compared for the travel times and presented in Table 9.

Links Travel Time	(27,20)	(20,10)	(27,19)	(35,24)	(16,6)	(18,17)	(19,8)	(8,10)	(8,2)	(52,53)	(91,52)	(97,95)
CPTS	2.07	3.31	2.95	0.88	1.66	2.59	2.01	1.69	1.32	2.2	1.46	1.33
10-Step Assignment	1.6	2.2	2.2	1.4	1.1	2.1	1.20	1.2	1.1	2.07	2.1	1.2
Difference	0.47	1.11	0.75	-0.52	0.56	0.49	0.81	0.49	0.22	0.13	-0.64	0.13

Table 9. Comparison Of Travel Time On Some Uncongested Links
Obtained From The 10-Step Assignment And CPTS.

1. Link (35,24) is a four-lane bridge from Phenix City to Columbus. The travel speed on this link reported by CPTS is 55 mph, which must assume that the travel is uninterrupted on the bridge. In our assignment algorithm, the maximum travel speed is the assigned speed limit, i.e. 35 mph. Because of this, the travel time in our assignment is higher.
2. Link (91,52) is a four-lane road with flow restrictions. The flow rate is only 94 vehicles per hour per lane in our assignment, thus the travel speed is the speed limit (35 mph). However, the travel speed in CPTS was 50 mph, so the travel time in our experiment is higher.
3. Link (20,10) is a 4-lane road. Flow rate in our assignment is only 153 vehicles per hour per lane, thus the travel speed is 40 mph. Due to the higher flow rate in the study, the travel speed is about 30 mph in the report; thus, our travel time is less than theirs.
4. Links (27,19) and (19,8) are both congested in the report but not in our assignment. There are only 211 and 266 vehicles per hour per lane assigned to these links in our assignment. This is the reason for the difference in travel time.
5. In the remaining links the difference in travel time is less than 0.5 minutes from that calculated in CPTS. This similarity may be due to the fact that we count only one intersection for the travel time on any link, while the travel time calculated from CPTS counts delay in both intersections of the link.

Stepwise Assignment With Diminishing Stepsizes

An 11-step assignment was also performed with diminishing stepsizes. One half of the demand is assigned to the network in the first step, with one twentieth of the demand assigned to the remaining ten steps. The final flow distribution is presented in Figure 39. The number of links in each range of flow for 10-step equal stepsize assignments and this 11-step assignment are given in Table 10.

Table 10. Number Of Links In Each Range Of Flow For 10-Step Equal Stepsize Assignments And 11-Step Diminishing Stepsize Assignments

Range of Flow	10-Step Assignment	11-Step Assignment
$800 < x$	0	0
$700 < x < 800$	0	0
$600 < x < 700$	18	11
$500 < x < 600$	29	42
$400 < x < 500$	25	21
$x < 400$	268	266

The results show that less links are assigned over or below their capacity, while more links are assigned near their capacity. Therefore, stepwise assignment with diminishing stepsizes seems to accelerate the convergence and force the links to be restricted by their capacity.

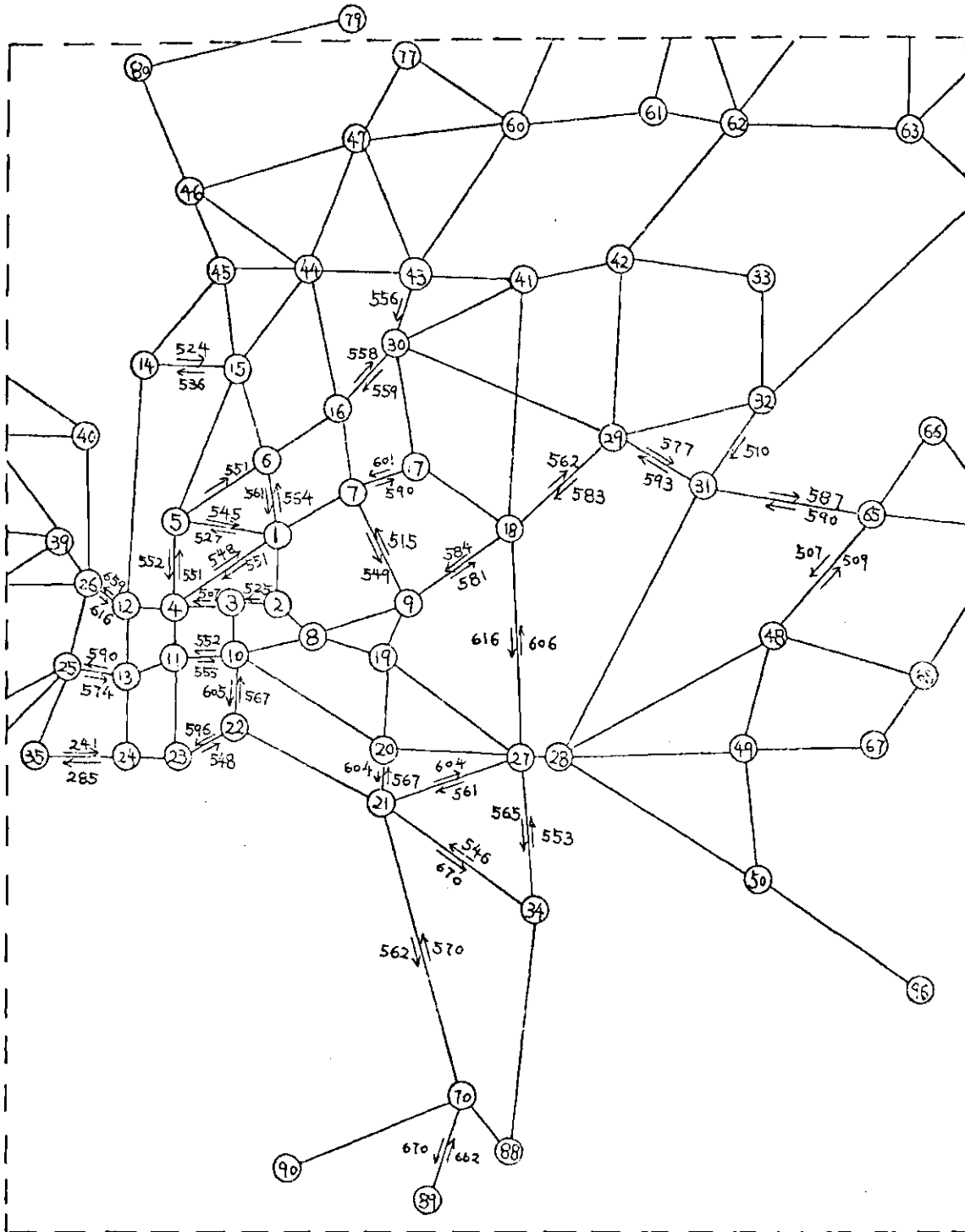


Figure 39. Flow Distribution Of 11-Step Assignment.

CHAPTER VII

CONCLUSIONS AND RECOMMENDATIONS
FOR FUTURE STUDYConclusions

In conclusion, the following features are observed:

1. To a great extent the aggregated network can reflect realistic phenomena as well as any large complex network.
2. The principle of assignment in light of the individual traveler's viewpoint is reassessed.
3. Short computation time, economical memory use, and efficient data input are features in the stepwise assignment algorithm devised in this study.
4. Flow distribution successfully reflects the relative congestion conditions in the network.
5. Travel time calculated from flow on links and nodes are in the reasonable range of realistic travel time.
6. The shortest path algorithm devised in this study is likely to be more efficient than other matrix methods, shortest path algorithms. This fact arises from the method of node numbering, combined with the alternating searching sequence, which requires fewer iterations; and from the technique of considering as intermediate nodes only the nodes adjacent to the destination node.

Recommendations For Future Study

Improvement On The Network

Efforts should be made to develop rules for aggregating the real road network into a simplified network which can still reflect significant traffic phenomena.

Improvement On The Demand

The static flow rates on each link determine the equilibrium flow distribution. Reasonable percentages of the total demand should be determined to reflect the average flow rate for the period the flow distribution is to be investigated.

Improvement On The Speed-Flow Relationships In The Link

Speed-flow relationships should be made more realistic for each category of road by considering more factors which are important, such as the number of intersections within the link or the characteristics of each category of road.

Improvement On The Delay-Flow Relationship In The Node

Instead of assuming 30 second red and green periods, the red and green period from each direction coming into the node should represent its actual value for each intersection. Thus, several delay functions under different red periods are necessary for each node. Flows coming from a specific link to the node will be subjected to a specific delay function depending on the red period in the link leading to the node.

Improvement On The Steps Of Assignment

Diminishing steps of assignment are recommended to test the optimal number of steps necessary to get reasonable results.

Improvement On The Shortest Path Finding TechniqueFor The Stepwise Assignment Algorithm

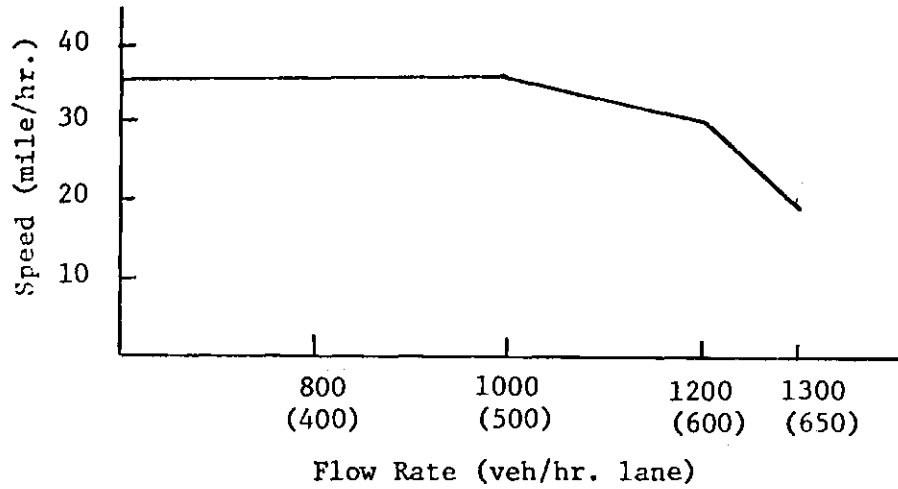
Using a tree building algorithm to find the shortest path for the node pairs would have resulted in different features for implementing the stepwise assignment algorithm. Therefore, it would be worthwhile to compare the advantages of using the tree building algorithms with those of the new matrix algorithm in supporting the stepwise assignment.

APPENDIX I

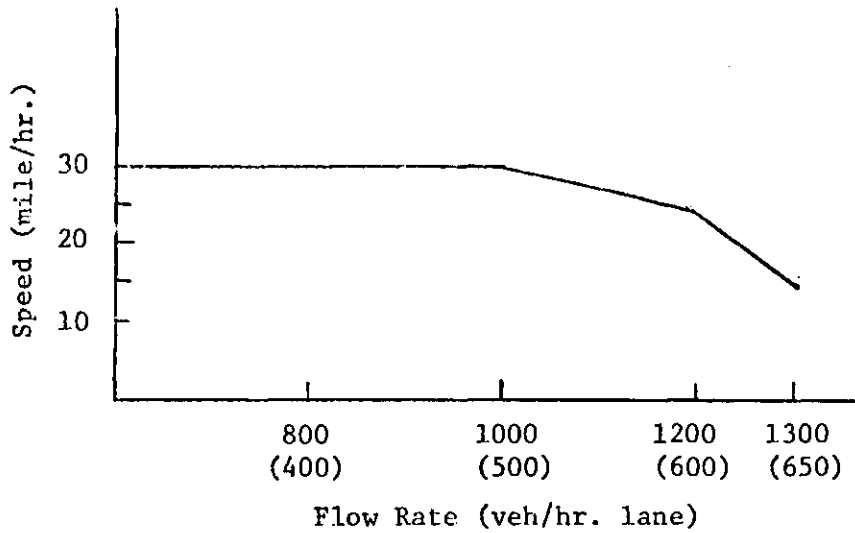
SPEED-FLOW RELATIONSHIPS

FOR MULTILANE ROADS

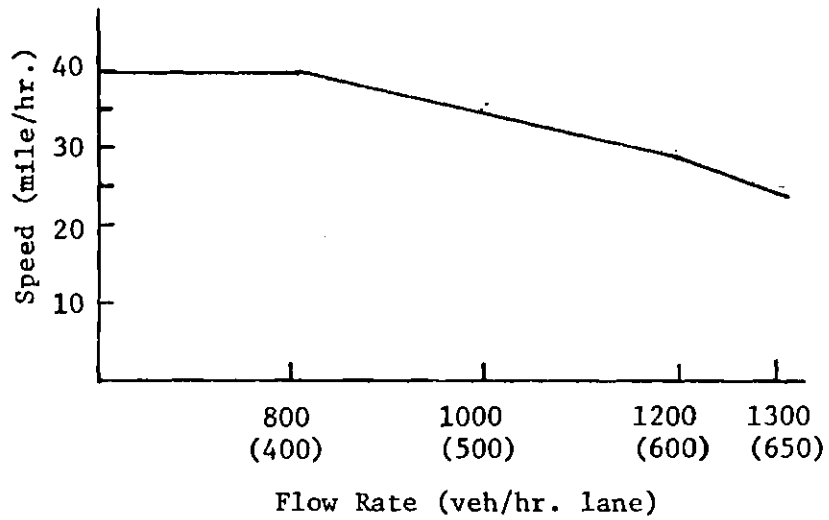
Function F2. Speed-Flow Relationship For
2-Lane Road.



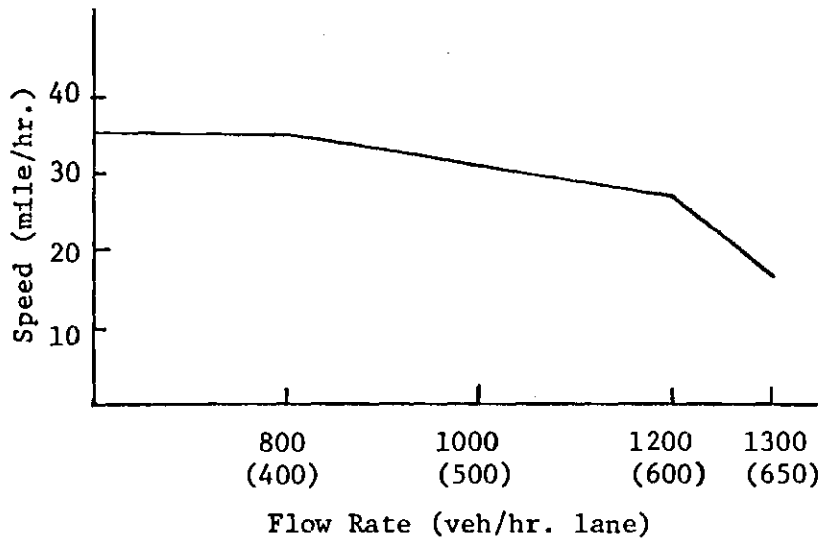
Function F2B. Speed-Flow Relationship For
2-Lane Road With Flow Restrictions.



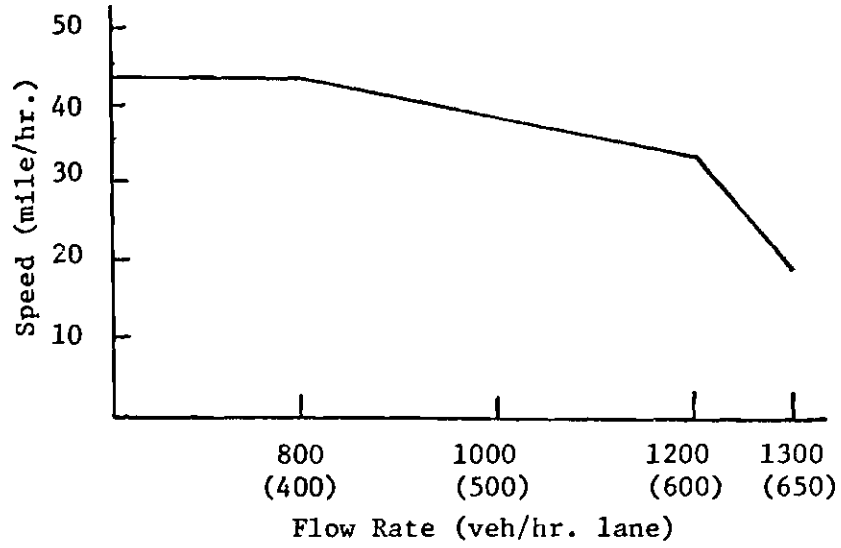
Function F4. Speed-Flow Relationship
For 4-Lane Road.



Function 4B. Speed-Flow Relationship
For 4-Lane Road With Flow Restrictions.



Function F6. Speed-Flow Relationship
For 6-Lane Road.



APPENDIX II

FORTRAN LISTING OF

STEPWISE ASSIGNMENT ALGORITHM

```

PROGRAM MAIN(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
DIMENSION AA(97,57),M(97,97),AE(578),MC(342),AG(578)
DIMENSION M4(97),M5(4128),M6(4128),M8(342),A9(11)
COMMON/S1/M1(97),M2(342),A3(342),N
DATA A9/1.,1.,2.,2.,3.,2.,2.,4.,5.,5.,3./
N=97
READ(5,30)(M1(J),J=1,N)
30  FORMAT(1X,78I1,/,1X,1811)
READ(5,55)(M2(I),I=1,342)
55  FORMAT(13(2X,29I2,/,)2X,30I2)
READ(5,41)(MC(I),I=1,342)
41  FORMAT(113(3X,25I2,/,),3),17I3)
READ(5,26)(M4(I),I=1,97)
26  FORMAT(12(4I2,/,),17I2)
READ(5,27)(M5(I),I=1,4128)
27  FORMAT(1143(4I2,/,),8I2)
READ(5,28)(M6(I),I=1,4128)
28  FORMAT(158(128I3,/,),20I3)
READ(5,40)(M8(I),I=1,342)
40  FORMAT(13(4I2,/,),22I2)
DO 33 J=1,578
33  AF(J)=1.
      ISTEP=1
C  UPDATE THE COEFFICIENT ON EACH LINK ACCORDING TO THE INCREASED FLUX
19  DO 31 I=1,97
      DO 32 J=1,57
31  AA(I,J)=9999.
      II=1
      DO 32 J=1,57
      AA(J,J)=0.
      K=C
34  K=K+1
      II=II+1
      J1=M2(II)
      L=6*(J-1)+K
      AF1=AF(L)/AS(J1)
      AFF=AF1*2.
      GC IC (1,2,3,4,5,6,7,8,9,10,11),J1
1  AS=F2(AFF)
   GO TO 35
2  AS=F2E(AFF)
   GO TO 35
3  AS=F4(AFF)
   GO TO 35
4  AS=F4E(AFF)
   GO TO 35
5  AS=F6(AFF)
   GO TO 35
6  AS=F2(AFF)
   GO TO 35
7  AS=F2E(AFF)
   GO TO 35
8  AS=F4(AFF)
   GO TO 35
9  AS=F6(AFF)
   GO TO 35
10 AS=F6(AFF)

```

```

GO TO 35
11 AS=F4(AFF)
35 AA(M2(II),J)=MC(II)/AS*1.136364+F10(AF1)
A3(II)=AA(M2(II),J)
IF(K.LI,M1(J)) GO TO 34
32 CONTINUE
C
CALL SFOPAT(AA,MN)
C
IF(ISTEP.EQ.11) GO TO 15
C ASSIGN FLOW TO EACH LINK ON THE SHORTEST PATH OF EVERY DEMAND PAIR
II=0
DO 25 I=1,N
K4=0
22 K4=K4+1
IF(K4.GI.M4(I)) GO TO 25
II=II+1
K1=M5(II)
23 J1=K1
K1=ABC(MN(II,J1),777B)
MN1=SHIFT(AND(MN(I,J1),7000B),-9)
K2=6*(J1-1)+MN1
AF(K2)=AF(K2)+M6(II)/10.
IF(K2.EQ.I122,23)
25 CONTINUE
C
ISTEP=ISTEP+1
GO TO 19
18 CONTINUE
WRITE(E,101)II
101 FORMAT(I10)
WRITE(E,29)(AF(I),I=1,578)
29 FORMAT(4X,6F6.0,6X,6F6.0,6X,6F6.0,/)
II=0
DO 42 I=1,97
K=0
44 K=K+1
II=II+1
J=6*(I-1)+K
J1=M8(II)
AG(J)=AF(J)/A9(J1)
IF(K.LI,M1(I)) GO TO 44
42 CONTINUE
WRITE(E,29)(AG(I),I=1,578)
END

```

```

SUBROUTINE SPOFAT(AA,MN)
COMMON/S1/M1(97),M2(142),A3(242),N
DIMENSION AA(97,97),MN(97,97),M7(E)
KK=KK+1
N1=N-1
DO 1 J=1,97
DO 1 J=1,97
1 MN(I,J)=I
C
ICOUNT=0
5 ICOUNT=ICOUNT+1
IO=1
C
II=341
DO 18 J1=1,N1
J=N-J1
K1=0
19 K1=K1+1
II=II-1
K=M2(III)
I=J
20 IF(II.EC.N)GO TO 21
I=I+1
IF(AA(I,J).LT.AA(I,K)+A3(II)+0.00091) GO TO 20
AA(I,J)=AA(I,K)+A3(II)
MN(I,J)=K
IO=0
GO TO 20
21 IF(K1.LT.M1(J)) GO TO 19
18 CONTINUE
C
II=343
DO 48 J1=1,N1
J=N-J1+1
K1=0
49 K1=K1+1
II=II-1
K=M2(III)
I=0
50 IF(II.EC.J-1)GO TO 51
I=I+1
IF(AA(I,J).LT.AA(I,K)+A3(II)+0.00001) GO TO 50
AA(I,J)=AA(I,K)+A3(II)
MN(I,J)=K
IO=0
GO TO 50
51 IF(K1.LT.M1(J)) GO TO 49
48 CONTINUE
C
IF(IC.EQ.1) GO TO 53
IO=1
ICOUNT=ICCOUNT+1
C
II=0
DO 38 J=1,N1
K1=0
39 K1=K1+1

```

```

      II=II+1
      K=M2(II)
      I=J
40  IF(I.EQ.N)GO TO 41
      I=I+1
      IF(AA(I,J).LT.AA(I,K)+A3(II)+C.00001) GO TO 40
      AA(I,J)=AA(I,K)+A3(II)
      MN(I,J)=K
      IO=0
      GO TO 40
41  IF(K1.LT.M1(J)) GO TO 39
38  CONTINUE
C
      II=5
      DO 8 J=2,N
      K1=0
79  K1=K1+1
      II=II+1
      K=M2(II)
      I=J
80  IF(I.EQ.J-1)GO TO 81
      I=I+1
      IF(AA(I,J).LT.AA(I,K)+A3(II)+C.00001) GO TO 80
      AA(I,J)=AA(I,K)+A3(II)
      MN(I,J)=K
      IO=0
      GO TO 80
81  IF(K1.LT.M1(J)) GO TO 79
8  CONTINUE
C
      IF(IC.NE.1) GO TO 5
53  CONTINUE
      WRITE(6,82)ICOUNT
82  FORMAT(4X,"NO. OF ITERATION = ",I4,/,)
      IF(KK.NE.11) GO TO 84
      WRITE(6,10) (MN(I,J),J=1,N),I=1,N)
10  FORMAT(4X,"MN(I,J)=",/,4X,40I3/,4X,17I3,/)
      WRITE(6,83) (AA(I,J),J=1,N),I=1,N)
83  FORMAT(4X,25F5.1/4X,25F5.1/4X,25F5.1/4X,22F5.1,/)
84  CONTINUE
C
      II=0
      DO 17 J=1,N
      K1=0
11  K1=K1+1
      II=II+1
      M7(K1)=M2(II)
      IF(K1.LT.M1(J)) GO TO 11
      DO 17 I=1,97
      IF(I.EQ.J) GO TO 17
      IF(MN(I,J).NE.M7(1)) GO TO 12
      MN(I,J)=CR(MN(I,J),15008)
      GO TO 17
12  IF(MN(I,J).NE.M7(2)) GO TO 13
      MN(I,J)=CR(MN(I,J),20008)
      GO TO 17
13  IF(MN(I,J).NE.M7(3)) GO TO 14

```

```

MN(I,J)=OR(MN(I,J),3000B)
GO TO 17
14 IF(MN(I,J).NE.M7(4)) GO TO 15
MN(I,J)=OR(MN(I,J),4000B)
GO TO 17
15 IF(MN(I,J).NE.M7(5)) GO TO 16
MN(I,J)=OR(MN(I,J),5000B)
GO TO 17
16 MN(I,J)=OR(MN(I,J),6000B)
17 CONTINUE
RETURN
END

```

```

FUNCTION F2(AFF)
IF (AFF.GT.1000.) GO TO 1
F2=35.
RETURN
1 IF(AFF.GT.1200.) GO TO 2
F2=35.-(AFF-1000.)*0.025
RETURN
2 IF(AFF.GT.1300.) GO TO 3
F2=30.-(AFF-1200.)*0.1
RETURN
3 F2=1.
RETURN
END

```

```

FUNCTION F10(AF1)
IF(AF1.GT.200.) GO TO 1
F10=0.17
RETURN
1 IF(AF1.GT.400.) GO TO 2
F10=0.17+(AF1-200.)*0.00042
RETURN
2 IF(AF1.GT.500.) GO TO 3
F10=0.25+(AF1-400.)*0.0025
RETURN
3 IF(AF1.GT.550.) GO TO 4
F10=0.5+(AF1-500.)*0.0133
RETURN
4 F10=100.
RETURN
END

```

```

FUNCTION F2E(AFF)
IF (AFF.GT.1000.) GO TO 1
F2B=30.
RETURN
1 IF (AFF.GT.1200.) GO TO 2
F2B=30.-(AFF-1000.)*0.025
RETURN
2 IF (AFF.GT.1300.) GO TO 3
F2B=25.-(AFF-1200.)*0.1
RETURN
3 F2B=1.
RETURN
END

```

```

FUNCTION F4(AFF)
IF (AFF.GT.800.) GO TO 1
F4=40.
RETURN
1 IF (AFF.GT.1200.) GO TO 2
F4=40.-(AFF-800.)*0.025
RETURN
2 IF (AFF.GT.1300.) GO TO 3
F4=30.-(AFF-1200.)*0.1
RETURN
3 F4=1.
RETURN
END

```

```

FUNCTION F4E(AFF)
IF (AFF.GT.800.) GO TO 1
F4B=35.
RETURN
1 IF (AFF.GT.1200.) GO TO 2
F4B=35.-(AFF-800.)*0.02
RETURN
2 IF (AFF.GT.1300.) GO TO 3
F4B=27.-(AFF-1200.)*0.1
RETURN
3 F4E=1.
RETURN
END

```

```

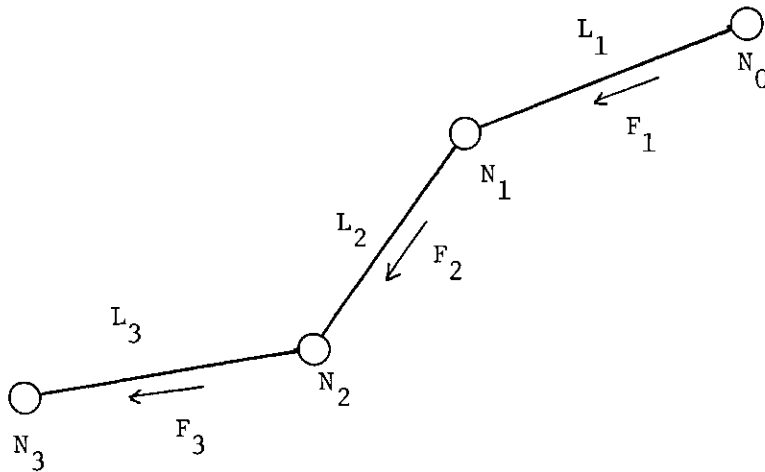
FUNCTION F6(AFF)
IF (AFF.GT.800.) GO TO 1
F6=43.
RETURN
1 IF (AFF.GT.1200.) GO TO 2
F6=43.-(AFF-800.)*0.025
RETURN
2 IF (AFF.GT.1300.) GO TO 3
F6=33.-(AFF-1200.)*0.13
RETURN
3 F6=1.
RETURN
END

```

APPENDIX III

THE DERIVATION OF THE TOTAL COST OF
TRAVELING ON A PATH

An example will be presented summarizing the derivation of the total cost of traveling on a path.



nodes: N_0, N_1, N_2, N_3

links: L_1, L_2, L_3

F_i : the flow on link L_i , $i = 1, 2, 3$

S_i : the speed-flow relationship for link L_i . $i = 1, 2, 3$

f : the delay relationship for all nodes, i.e. time delay versus the rate of flow coming into the node.

D_i : the distance of link L_i . $i = 1-2-3$.

The total cost for traveling from N_0 to N_3 through Link $L_1, L_2,$

L_3 is:

$$\frac{D_1}{S_1(F_1)} + f(F_1) + \frac{D_2}{S_2(F_2)} + f(F_2) + \frac{D_3}{S_3(F_3)} + f(F_3)$$

BIBLIOGRAPHY

1. Charnes, A. and Cooper, W. W., "Multicopy Traffic Network Models," Proceedings of the Symposium on Traffic Flow (R. Herman, ed.), 1961.
2. Chicago Area Transportation Study: Final Report, Vol. 2, pp. 104-110, July, 1960.
3. Columbus-Phenix City Transportation Study, Technical Report No. 6, "Travel Time," February, 1969.
4. Dantzig, G. B., "All Shortest Routes in a Graph," Technical Report 66-3, Operations Research House, Stanford University, 1966.
5. Dial, R. B., "A Probabilistic Multipath Traffic Assignment Model Which Obviates Path Enumeration," Transportation Research, Vol. 5, 1970, pp. 83-111.
6. Edie, L. C., and Foote, R. S., "Analysis of Single Lane Traffic Flow," Traffic Engineering, January, 1963.
7. Floyd, R. W., "Algorithm 97, Shortest Path," Communications of the Association of Computing Machinery, 5, 345, 1962.
8. Highway Research Board, Highway Capacity Manual, 1965, Special Report 87, Highway Research Board, 1966.
9. Irwin, N. A., Dodd, N., and von Cube, H. G., "Capacity Restraint in Assignment Programs," Highway Research Board Bull. 297, 1961, pp. 109-127.
10. Irwin, N. A., and von Cube, H. G., "Capacity Restraint in Multi-Travel Mode Assignment Programs," Highway Research Board Bull. 347, 1962, pp. 258-289.
11. Martin, B. V., and Manheim, M. L., "A Research Program for Comparison of Traffic Assignment Technique," Dept. of Civil Engineering, Massachusetts Institute of Technology, January, 1964.
12. Steel, M. A., "Capacity Restraint - A New Technique," Traffic Engineering and Control, October, 1965.

13. Tomlin, J. A. "A Mathematical Programming Model for the Combined Distribution and Assignment of Traffic," Transportation Science, Vol. 5, No. 2, May, 1971.
14. Webster, F. V., "Traffic Signal Settings," Road Research Laboratory Technical Paper No. 39, HMSO, London, England, 1958.
15. Yang, T. C. and Snell, R. R., "Traffic Assignment by the Maximization Principle," Journal of the Highway Division, ASCE, Vol. 92, No. HW2, 1966.