NBSIR 78-1426

Comparison of the Performance of Three Algorithms for Use in an Automated Transit Information System (ATIS)

Judith F. Gilsinn
Elizabeth L. Leyendecker
Douglas R. Shier

Institute for Basic Standards
Applied Mathematics Division
National Bureau of Standards
Washington, D.C. 20234

March 1978

Final Report

Technical Report To:

Office of Socio-Economic and Special Projects
Urban Mass Transportation Administration
Department of Transportation
Washington, D.C. 20590

NBSIR 78-1426

COMPARISON OF THE PERFORMANCE OF THREE ALGORITHMS FOR USE IN AN AUTOMATED TRANSIT INFORMATION SYSTEM (ATIS)

Judith F. Gilsinn
Elizabeth L. Leyendecker
Douglas R. Shier

Institute for Basic Standards
Applied Mathematics Division
National Bureau of Standards
Washington, D.C. 20234

March 1978

Final Report

Technical Report To:
Office of Socio-Economic and Special Projects
Urban Mass Transportation Administration
Department of Transportation
Washington, D.C. 20590

ABSTRACT

This paper compares the performance of three algorithms for computing trip itineraries for use in an automated transit information system. One of the approaches (TIMEXD) is based on a time-expanded network. The other two both compute paths in a bipartite route/stop network; one algorithm (LABCOR) is based on the label-correcting approach and the other (LABSET) on the label-setting approach. The transit networks upon which the performance comparison is based are of two types: a grid network with specified, possibly non-uniform, distances between streets, and a spider web type of network. TIMEXD is fastest on all the larger networks, but it requires most computer storage and outputs paths with more transfers. LABCOR is the slowest, but is guaranteed to produce the best routing, since it always outputs an optimal path with fewest transfers. Computation time estimates extrapolated to large transit networks indicate times of 1.5 to 2.5 seconds per itinerary for TIMEXD and LABSET respectively, well within the acceptable range for such networks.

Key Words: Algorithms; algorithm testing; mass transit; routing; shortest paths; transit; transit information systems; transit routing; transportation; urban transportation.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Most larger transit systems operate a telephone transit information center to provide prospective riders with itineraries for potential trips. Usually transit system staff provide these itineraries by using maps, schedules and routing information to piece together manually a trip to meet the caller's request. The largest systems may have as many as 80 people involved in answering telephoned requests for transit information. Consideration is currently being given to automating the route-finding portion of the transit information activity. Transit system staff would still be required to interpret the caller's request and relay the computer-produced itinerary to the patron. It is expected that automation of the route-finding portion of a call would significantly reduce the longer calls and could result in an overall average reduction of about 20 percent in call length. Other benefits resulting from automation include repeatability of response and lessened requirements for training of telephone-answering staff in city geography and transit system routes. Analysis of the costs and benefits of an automated transit information system is given in [3].

At the heart of an automated system is a procedure (mathematical algorithm) for finding trip itineraries in a transit network. Such an algorithm would have available to it computerized descriptions of transit stops, routes and schedules, as well as trip origin and destination and either a desired departure or a desired arrival time. Using these data the algorithm would produce an optimal trip itinerary, "optimal" in the sense of describing either a trip which arrived soonest, having departed at or after the desired departure time, or a trip which departed latest to arrive at or before the desired arrival time. In addition to the route-finding algorithm, the computer would also have a procedure for identifying geographically the caller's trip origin and destination and appropriate transit stops accessible to these points. The itinerary produced by the route-finding algorithm would specify boarding time and stop, transfer stops and the arrival and boarding times at such stops whenever transferring is required, arrival stop and time, and routes for each segment. An example of such an itinerary follows:

Board Route RED at 5TH AND ELM at 9:00 A.M.,
Arrive at 5TH AND OAK at 9:15 A.M.

Board Route BLUE at 5TH AND OAK at 9:17 A.M.,
Arrive at 10TH AND OAK at 9:25 A.M.

Further discussion of the data structures, programs and procedures involved in an automated transit information system is presented in [1]. Also included in an appendix to that report are the descriptions of three specific route-finding algorithms which are the subject of the analysis presented below. The main objective of [1] was to assess the feasibility of designing algorithms which could compute itineraries fast enough to improve the information system's response to the caller. With feasibility established, the current analysis turns to the choice of algorithm as it relates to the characteristics of a transit system, such as the size and complexity of network structure, the regularity of departures, transfer times, relative speeds of express and local service, and various patterns of preferential service.

Section 2 below contains descriptions, excerpted from [1], of the three algorithms which are compared in the present analysis. Section 3 contains descriptions of the two network-generation programs used to produce transit-like networks on which the algorithms were tested. The results are discussed in Section 4, and recommendations for choice are provided in Section 5. Program documentation and listings appear in appendices.

The results of the analyses described in Section 4 may be summarized as follows. The label-correcting bipartite route/stop algorithm (LABCOR) provides the most desirable trip output of the three algorithms, since it always produces, from among those trips arriving at the same time (under the departure-oriented criterion), that trip requiring fewest transfers. However, LABCOR is significantly slower than the other two algorithms. The time-expanded network algorithm (TIMEXD) is fastest on the test networks, and its speed increases with network size at the lowest rate of the three algorithms. However, TIMEXD requires significantly more storage than the other two algorithms, mainly because of the need to store explicitly each possible transfer. The third algorithm, the label-setting bipartite route/stop approach (LABSET), is slower than TIMEXD but faster than LABCOR. Path output from LABSET has more transfers than that from LABCOR, but not as many as from TIMEXD. Although LABSET requires somewhat more storage than LABCOR, both require significantly less than TIMEXD. Computation times for all three algorithms depend mainly on the number of stops in the transit network. Other factors which have some effect include the number of transfers required, the average transfer time and its variability, and the relative speeds of express and local service.

2. ALGORITHM DESCRIPTIONS

Descriptions of the three algorithms were given in an appendix to [1] but are repeated here to make this report more self-contained. The algorithms rely for their efficiency on specialized representations of the transit data base of stops, routes and schedules. The three algorithms which are analyzed in Section 4 all use the "departure-oriented" optimality criterion for finding a best path, and thus produce a trip which arrives soonest while departing at or after the desired departure time. We include below (and in the appendix) the description of one of the algorithms, the time-expanded algorithm, programmed for the arrival-oriented case. We expect no difference in the performance between departure- and arrival-oriented algorithms and have therefore focussed in the analysis on the departure-oriented criterion.

2.1 Bipartite Route/Stop Algorithms

The nodes of the bipartite route/stop network are of two types, one representing the geographical transit stops and the second representing individual transit routes. Network arcs are also of two types: for each transit stop an arc connects it to those lines stopping there, and for each route an arc connects it to the stops along that route. (The arcs associated with a route appear in the order of the stops along the route.) Thus the network described here is bipartite in the usual graph-theoretic sense that the nodes of the network may be partitioned into two sets in such a manner that arcs connect a node in one set with a node in the other set but do not connect nodes in the same set. Figure 1 displays an example of such a network. Note that a dummy route was introduced for a walk transfer link connecting two other routes. A path in this network is an alternating list of stops and routes, beginning with the origin stop and ending with the destination stop. The route node appearing between each pair of stops specifies the route which should be taken between them. The number of transfers is thus one less than the number of routes appearing in the list, or alternatively, since each stop other than the origin and destination represents a transfer, two less than the number of stops in the path.

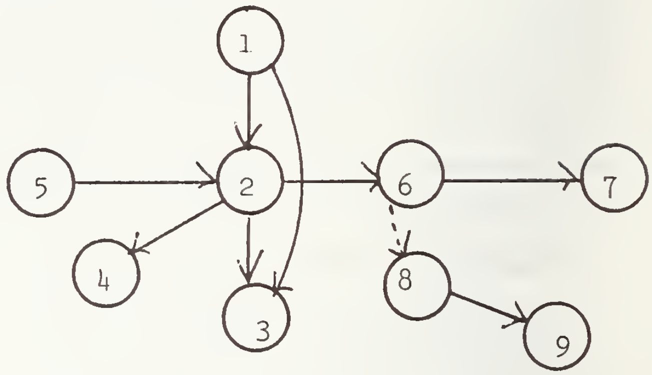
The network as described above does not have associated with it the time data specifying each discrete departure. The arcs connecting the routes to the stops actually represent a whole list of scheduled vehicle trips along the route, to be fetched during the course of the algorithm as needed. An example of such a list for one route is given in Figure 2. Each column gives times at a stop and each row represents one transit vehicle's trip along the route. Thus if the arcs emanating from a route node are listed in the order of the stops along the route, a row of Figure 2 represents arrival times at the arc endpoints in order. We will describe two computational schemes based on this network, one using the basic label-correcting procedure and a sequence list ordered by cardinality distance (in this case the number of vehicles used) and the second using the label-setting procedure and a list ordered by temporary label (in this case trip arrival time). A more detailed description of

FIGURE 1

Illustrative Bipartite Route/Stop Network

ORIGINAL NETWORK

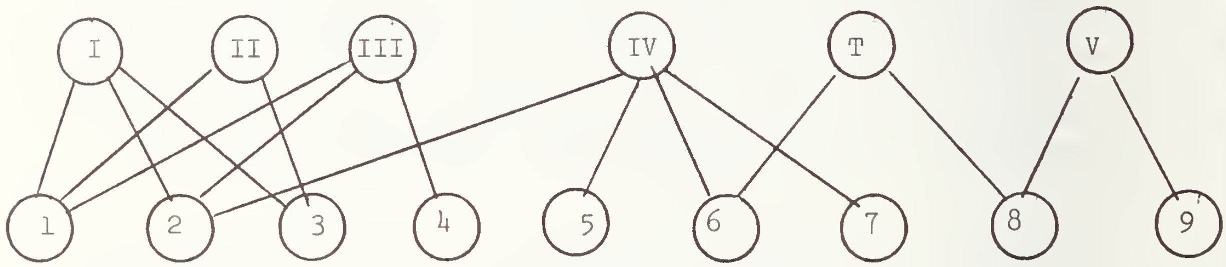
Route	Stops
I	1, 2, 3
II	1, 3
III	1, 2, 4
IV	5, 2, 6, 7
V	8, 9
T	6, 8



9 Nodes

9 Arcs

BIPARTE ROUTE/STOP NETWORK



15 Nodes

32 Arcs (Each Connection is Two-way)

FIGURE 2

Illustrative Route Schedule

Stop 1	Stop 2	Stop 3	Stop 4
7:00	7:10	7:15	7:30
7:30	7:45	7:55	8:10
8:00	8:15	8:25	8:40
8:30	8:40	8:45	8:55
10:00	10:05	10:09	10:15
12:00	12:07	12:15	12:25
2:00	2:05	2:09	2:15
4:00	4:07	4:15	4:25
4:30	4:40	4:50	5:05
5:00	5:15	5:25	5:40
5:30	5:45	5:52	6:05
6:00	6:10	6:15	6:25
6:30	6:40	6:45	6:55
8:00	8:05	8:09	8:15
10:00	10:05	10:08	10:13

the basic label-correcting and label-setting procedures can be found in [2]. The following notation will be used in describing the schemes.

R	the set of all transit routes
r	a particular route
S	the set of all transit stops
s	a particular transit stop
ORG	the origin transit stop
DST	the destination transit stop
N	$R \cup S$, the nodes of the whole bipartite network
T(i)	arrival time at stop i via best path from ORG, for $i \varepsilon S$
P(i)	node preceding i in best path from ORG. (Note that if $i \varepsilon R$ then $P(i) \varepsilon S$, while if $i \varepsilon S$ then $P(i) \varepsilon R$.)
L(k)	sequence list of nodes in S, developed by the scheme, indicating the order in which they are to be fanned out from. In the label-correcting method L is maintained in cardinality distance order; in the label-setting method, it is ordered by arrival time.
F(i)	position of node i in sequence list L
u	current position in the sequence list
v	last position filled in the sequence list
END	last entry in sequence list L

2.1.1 LABEL-CORRECTING BIPARTITE ROUTE/STOP ALGORITHM (LABCOR)

A computational scheme LABCOR for a label-correcting procedure for use with the bipartite route/stop network is given below.

<u>Initialization</u>: Set $T(i) = \infty$ for all $i \neq ORG$ and set T(ORG) = desired departure time. Set P(i)=0 and $F(i)=\infty$ for all $i \varepsilon N$. Set u=0 and v=0.

<u>Step 1</u>: Let i=ORG. Let r be the first listed route stopping at i.

<u>Step 2</u>: Search the schedule for route r for the first departure from i at or after T(i). Let s be the first stop occurring after i in route r.

<u>Step 3</u>: Compare the arrival time at s of the scheduled vehicle found in Step 2 with the current value of T(s). If it is not less, go to Step 6; if it is less, set P(s)=r and P(r)=i.

Step 4: If $T(s) = \infty$, go to Step 5. Otherwise let $k=F(s)$. If $k>0$, remove s from its previous position by setting $L(k)=0$.

Step 5: Set $T(s)=$ the time of arrival of route r at node s. Let $v=v+1$. If $v>END$, set $v=1$. Set $L(v)=s$ and $F(s)=v$.

Step 6: Let s be the next stop on route r, and go to Step 3. If there are no more stops on r, let r be the next route stopping at i and go to Step 2. If there are no more routes stopping at i, set $F(i)=0$.

Step 7: If $u=v$, stop. Otherwise let $u=u+1$. If $u>END$, let $u=1$. Let $i=L(u)$. If $i=0$, repeat Step 7. Otherwise let r be the first route stopping at i and go to Step 2.

These computations do actually maintain the sequence list L in cardinality distance order. Note that successive path segments are always by different routes, so that cardinality distance is associated with the number of different routes used in a path. ("Actual" cardinality length of a path in this network is twice the number of routes used since paths consist of an alternating stop-route sequence. However since L contains only stops, it can be used in obtaining directly the number of routes used.) If it is desired to consider only paths using no more than some maximum number of routes, say r_{max}, then the computational scheme given above can be modified easily to accommodate this additional constraint, utilizing two additional pointers:

m the cardinality distance (actually the number of routes) used in the current path from ORG to node i.

j the position in L of the last node of cardinality distance m.

Both m and j are initialized at 0. The computational scheme is modified by the addition of a Step 6.5 between Steps 6 and 7 above.

Step 6.5: If $u=j$, let $m=m+1$. If $m=r_{max}$, stop. Otherwise, set $j=v$.

2.1.2 LABEL-SETTING BIPARTITE ROUTE/STOP ALGORITHM (LABSET)

A label-setting scheme LABSET for use with the bipartite route/stop network is similar to that given above, but the sequence list L is kept ordered by arrival time at each node. The length of L is determined by M, one plus the maximum arc length. See [2] for a more complete discussion of the label-setting procedure. Since for transit networks the transferring time must be included in M, it is perhaps easiest to set M at some reasonable trip length level (say 3 hours, or if desired 24 hours).

Initialization: Set $T(i)=\infty$ for all nodes $i \neq$ ORG and T(ORG)= desired departure time. Set $P(i)=0$ for all nodes i. Let u be one plus the desired departure time (mod M).*

Step 1: Let i=ORG. Let r be the first route stopping at i.

Step 2: Search the schedule of route r for the first departure from i at or after T(i). Let s be the first stop occurring after i in route r.

Step 3: Compare the arrival time at s of the scheduled vehicle found in Step 2 with the current value of T(s). If it is equal or greater, go to Step 5; if it is less, replace the old T with the new value and set P(s)=r and P(r)=i.

Step 4: Let k=T(s) (mod M). Store s in position k+1 of L.

Step 5: Let s be the next stop on route r and go to Step 3. If there are no more stops on r, let r be the next route stopping at i, and go to Step 2. If there are no more routes stopping at i, continue to Step 6.

Step 6: Let u=u+1. If u>M, let u=1. If L(u)=DST, stop. Otherwise, let i=L(u). If i=0, repeat Step 6. Otherwise let r be the first route stopping at i and go to Step 2.

Note that termination occurs when DST is the node to be fanned out from. Thus if DST is fairly close to ORG the label-setting procedure requires much less calculation than the label-correcting procedure, which terminates only after best paths to all nodes have been found.

2.2 Time-Expanded Network Algorithms

Nodes in the time-expanded network are defined by a pair of entities, the geographical transit stop (a separate node for each route) and a time of day. Thus the geographical node Sixteenth and K Street will become several nodes, one for each time a transit vehicle stops there. The network arcs become transit trips departing one stop at a particular time and arriving at another stop at a different (later) time. Transfer arcs, representing allowable transfers (i.e., those obeying minimum transfer times), must be coded directly. An example of such a network is depicted in Figure 3. In this example there are four transit stops and three bus lines: a local stopping at each stop (its two daily runs are represented by the paths $1 \rightarrow 3 \rightarrow 6 \rightarrow 8$ and $12 \rightarrow 16 \rightarrow 17 \rightarrow 20$), a faster vehicle starting at the second stop of the first route and proceeding directly to the last stop of that route (with four runs: $5 \rightarrow 7$, $9 \rightarrow 10$, $14 \rightarrow 15$, and $18 \rightarrow 19$), and one coming from the last stop of the first route

---

* By X(mod M) is meant the remainder when X is divided by M. Its calculation is usually available in FORTRAN through the function MOD(X,M).

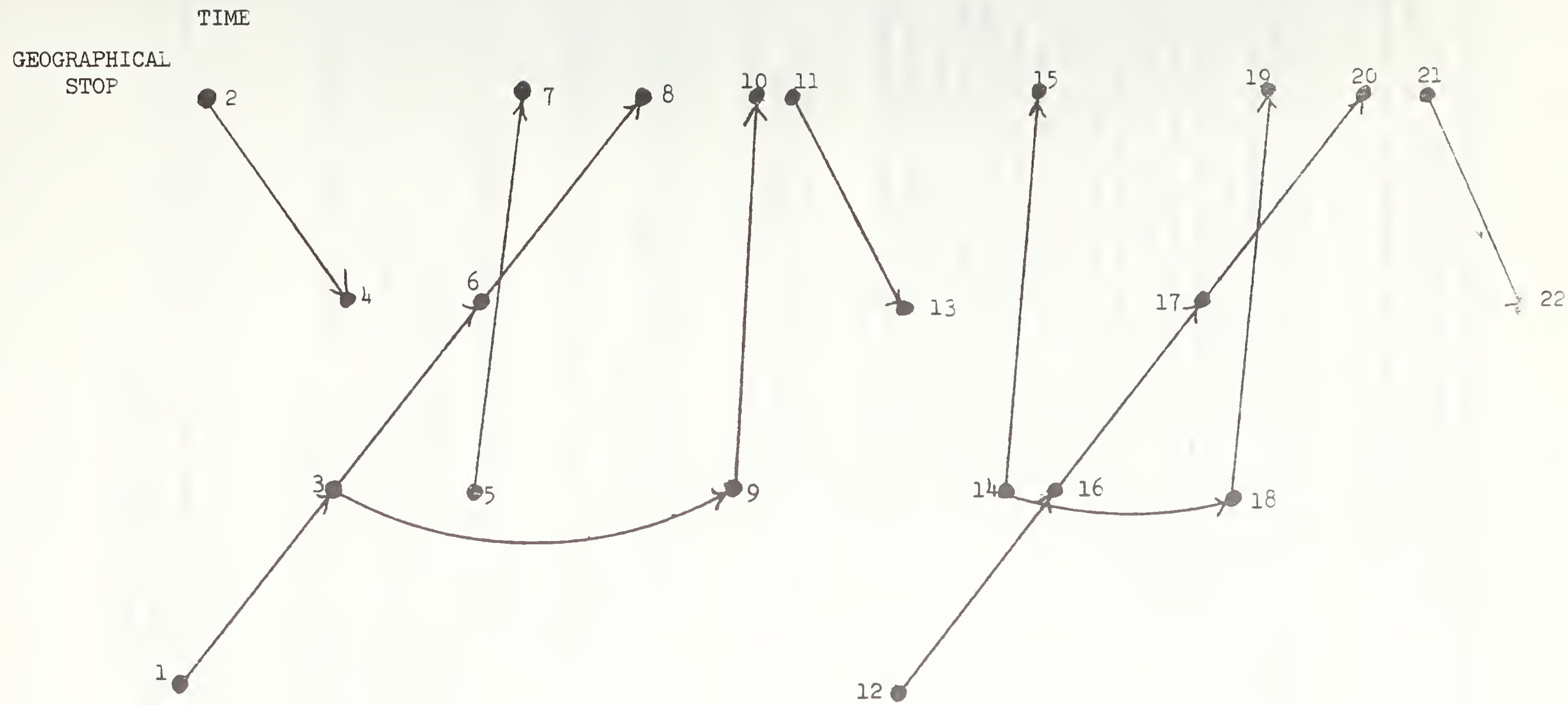


FIGURE 3

Illustrative Time-Expanded Network

back to the next-to-last stop of that route (its runs are represented by 2→4, 11→13, and 21→22). Two transfer arcs, 3→9 and 14→18, have been included. Note that in the left transfer, one is prevented by a minimum transfer time restriction from making the earliest vehicle on the second line.

The four stop network has been transformed into one with 22 nodes and 15 arcs. In general, time-expanding the network greatly increases the number of nodes, in fact by a factor equal to the average number of transit vehicle departures per geographical node. Generally it is desirable to decrease rather than increase network size, but the fact that the resulting time-expanded network is acyclic* means that the increase is likely to be beneficial. On an acyclic network the network nodes can be numbered at the outset in such a way that for each arc (i, j), i<j. That is, in the numbering, arcs always lead from lower numbered nodes to those with higher numbers. Of course a similar property also holds true of paths. This limits the search for path extensions to nodes whose numbers are greater than nodes already in the path, so that nodes can be interrogated in the order of their numbering. The node order assumed below is the one determined entirely by the time component of node identity, except for "ties" which cause no trouble unless some arc requires no time to traverse. If only major stops are included this situation is unlikely to occur, but if it were, procedures exist to number nodes having identical time components.

The node numbering procedure allows the network to be broken up into pieces (which will be called pages) so that the computational scheme only needs one of them at a time and can finish with the current one before needing the next.

The computational schemes proposed for computing transit paths in such a network appear below and rely on the basic label-correcting scheme. We will use the following notation:

n(i)	the geographical transit node associated with the network node numbered i
t(i)	the time associated with network node numbered i
ORG	geographical node of transit origin
DST	geographical node of transit destination
P(i)	the network node preceding i in a best path from ORG to the network node numbered i

*A network is <u>acyclic</u> if there are no paths containing more than one node and beginning and ending at the same node. Since all arcs in the time-expanded network go forward in time, no path can return to a node once it has left that node, so this network is acyclic.

DONE — the first node associated with DST encountered (i.e. the one for which t is minimum) in a path starting at a node associated with ORG

2.2.1 DEPARTURE-ORIENTED ALGORITHM (TIMEXD)

The computational scheme proceeds through the following steps:

Initialization: DONE = ∞; P(i)=0 for each node i.

Step 1: Scan the arc list starting with the first node i for which t(i) is not less than the desired departure time. Let i be the first node encountered with n(i)=ORG.

Step 2: Let a=(i,j) be the first arc originating at node i. If there are none, go to Step 6.

Step 3: If P(j)$\neq$0, go to Step 5. Otherwise set P(j)=i.

Step 4: If n(j)$\neq$DST, go to Step 5. Otherwise set DONE=min (DONE, j).

Step 5: Let a=(i,j) be the next arc originating at node i, if there is one, and go to Step 3. Otherwise continue.

Step 6: Let i=i+1. Stop if i=DONE.

Step 7: If P(i)=0 and n(i)$\neq$ORG, go back to Step 6. Otherwise go to Step 2.

It is clear from this description that only the nodes numbered between the first departure from ORG after the desired departure time (which we shall call i') and the node DONE are examined as arc origin nodes. In most instances this should be considerably fewer than the total number of nodes in the network. To take advantage of this fact, one may store information about node i in position i-i'+1 in the P array.

The algorithm described above requires only one pass through the nodes, and only a subset of the nodes at that. No sorting or sequencing of nodes is necessary, since nodes are examined in numerical order. Since arcs are stored sorted by origin, only that portion of the network originating at nodes i' through DONE need be referenced for this path calculation. An arc in this application only requires identification of its origin and destination nodes since the t and n pointers describe the relevant arc characteristics.

2.2.2 ARRIVAL-ORIENTED ALGORITHM (TIMEXA)

For the arrival-oriented criterion a modified version of the above scheme may be applied, using the same network and examining the nodes in reverse order starting from the last node associated with DST whose time is before the desired arrival time. This scheme will be described below. Use of two schemes has the advantage that only one copy of the network, the forward star form, need be stored to handle both the departure oriented and arrival oriented criteria. This is particularly necessary for the time-expanded network because of its large size. Different schemes are then applied to the network for the two criteria, the one above for the departure oriented criterion and the one below for the arrival oriented criterion. The following array and variable will be used in describing the scheme, together with the arrays n and t and variables ORG and DST listed above:

- S(i) the network node succeeding i in a best path from the network node numbered i to DST
- FIN the first node associated with ORG encountered (i.e., the one for which t is maximum) in a path ending at a node associated with DST.

The computational scheme proceeds through the following steps:

Initialization: FIN=0; S(i)=0 for all i.

Step 1: Scan the arc list backwards, starting with the last node i for which t(i) is at most the desired arrival time. Let i be the first node encountered with n(i)=DST. Set S(i)=i. Go to Step 6.

Step 2: Let a=(i,j) be the first arc originating at i. If there are none, go to Step 6.

Step 3: If S(j)=0, go to Step 5. Otherwise set S(i)=j.

Step 4: If n(i)≠ORG, go to Step 5. Otherwise set FIN=max(FIN,i).

Step 5: Let a=(i,j) be the next arc originating at node i, if there is one, and go to Step 3. Otherwise continue.

Step 6: Let i=i-1. Stop if i=FIN.

Step 7: If n(i)≠DST, go to Step 2. Otherwise, set S(i)=i and go back to Step 6.

In the runs described in Section 4, only the algorithms using the departure-oriented criterion, LABCOR, LABSET and TIMEXD, were tested. Runs of TIMEXA and TIMEXD suggested that the two algorithms performed similarly. In addition, the symmetry of the criteria and algorithms suggest that the two criteria should be equally efficient to process.

3. TEST-PROBLEM GENERATION

Two network generation programs have been written to produce the route and schedule information for the algorithm testing described in Section 4. One program generates a p by q grid network in which routes run either horizontally or vertically and transferring is possible at any intersection. Every trip requires at least one transfer, as long as the origin and destination are not on the same horizontal or vertical route. Subsets of the horizontal and vertical streets may be designated as main streets. Express routes begin and end at the outside of the grid, run along main streets, and stop only at intersections of other main streets. An example of such a network is shown in Figure 4, with express routes indicated by the wider lines and stops on express routes as blackened disks.

A second program generates a spider web radial type of network in which routes run inward or outward along radials from a central node and also clockwise or counterclockwise along beltways or partial beltways connecting the radials. Other routes may run from the center out along a radial, diverging from the radial at some point along it. As with the grid network, radials and beltways may be designated as major arteries along which express routes run. The express routes only stop at intersections of other express routes. Any node in the network is accessible from any other node either by traveling to the center along one radial and out along another, or else by traveling around one of the beltways. At least one transfer is required if the origin and destination are not on the same radial or on the same beltway. Figure 5 displays an example of a radial network, with express routes shown as wider lines and stops on express routes as blackened disks.

Schedule information for both types of networks is given by providing the initial run's departure time, the number of runs and the headways for each time period and route. Several routes may be grouped together if they have the same headways and numbers of runs. Distances between grid elements are provided as input to the grid type networks. Distances of nodes out from the center along the radials are input to the radial network generator, as are the angles between radials. Distances along beltway sections are then calculated as circular arc approximations. Using the arc distances, run departure times, and speed factors for either local or express routes, stop times at later nodes along each route can be calculated.

Output from the network generation programs consists, for each route, of the stops along that route and, for each departure along the route, the time it reaches each of the stops.

The networks generated by these programs clearly represent idealizations of transit system structure, but examination of several transit system maps has indicated that many systems have an underlying grid or radial structure or a combination of the two. Any undue simplification

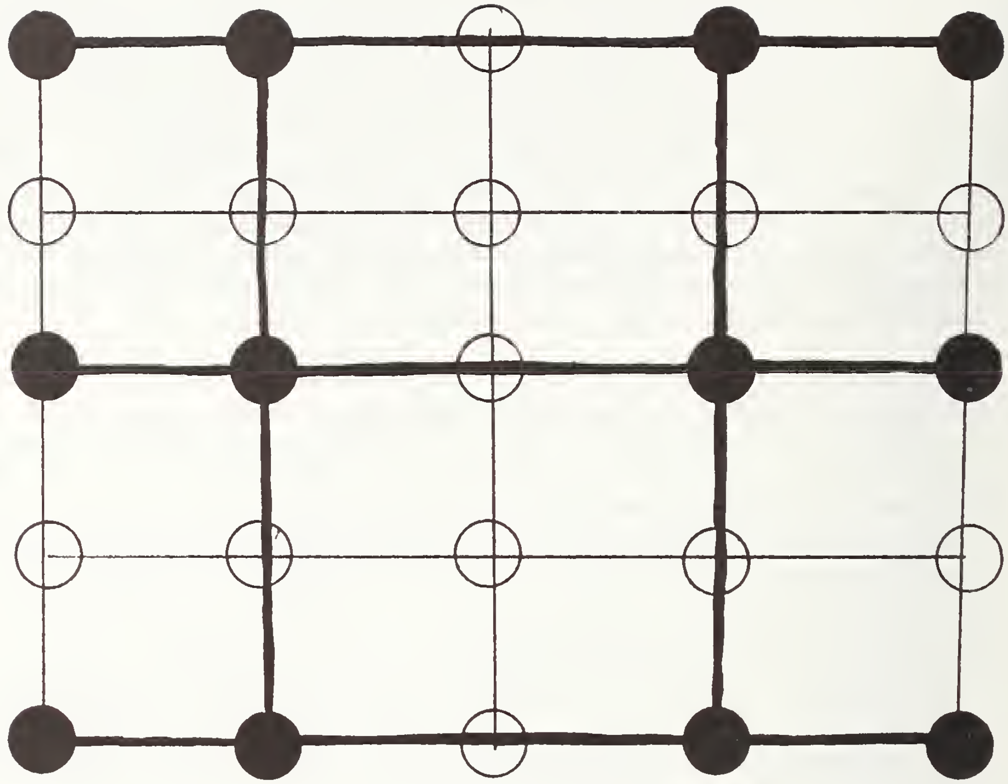


FIGURE 4

Example of a Grid Network

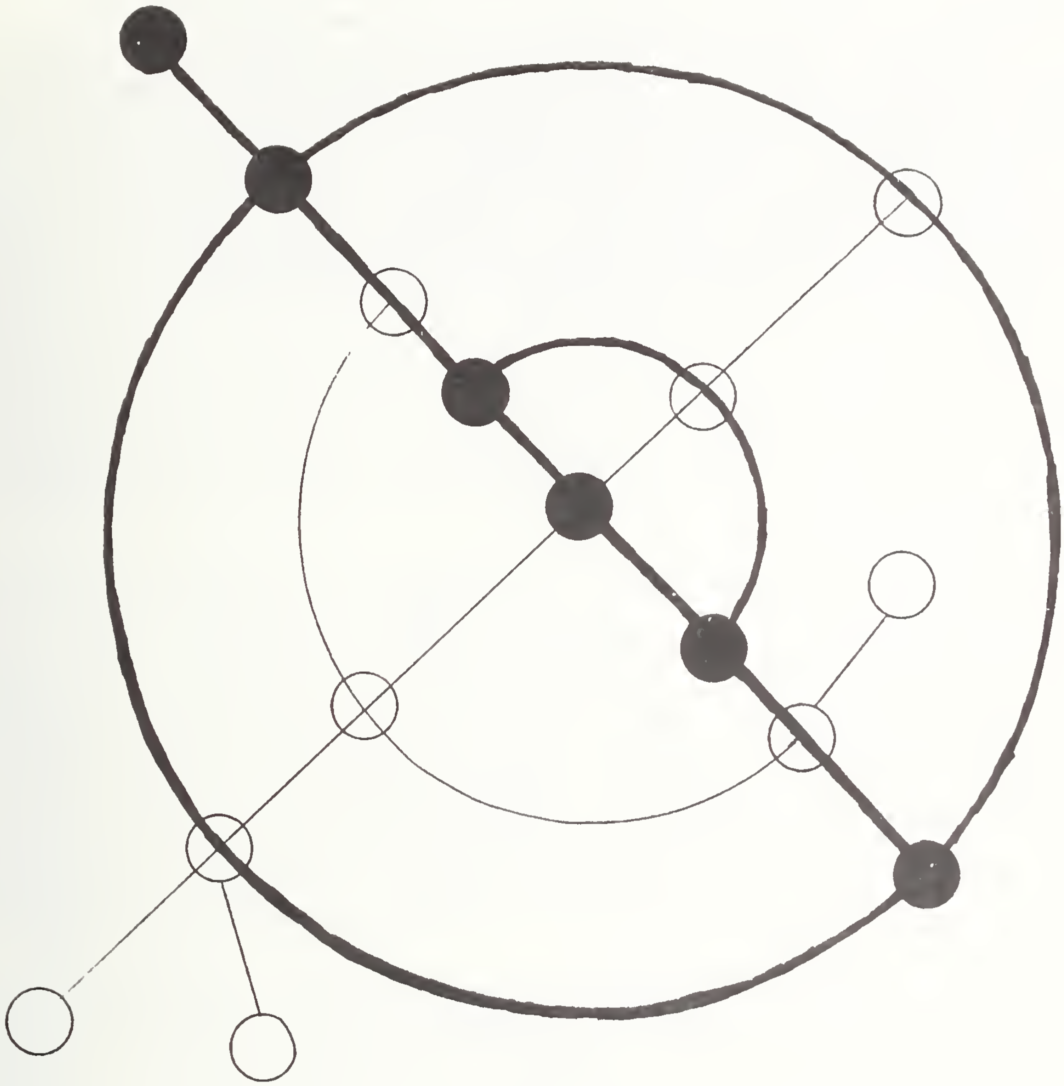


FIGURE 5

Example of a Radial Network

associated with the topological regularity implicit in the grid and radial structures is counteracted somewhat by the variability in the service availability. In fact, since almost all trips calculated for the analyses in Section 4 required at least one transfer, whereas most transit systems are designed so that many frequently-made trips require no transfer, the pure grid and radial network structures may actually be a more difficult test case than would be real networks of the same size. Post facto justification for using idealized networks in the tests is the finding, reported in Section 4, that the particular network structure is less important in determining algorithm performance than are various measures of the complexity and variability of network parameters.

4. ANALYSIS OF ALGORITHM PERFORMANCE

The network generation programs described in the previous section were used to produce transit networks with specified characteristics for use in testing the three algorithms, LABCOR, LABSET and TIMEXD. Twenty-six different situations were generated, with a particular test network characterized by the network generator used (i.e., grid or radial), the minimum transfer times required, and the network input parameters such as size (number of nodes and runs), frequency of service, and speeds of express and local vehicles. For each test, twenty-five itineraries were calculated, with their origin-destination pairs and times of day chosen to represent "reasonable" trips. The O-D pairs were not formally chosen by any random process and probably overrepresent the longer and more complicated trips, just those which are more difficult and more time-consuming to calculate. In some cases, no trip was found within the desired time period. In order to ascertain that no trip existed, the algorithms had to process some data; timings for such situations have therefore been included in the analysis.

The tests were run on the UNIVAC 1108 at the National Bureau of Standards (NBS) under the EXEC 8 operating system. The programs were all coded in FORTRAN V, UNIVAC's enhanced version of FORTRAN IV. Care was taken to insure that the programs were coded without utilizing special peculiarities of the UNIVAC 1108 and its FORTRAN compiler. The algorithms were coded to make them as comparable as possible. Timings include only the algorithm calculation portion of each of the programs; they do not include any input/output operations. All problems were core-size problems, that is, the data base for each problem was sized so that it could be accommodated in the main memory of the computer. This simplified programming and eliminated one possible source of bias or variability in performance.

The analyses described below investigate the comparative performance of the three algorithms LABCOR, LABSET, and TIMEXD on small transit networks containing from 40 to 225 nodes. One test example used a grid network containing 1600 nodes, which is similar in size to the transit system in a medium size city. The runs were made primarily to test algorithm performance on a variety of types of networks with varying characteristics. The network generation programs were designed to allow input control of several different network parameters including network size as measured by the numbers of nodes and vehicle runs, network shape as measured by the number of horizontal and vertical routes in a grid or the geometry of a radial network, the variability of schedules, and the relative speeds of express and local service. By careful selection of appropriate input parameters, scenarios representing a variety of reasonable network types can be simulated and variations on these networks can be evaluated. The tests and analyses performed to date do not exhaust all possible tests which could be informative, but they were designed to reveal the general performance of the algorithms across the spectrum of likely input situations.

4.1 Analysis of the Path Output

In comparing the itineraries output by the three algorithms for a given trip, it was necessary to choose the most desirable. When all had the same number of segments, the main difference was usually whether the traveler waited at the origin or at an intermediate stop, a choice with neither alternative always being preferable. When there was a difference in the number of transfers, however, it was believed that the trip with fewer transfers should always be considered preferable. Table 1 displays, therefore, the number of transfers required by each of the three algorithms for each of the 25 requested itineraries in each of the 26 test networks. As noted in the descriptions of the algorithms, LABCOR always produces that trip which arrives first while departing at or after the desired departure time and which also has the fewest transfers. The other two algorithms will produce trips arriving at the same time, but these trips may require more transfers. In fact, 9 percent of LABSET trips and 14 percent of TIMEXD trips required more transfers than LABCOR trips.

One type of situation in which a routing algorithm may provide extra transfers occurs when an express vehicle overtakes a local vehicle. This is illustrated in Figure 6. The soonest arrival time for a trip from node 1 to node 5 departing at or after 5:00 is 5:35. Two different itineraries with the same arrival time are possible: The first starts out at 5:00 on the local and arrives at stop 3 at 5:20, transfers to the express at 5:25 and arrives at node 5 at 5:35. The second waits 15 minutes to board the express at node 1 at 5:15 and takes the express direct to the destination at 5:35. Taking the express is preferable since it does not involve a transfer. Examples of the overtake situation occurring in the test runs are shown in Figure 7. In each case the routing produced by LABCOR (and in these cases also by LABSET) waits at an intermediate node for an express route, while TIMEXD takes the local route which is the first vehicle leaving and transfers to the express just before it overtakes the local.

In an attempt to avoid so many transfers by TIMEXD, we tried varying the choice criterion for trips which arrive at the same time-expanded node (that is, trips which arrive at the same geographical node at the same time). Table 2 shows the results of using two criteria, one which always picks the first trip encountered, which is also the trip whose last segment starts earliest, and a second which picks the trip whose final segment starts latest. Although the latter criterion might seem to correct the difficulties encountered in an overtake situation, it does not always produce trips with fewer transfers as can be seen in Table 2 . These results were obtained for network number 2, which had no express routes and may thus make the second criterion appear less desirable than is actually the case. However, neither criterion seems to reduce appreciably the incidence of extra transfers using TIMEXD.

TABLE 1
NUMBER OF TRANSFERS IN EACH PATH
Transfers for LABCOR

NETWORK NO. / CASE	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	1	2	1	0	1	1	0	1	2	1	1	1	1
2	2	1	1	1	1	1	1	1	1	2	1	1	1	1
3	1	1	1	1	0	0	0	0	1	1	1	1	1	1
4	1	1	1	1	0	0	0	0	1	2	1	1	1	1
5	1	1	1	1	2	1	1	2	1	1	1	1	1	1
6	1	1	1	1	3	1	1	1	2	2	0	1	1	1
7	1	1	-	1	0	0	0	0	0	0	1	1	1	1
8	2	1	1	1	0	0	0	0	1	2	1	2	1	0
9	1	1	1	1	1	1	1	1	0	0	1	1	1	1
10	1	1	1	1	0	0	0	0	0	0	1	1	2	1
11	1	1	1	1	2	1	1	2	0	0	1	1	1	1
12	1	1	1	1	0	0	0	0	0	0	1	1	1	1
13	2	1	-	1	0	0	0	0	0	0	1	1	1	1
14	1	1	1	1	2	1	1	2	1	1	1	1	1	1
15	1	1	-	1	1	1	1	1	1	1	1	1	1	1
16	1	1	1	1	2	1	1	1	0	0	2	1	1	0
17	1	1	1	1	0	1	1	0	1	2	1	1	1	1
18	1	1	1	1	1	1	1	1	0	0	1	1	2	1
19	1	1	-	1	0	0	0	0	0	0	2	1	1	1
20	-*	1	1	1	1	1	1	1	0	0	1	2	1	0
21	1	1	-	1	1	1	1	2	1	1	1	1	1	1
22	1	1	1	1	2	1	1	1	0	0	1	1	1	0
23	1	1	-	1	1	1	1	1	1	2	1	1	1	1
24	1	1	1	1	1	1	1	1	1	0	1	1	1	1
25	2	1	1	1	1	1	1	1	1	2	1	1	2	1

* "-" indicates no trip was found. "0" indicates no transfers, i.e. a direct trip.

TABLE 1 (Continued)
Transfers for LABCOR

NETWORK NO. / CASE	15	16	17	18	19	20	21	22	23	24	25	26
1	1	1	4	2	1	1	1	1	2	2	1	2
2	2	1	1	2	1	1	1	1	1	1	1	1
3	1	1	3	0	2	1	1	1	2	1	1	1
4	1	1	1	2	1	1	1	1	1	2	2	1
5	1	1	1	1	1	1	1	2	1	1	1	2
6	1	1	3	2	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1	1	1
8	2	1	4	0	2	0	0	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1	1	1
10	1	1	1	1	2	1	1	1	1	1	1	1
11	1	1	0	1	3	1	1	1	1	1	1	1
12	2	1	1	1	2	1	1	1	1	1	1	1
13	1	1	1	2	1	1	2	1	1	1	1	1
14	1	1	1	0	1	1	1	2	2	1	2	1
15	1	2	1	1	1	1	1	1	2	2	1	2
16	1	1	3	1	1	0	1	1	1	1	1	1
17	1	1	0	1	2	1	1	1	1	1	1	1
18	1	1	2	1	3	1	1	2	1	2	2	2
19	1	1	1	1	1	1	1	1	2	1	1	1
20	0	1	1	2	0	1	1	1	1	1	1	1
21	1	1	1	1	1	1	1	1	2	1	2	1
22	1	1	1	1	0	1	1	2	2	1	0	1
23	1	1	2	1	1	2	1	1	3	1	1	1
24	1	1	2	0	1	2	1	1	1	3	1	1
25	1	1	0	1	1	3	1	2	2	2	2	1

TABLE 1 (Continued)
Transfers for LABSET

NETWORK NO. / CASE	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	1	2	1	0	1	1	0	1	2	1	2	1	1
2	2	1	1	1	1	1	1	1	1	2	1	2	1	1
3	1	1	1	1	0	0	0	0	1	1	1	2	1	1
4	1	1	1	1	0	0	0	0	1	2	2	2	1	1
5	1	1	2	1	2	1	1	2	1	1	1	2	1	1
6	1	1	-	1	3	1	1	1	2	2	0	2	1	1
7	1	2	1	1	0	0	0	0	0	0	2	1	1	1
8	2	2	1	1	0	0	0	0	1	2	1	2	2	0
9	1	1	1	1	1	1	1	1	0	0	1	1	2	1
10	1	1	1	1	0	0	0	0	0	0	1	2	2	1
11	1	1	1	1	2	1	1	2	0	0	1	1	1	1
12	2	1	1	2	0	0	0	0	0	0	1	1	1	1
13	2	1	-	1	0	0	0	0	0	0	1	1	1	1
14	1	1	1	1	2	1	1	2	1	1	1	2	1	1
15	2	1	-	1	1	1	1	1	1	1	1	1	1	1
16	1	2	1	1	3	1	1	1	0	0	2	2	2	0
17	1	2	1	1	0	1	1	0	1	2	2	1	1	2
18	1	1	1	1	1	1	1	1	0	0	1	1	2	1
19	1	1	-	1	0	0	0	0	0	0	2	1	1	1
20	-	1	1	1	1	1	1	1	0	0	1	2	1	0
21	1	1	-	1	1	1	1	2	1	1	1	1	1	1
22	2	1	1	1	2	1	1	1	0	0	1	2	1	0
23	1	1	-	1	1	1	1	1	1	2	1	1	1	2
24	2	1	1	1	1	1	1	1	1	1	1	2	1	2
25	2	1	1	1	1	1	1	1	1	2	1	1	2	2

TABLE 1 (Continued)

Transfers for LABSET

CASE \ NETWORK NO.	15	16	17	18	19	20	21	22	23	24	25	26
1	1	1	4	2	1	1	1	1	2	2	1	2
2	2	1	2	2	1	1	2	1	1	1	1	1
3	1	1	3	0	2	1	2	1	2	1	1	1
4	1	2	1	2	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	2	2	2	2	2	2
6	1	1	3	2	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1	1	1
8	2	2	4	0	2	0	0	1	1	1	1	1
9	1	2	1	1	1	1	2	1	2	1	1	1
10	1	1	1	1	2	1	1	1	1	1	1	1
11	1	1	0	1	3	1	1	2	2	2	2	2
12	2	1	1	1	2	1	1	1	1	1	1	1
13	1	1	1	2	1	1	2	1	1	1	1	1
14	1	1	1	0	1	1	1	2	2	1	2	1
15	1	2	1	1	1	1	2	1	2	2	1	2
16	1	1	3	1	1	0	1	1	1	1	1	1
17	1	1	0	1	2	1	1	1	1	1	1	1
18	1	2	2	1	3	1	2	2	2	2	2	2
19	1	2	1	1	1	1	2	1	2	1	1	1
20	0	1	1	2	0	1	1	1	1	1	1	1
21	1	1	1	1	1	2	1	2	2	2	2	2
22	1	2	1	1	0	1	1	2	2	1	0	1
23	1	1	2	1	1	2	1	1	2	1	1	1
24	2	1	2	0	1	2	1	2	1	3	2	2
25	2	1	0	1	1	3	1	2	2	2	2	1

TABLE 1 (Continued)

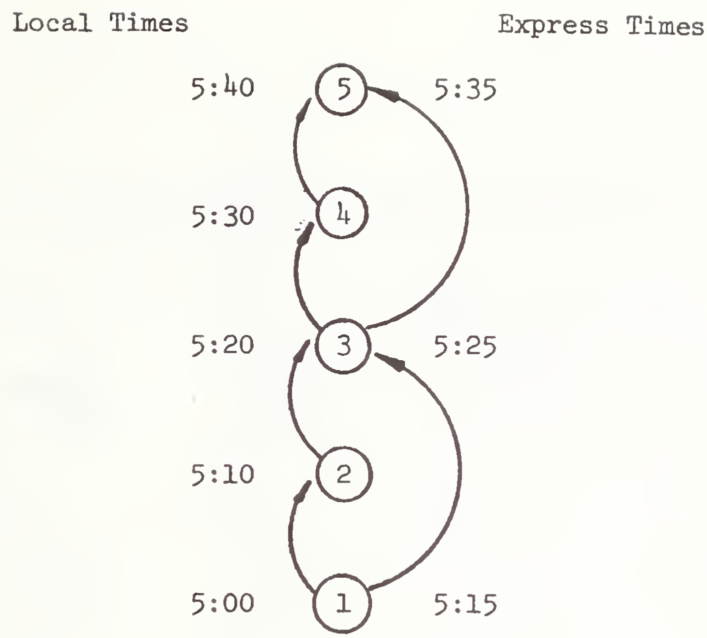
Transfers for TIMEXD

NETWORK NO. / CASE	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	3	2	1	0	1	1	0	1	2	1	1	1	1
2	2	3	1	1	1	1	1	1	1	2	1	2	1	1
3	1	1	2	3	0	0	0	0	1	1	1	2	1	1
4	1	1	1	1	0	0	0	0	1	2	2	1	1	1
5	1	1	1	1	2	1	1	2	1	1	1	1	1	1
6	1	1	1	1	3	1	1	1	2	2	0	2	1	1
7	1	4	-	1	0	0	0	0	0	0	2	1	1	1
8	3	4	3	1	0	0	0	0	1	2	1	1	2	0
9	1	1	2	1	1	1	1	1	0	0	1	1	2	1
10	1	1	1	1	0	0	0	0	0	0	1	1	2	1
11	1	3	1	1	2	1	1	2	0	0	1	1	1	1
12	1	3	1	1	0	0	0	0	0	0	1	1	1	1
13	2	3	-	1	0	0	0	0	0	0	1	1	1	1
14	1	3	1	1	2	1	1	2	1	1	1	1	1	1
15	2	1	-	1	1	1	1	3	1	1	1	1	1	1
16	1	1	1	1	3	2	1	1	0	0	2	1	2	0
17	1	1	1	1	0	1	1	0	1	2	1	1	1	1
18	1	1	1	1	1	1	1	1	0	0	1	1	2	1
19	1	1	-	1	0	0	0	0	0	0	2	1	1	1
20	-	1	3	1	1	1	1	1	0	0	1	1	1	0
21	2	1	-	1	1	1	1	3	1	1	1	1	1	1
22	2	1	3	1	2	1	1	1	0	0	1	2	1	0
23	1	1	-	1	1	1	1	1	1	2	1	1	1	2
24	2	1	3	1	1	1	1	1	1	1	1	1	1	2
25	2	1	1	1	1	1	1	1	1	2	1	1	2	2

TABLE 1 (Continued)

Transfers for TIMEXD

NETWORK NO. / CASE	15	16	17	18	19	20	21	22	23	24	25	26
1	1	1	4	3	2	2	1	1	3	2	1	2
2	2	1	3	3	1	1	2	1	1	1	1	1
3	1	1	3	0	2	1	2	1	2	1	1	1
4	1	1	2	2	1	1	1	1	2	1	3	1
5	1	1	4	2	1	1	2	2	2	2	2	2
6	1	1	3	2	1	1	1	1	1	2	1	3
7	1	1	1	1	1	1	1	1	1	1	1	1
8	3	1	6	0	2	0	0	1	1	1	1	1
9	1	1	1	1	1	2	2	1	2	1	1	1
10	1	1	1	1	2	1	1	2	3	1	2	2
11	1	1	0	1	3	1	1	2	2	1	2	1
12	2	1	1	1	2	1	1	1	1	1	1	1
13	1	1	2	2	1	1	2	1	1	1	1	1
14	1	1	1	0	3	1	1	2	3	3	2	1
15	1	2	1	1	1	1	2	1	2	2	1	2
16	1	1	3	1	1	0	1	1	1	1	1	1
17	1	1	0	1	2	1	1	1	2	1	1	1
18	1	1	2	1	3	1	2	2	2	2	2	2
19	1	2	1	1	1	1	2	2	2	1	1	1
20	0	1	1	2	0	1	1	1	1	1	1	1
21	1	1	1	2	1	3	1	2	2	2	2	2
22	1	1	1	2	0	3	1	2	2	2	0	2
23	1	1	2	1	1	2	1	2	3	2	2	3
24	2	1	2	0	1	3	1	1	2	3	1	1
25	2	1	0	3	2	3	1	2	2	2	2	1



<u>No Transfer</u>

$$\begin{array}{ccc} 1 & \xrightarrow{\text{Express}} & 5 \\ 5{:}15 & & 5{:}35 \end{array}$$

<u>One Transfer</u>

$$\begin{array}{ccccc} 1 & \xrightarrow{\text{Local}} & 3 & \xrightarrow{\text{Express}} & 5 \\ 5{:}00 & & 5{:}20,\ 5{:}25 & & 5{:}35 \end{array}$$

FIGURE 6

<u>Example of Overtake</u>

FIGURE 7

Overtake Occurrences in the Test Runs

Itineraries Produced by LABCOR and LABSET

$\underset{728}{22} \xrightarrow{22} \underset{740,746}{49} \xrightarrow{\overset{\text{express}}{39}} \underset{751}{53}$*

$\underset{746}{69} \xrightarrow{44} \underset{753,764}{65} \xrightarrow{\overset{\text{express}}{49}} \underset{776}{2}$

$\underset{721}{18} \xrightarrow{41} \underset{723,744}{17} \xrightarrow{\overset{\text{express}}{48}} \underset{754,757}{71} \xrightarrow{17} \underset{760}{70}$

$\underset{741}{69} \xrightarrow{44} \underset{743,754}{67} \xrightarrow{\overset{\text{express}}{50}} \underset{764}{13}$

Itineraries Produced by TIMEXD

$\underset{728}{22} \xrightarrow{22} \underset{740,742}{49} \xrightarrow{\overset{\text{local}}{6}} \underset{746,748}{51} \xrightarrow{\overset{\text{express}}{39}} \underset{751}{53}$

$\underset{746}{69} \xrightarrow{44} \underset{753,756}{65} \xrightarrow{\overset{\text{local}}{29}} \underset{765,767}{47} \xrightarrow{\overset{\text{express}}{49}} \underset{776}{2}$

$\underset{721}{18} \xrightarrow{41} \underset{723,728}{17} \xrightarrow{\overset{\text{local}}{26}} \underset{749,751}{53} \xrightarrow{\overset{\text{express}}{48}} \underset{754,757}{71} \xrightarrow{17} \underset{760}{70}$

$\underset{741}{69} \xrightarrow{44} \underset{743,745}{67} \xrightarrow{\overset{\text{local}}{31}} \underset{757,760}{31} \xrightarrow{\overset{\text{express}}{50}} \underset{764}{13}$

---

*Itineraries are listed in the form:

$\underset{\text{depart. time}}{\text{stop}} \xrightarrow{\text{route number}} \underset{\text{arr., depart. time time}}{\text{stop}} \xrightarrow{\text{route number}} \underset{\text{arr. time}}{\text{stop}}$

TABLE 2

Comparison of Paths for TIMEXD Using First Path Encountered to a Node Versus Last Path to That Node

TRIP	NUMBER OF TRANSFERS		
	LABCOR	TIMEXD (last arrival)	TIMEXD (first arrival)
1	1	3	1
2	1	3	1
3	1	1	1
4	1	1	1
5	1	1	2
6	1	1	2
7	1	4	4
8	1	4	4
9	1	1	2
10	1	1	2
11	1	3	1
12	1	3	1
13	1	3	1
14	1	3	1
15	1	1	2
16	1	1	2
17	1	1	4
18	1	1	4
19	1	1	3
20	1	1	3
21	1	1	1
22	1	1	1
23	1	1	1
24	1	1	1
25	1	1	1
Average	1.0	1.68	1.88

In conclusion, LABCOR provides the most desirable path output since it always produces an itinerary with the minimum number of transfers. The other two algorithms output a small but significant number of trips with extra transfers (9 percent for LABSET and 14 percent for TIMEXD). When the overtake problem occurs, using TIMEXD may lead to an extra transfer from a local to an express because the express starts later than the local and passes it enroute, arriving first at the destination. Varying the criterion for choosing between ties does not result in a decrease in the number of extra transfers for TIMEXD.

4.2 Computer Storage Required by the Algorithms

The storage required by the algorithms is largely a function of the size and configuration of the network and schedules. The particular form in which the algorithms have been coded for this study does not attempt to optimize use of storage of the basic network data. For instance, in LABCOR and LABSET the routes are stored in a doubly indexed array, ROUTE (i,j), in which the entry for i,j is the jth stop on route i. Thus the size of the array is the number of routes by the maximum number of stops per route. If one or a few routes are long while all others are short, this wastes much storage. The compensating benefit is that of easier reference to a desired piece of data. More efficient storage would involve more complicated indexing than was used in the programming. The decision in favor of the simpler representation was made in the interest of facilitating quick coding. In addition, it is the LABCOR and LABSET algorithms which are most severely affected by this decision, and they require less storage than does TIMEXD for the same network. Since we intended to run all three algorithms on the same problems, we were less concerned with wasting storage space in programming LABCOR and LABSET.

In estimating the storage required by LABCOR and LABSET we will use the following notation:

- R number of routes
- S number of stops
- L maximum number of stops per route
- K maximum number of routes per stop
- D number of vehicle departures
- T number of possible time intervals (e.g. 1440 minutes per day)

Then the storage required by algorithm LABCOR is approximately:

$$3R + 8S + (R+D)L + S\cdot K.$$

Similarly the storage required by the algorithm LABSET is:

$$3R + 10S + (R+D)L + S\cdot K + T.$$

Additional storage is required for printing paths, but these expressions contain the major elements requiring computer space.

Notation used in estimating the storage for TIMEXD follows:

- N number of time-expanded nodes
- A number of arcs in the time-expanded network
- T number of possible time intervals (as above)
- D number of vehicle departures
- ℓ average number of stops per route
- k average number of routes per stop.

The algorithm requires

$$5N + 2A + T$$

storage locations, exclusive of path printing and incidentals. To relate this to the other two algorithms we approximate N and A as follows:

$$N = D\cdot\ell$$

$$A = D\cdot\ell + N\cdot k = D\cdot\ell(k+1).$$

Assuming each vehicle arrival gives rise to a new node, we arrive at an overestimate to the number of nodes. Arcs are of two kinds, those which represent vehicle trips--whose number is the number of vehicle departures times one less than the number of stops per route (approximated as $D\ell$)--, and transfer arcs, whose number is the number of routes stopping at a node times the number of nodes, under the assumption that all possible transfers at each node are available and reasonable. The total requirement then becomes

$$6D\ell + 2D\ell k + T.$$

These approximations overestimate storage requirements but are useful for comparison purposes.

In a square grid network of size P x P:

$$S = P^2$$
$$R = 4P$$
$$L = \ell = P$$
$$K \approx k = 4$$

so the storage required by LABCOR is

$$12P + 16P^2 + DP,$$

for LABSET is

$$12P + 18P^2 + DP + T,$$

and for TIMEXD is

$$14DP + T.$$

For several of the test cases P was about 15, D was about 300 and T was 1440, making the storage requirements 8200 for LABCOR, 10170 for LABSET, and 64440 for TIMEXD. We note again that the formula for TIMEXD overestimates the storage required, in this case since the actual number of transfers per interior node is 3, rather than 4, and the number possible at peripheral nodes is 2 or 1.

The difference in storage requirements is greatest for the sort of situation described above when $L = \ell$ and $K = k$. For a radial network it is likely that $K >> k$, since all routes in or out along radials stop at the center node. Thus in a radial network with 6 spokes, at least 12 routes (and more if there are spike routes *) stop at the center node, whereas most other nodes have at most 4. A full beltway would have at most 7 stops (one stop repeated) but a radial route could have as many as desired. A partial beltway might have only 2 stops. Therefore there is a great variability in the number of stops per route and routes per stop, leading to a difference between K and k and between L and ℓ. Similarly in a rectangular network which is long and thin, storage must be provided as if all routes had as many stops as the longer routes. Thus whenever $K > k$ and/or $L > \ell$, LABCOR and LABSET, because of inefficient storage design, require more storage than they actually use, whereas TIMEXD can be sized more tightly. In spite of not using storage most efficiently, for most of the test cases LABCOR and LABSET required significantly less storage than TIMEXD.

4.3 Comparison of Computation Times

Timings for the 26 test runs and the 25 cases for each run are listed in Table 3. The timings include only processing time, no input or output operations, since test cases were chosen to be small enough not to require reference to external storage. For each run, timings were made on all three algorithms at the same time of day (actually within the same computer run), to ensure that the computer environments were as comparable as possible.

There are many problems with timing an algorithm in a multiprocessing environment, such as the UNIVAC 1108 under the EXEC 8 operating system, in which several programs are active in various stages of processing at one instant in time. In calculating the run time for each algorithm on each case, we used a computer subroutine, CPUSUP, available at NBS for summing the CPU time of a designated section of one program only. However, our experience has been that timings of the same problem

*Spike routes are ones which begin or end at the center node, include stops along one spoke and then deviate from that spoke to one final stop not on any spoke.

TABLE 3
ALGORITHM TIMINGS FOR EACH PATH
Timings for LABCOR (Milliseconds per Path)

CASE \ NETWORK NO.	1	2	3	4	5	6	7	8	9	10	11	12	13
1	164	151	85	106	24	20	21	22	33	29	143	159	158
2	120	170	98	108	26	23	24	27	32	28	102	129	122
3	165	152	84	100	29	21	21	23	29	30	142	159	158
4	121	171	99	107	34	23	25	26	27	29	103	137	130
5	164	150	82	100	26	23	24	25	32	28	142	171	158
6	121	172	98	106	27	23	24	24	30	29	103	140	133
7	155	147	65	100	29	29	31	32	33	29	137	166	176
8	136	179	96	112	33	30	30	31	34	29	124	146	174
9	153	146	64	100	32	29	29	29	27	28	138	166	187
10	134	181	96	111	32	25	26	32	28	28	123	144	176
11	155	148	64	100	25	22	23	28	26	27	139	161	184
12	135	179	95	112	28	25	28	28	28	28	123	144	158
13	176	145	22	100	35	31	31	32	24	28	138	168	175
14	118	170	94	106	36	29	32	34	28	31	107	139	156
15	176	143	24	100	29	25	25	30	26	28	138	157	178
16	118	170	94	105	24	23	25	24	27	27	108	129	156
17	175	145	23	100	24	22	21	22	28	30	138	155	173
18	118	171	95	106	32	27	28	29	28	28	107	128	153
19	175	148	7	99	29	27	28	32	28	27	139	164	168
20	97	172	93	106	23	22	21	24	28	29	95	140	153
21	173	150	8	101	27	22	24	27	27	28	139	165	168
22	96	169	92	105	24	21	22	23	26	28	95	139	153
23	173	148	8	100	32	30	32	33	27	29	139	165	168
24	97	169	92	105	25	21	24	25	28	29	96	140	152
25	159	167	92	104	31	27	27	34	26	28	134	167	186
AVERAGE	142.96	160.52	70.80	103.96	28.64	24.80	25.84	27.84	28.46	28.48	123.68	151.12	162.12
STD.DEV.	27.42	13.04	33.59	4.10	3.86	3.38	3.58	3.92	2.43	.96	17.83	14.0	16.86

TABLE 3 (Continued)

Timings for LABCOR (Milliseconds per Path)

NETWORK NO. / CASE	14	15	16	17	18	19	20	21	22	23	24	25	26
1	151	165	36	47	65	47	87	52	61	63	52	65	52
2	105	121	41	41	37	70	82	54	61	69	52	69	53
3	154	164	35	48	59	69	88	50	61	63	53	68	51
4	109	121	41	47	53	71	88	55	66	68	52	74	53
5	153	165	36	41	50	69	98	53	65	70	59	77	55
6	108	122	41	44	66	79	100	56	65	73	55	56	53
7	150	160	37	24	82	74	98	50	66	84	56	64	57
8	108	132	41	45	52	78	94	54	66	84	57	59	57
9	152	161	37	36	77	74	98	64	65	65	62	60	54
10	101	133	42	54	82	72	101	60	62	64	58	65	53
11	150	161	38	43	65	65	100	57	55	64	52	52	51
12	101	131	42	44	88	43	100	55	63	59	62	66	63
13	156	170	35	52	47	66	88	50	60	71	51	60	57
14	88	112	46	47	57	80	89	58	69	69	53	63	62
15	155	170	36	33	48	70	88	58	66	78	56	50	63
16	89	111	45	51	77	68	81	59	67	75	57	59	60
17	155	169	36	44	69	81	81	57	67	73	57	59	57
18	88	121	46	44	50	83	81	54	67	72	56	60	61
19	154	177	38	40	51	83	81	57	63	79	53	54	54
20	81	99	41	33	34	70	81	60	62	60	52	54	56
21	162	170	37	44	56	67	85	56	62	75	50	54	54
22	94	100	42	39	49	68	85	58	64	67	55	68	60
23	176	175	39	42	47	70	86	51	67	84	59	63	59
24	94	97	41	43	35	68	85	62	64	66	59	60	55
25	180	166	41	51	49	70	85	58	69	69	55	68	53
AVERAGE	128.56	142.92	39.60	43.08	57.8	70.2	89.2	55.92	64.12	70.56	55.32	61.88	56.12
STD.DEV.	32.17	27.43	3.30	6.63	14.93	9.27	7.14	3.72	3.18	7.21	3.33	6.71	3.7

TABLE 3 (Continued)

Timings for LABSET (Milliseconds per Path)

NETWORK NO. / O-D PAIR	01	02	03	04	05	06	07	08	09	10	11	12	13
1	50	115	29	51	19	24	23	17	32	29	47	166	99
2	73	121	55	68	18	20	18	17	20	21	60	139	84
3	81	108	70	74	3	3	4	2	18	17	39	99	124
4	55	111	32	36	8	7	9	7	22	24	82	42	62
5	118	146	67	74	14	21	19	22	17	16	141	159	149
6	118	156	92	105	21	17	20	17	19	18	116	124	116
7	98	132	72	66	28	30	32	31	8	8	135	167	159
8	118	159	91	46	6	5	5	5	29	27	107	135	153
9	41	108	55	72	20	26	22	21	11	9	143	181	99
10	42	117	65	95	6	7	17	11	13	11	130	147	101
11	89	122	30	70	15	19	19	17	14	12	121	100	68
12	95	131	59	108	15	17	16	16	6	7	111	80	83
13	79	134	33	16	3	3	3	3	22	19	38	62	91
14	102	156	41	61	31	30	35	31	21	21	70	55	70
15	121	109	34	54	18	16	15	20	22	20	94	178	114
16	53	114	80	88	22	22	27	23	20	18	78	144	116
17	55	117	24	79	20	21	20	17	30	28	141	150	54
18	90	131	54	73	20	19	18	15	13	12	100	91	85
19	147	143	18	73	10	15	8	9	16	15	137	172	164
20	108	158	76	105	9	9	10	14	15	11	84	144	131
21	106	117	18	35	23	16	18	21	16	18	115	85	35
22	67	125	43	77	24	20	20	22	8	9	57	130	88
23	59	108	20	80	14	12	10	11	20	19	108	185	82
24	85	110	86	78	11	18	26	25	21	19	97	156	114
25	151	168	88	109	14	17	10	11	23	27	134	165	125
AVERAGE	88.04	128.64	53.28	71.72	15.68	16.44	16.98	16.20	18.24	17.40	99.40	130.24	102.64
STD.DEV.	31.04	18.97	24.58	23.48	7.45	7.59	8.25	7.69	6.58	6.46	33.44	41.94	33.36

TABLE 3 (Continued)

Timings for LABSET (Milliseconds per Path)

NETWORK NO. / CASE	14	15	16	17	18	19	20	21	22	23	24	25	26
1	85	99	31	31	60	40	41	53	38	47	50	55	48
2	56	70	20	27	31	32	35	30	20	31	21	28	21
3	55	71	28	38	49	23	37	44	47	47	43	51	41
4	79	50	40	27	49	14	28	63	52	50	48	67	51
5	147	121	35	35	47	32	57	48	36	50	38	46	37
6	95	118	40	38	38	49	27	34	40	28	39	35	34
7	69	99	36	17	16	55	28	46	48	38	49	64	49
8	109	121	42	36	68	57	14	4	7	16	7	10	7
9	118	39	29	31	54	29	39	53	35	47	30	45	27
10	85	39	36	10	38	44	35	45	42	40	39	55	43
11	136	88	10	4	28	29	18	42	30	34	27	36	30
12	54	89	21	39	30	21	52	15	21	21	22	25	22
13	135	169	34	23	25	13	16	30	39	36	46	42	38
14	73	101	17	26	9	56	12	66	35	48	49	46	51
15	90	120	36	23	38	30	25	42	26	27	27	33	27
16	100	51	36	34	50	10	15	39	27	26	27	30	26
17	99	52	24	22	24	51	39	40	23	35	24	25	23
18	60	89	37	32	49	39	12	32	42	32	43	44	41
19	140	159	39	15	50	23	15	51	27	41	33	28	27
20	91	111	41	9	23	36	64	22	29	22	27	28	27
21	76	107	13	16	54	50	71	56	49	39	49	48	52
22	91	70	37	21	57	6	32	54	36	39	46	57	49
23	110	59	23	38	40	61	11	44	50	47	51	52	45
24	73	86	20	38	20	24	37	39	40	39	34	41	37
25	156	176	41	9	50	30	41	24	36	49	42	40	37
AVERAGE	95.28	94.16	30.64	25.56	39.88	34.16	32.04	40.64	35.0	37.16	36.44	41.24	35.60
STD.DEV.	29.77	37.91	9.51	10.71	15.14	15.70	16.58	14.54	10.84	9.87	11.55	13.58	11.67

TABLE 3 (Continued)

Timings for TIMEXD (Milliseconds per Path)

NETWORK NO. / O-D PAIR	01	02	03	04	05	06	07	08	09	10	11	12	13
1	36	58	24	38	35	55	51	38	43	49	27	98	49
2	44	55	31	33	40	42	48	40	22	28	34	54	53
3	45	54	64	49	9	9	12	9	21	21	27	37	68
4	34	51	27	25	15	21	20	15	24	34	42	29	45
5	64	78	65	50	31	52	52	61	21	20	84	78	85
6	64	81	71	48	38	40	43	33	25	23	54	52	76
7	41	78	109	45	26	35	50	36	12	11	57	85	87
8	66	81	117	29	14	16	19	15	49	49	43	55	87
9	26	68	53	46	31	43	48	42	14	14	73	90	50
10	28	65	38	41	13	18	30	24	15	15	62	58	47
11	37	69	37	47	30	38	42	36	16	16	48	50	41
12	44	72	37	46	23	28	41	34	12	11	44	40	41
13	41	78	79	24	9	9	12	8	24	23	32	37	47
14	47	83	35	33	69	68	84	68	24	30	32	34	47
15	62	62	88	40	34	33	35	42	27	23	54	92	49
16	34	63	49	40	58	52	47	40	20	20	47	57	70
17	31	70	38	52	37	51	50	39	56	49	80	81	37
18	43	77	35	36	37	43	40	38	15	15	52	45	54
19	86	87	60	51	18	21	21	16	18	16	77	84	111
20	61	88	77	55	17	22	23	33	20	17	42	51	72
21	56	59	61	32	52	38	42	48	22	21	64	32	29
22	48	57	61	38	56	44	65	53	14	12	35	47	48
23	34	54	67	52	24	25	31	24	26	24	53	94	40
24	54	52	82	43	24	35	51	45	28	22	44	55	64
25	69	63	78	53	23	27	28	23	27	44	87	95	68
AVERAGE	47.80	68.12	59.32	41.84	30.52	34.60	39.40	34.40	24.20	24.28	51.76	61.20	58.60
STD.DEV.	14.97	11.60	24.80	8.91	15.70	15.00	16.76	15.13	11.21	11.91	17.66	22.63	19.57

TABLE 3 (Continued)

Timings for TIMEXD (Milliseconds per Path)

NETWORK NO. / CASE	14	15	16	17	18	19	20	21	22	23	24	25	26
1	56	70	27	50	61	44	31	40	44	79	72	60	60
2	47	53	21	42	32	36	23	28	24	41	29	26	23
3	45	49	30	53	52	28	25	38	58	76	58	60	46
4	53	43	27	33	50	20	23	77	82	73	86	107	86
5	102	79	37	55	43	36	37	40	44	71	45	42	34
6	57	72	27	62	38	43	20	29	48	31	44	30	38
7	41	56	76	28	17	53	21	36	57	43	60	54	49
8	61	80	29	61	92	49	13	8	12	16	11	12	11
9	59	36	27	45	57	30	24	38	41	59	33	40	28
10	52	36	26	15	39	39	23	35	57	53	52	64	52
11	69	46	19	12	20	31	15	35	40	43	36	47	35
12	43	52	22	63	25	27	33	16	29	35	28	30	27
13	84	101	29	35	24	17	14	23	48	48	48	44	40
14	47	56	20	44	8	62	11	66	42	66	69	49	70
15	57	68	40	38	32	35	16	39	33	39	31	31	30
16	63	42	29	47	41	17	12	30	25	23	27	23	23
17	59	35	23	27	17	38	23	32	28	42	27	27	22
18	44	51	27	50	43	31	11	29	47	36	48	41	38
19	79	92	51	26	47	22	13	45	32	57	33	31	26
20	64	70	31	21	24	32	51	20	34	33	31	33	29
21	54	63	19	29	69	48	66	52	87	45	73	60	73
22	64	54	29	34	52	11	23	48	43	59	59	74	62
23	70	40	43	59	35	50	17	37	70	62	76	61	56
24	56	61	21	65	20	26	27	37	54	57	42	53	46
25	125	155	44	12	54	29	29	28	46	81	55	47	47
AVERAGE	62.04	62.04	30.96	40.24	39.68	34.16	24.04	36.24	45.00	50.72	46.92	45.84	42.04
STD.DEV.	19.06	25.93	12.47	16.36	19.04	12.44	12.71	14.43	17.42	17.62	18.84	19.82	18.29

may differ by as much as 20 percent depending on the time of day and machine load at the time. By timing all three algorithms on the same network in immediate sequence, we sought to make cross-comparisons on the same problem appropriate and valid. Most runs were made in low activity periods to minimize the influence of other programs being run on the computer at the same time. (It is generally true that the timing of the same program will be greater during periods in which the computer has a heavy load.) Thus care should be exercised in imputing precision to the timings reported here and in extrapolating from them. However, the testing procedure was designed to facilitate comparisons of performance among algorithms.

Table 4 contains a summary of the average and standard deviation of the 25 timings for each of the 26 test runs along with the size parameters describing each test network. TIMEXD is the fastest on the average, but in more than half of the runs LABSET was as fast or faster. In general LABSET is faster than TIMEXD on networks with fewer nodes. Since the networks used in these tests are all fairly small, compared to most real transit systems, one would generally expect TIMEXD to calculate itineraries more quickly for real transit systems. In all but four small networks, LABCOR took more time than did either of the other two algorithms. It is always slower than LABSET, but beats TIMEXD on very small networks. However, we note again that LABCOR always produces paths which are at least as desirable and often more desirable, because they have fewer transfers, than those produced by the other algorithms. On the average for these test runs, these better paths required 77 percent more time to calculate, and the time difference was greater for larger networks. For example, for the five 15 by 15 grid networks, TIMEXD was on the average 2.5 times faster than LABCOR.

The average standard deviations for the LABCOR calculations were smaller than those of either of the other two algorithms, indicating that for a particular network the timings of different cases using LABCOR were closer together than those of LABSET or TIMEXD. However, the standard deviation of the average times for the 26 networks was greatest for LABCOR, which is indicative of greater variation of the timing from network to network. TIMEXD varies least with network. Thus LABCOR has most consistent timing for one network but varies most among different networks, whereas LABSET and TIMEXD show greater variability in timing the different cases for the same network, and TIMEXD varies least from network to network with LABSET varying more but less than LABCOR.

Table 4 lists for each of the 26 test networks the number of nodes and the number of vehicle departures (or runs) for that particular network. These two parameters, along with the product of the two, were thought to most directly characterize the network size. Correlation coefficients and Spearman rank correlation coefficients were calculated to assess the degree of relationship between the timings and each of these parameters. The correlation coefficients are displayed in Table 5. Correlation coefficients are statistics with values between -1 and +1. A coefficient of 0 indicates no association; coefficients close to +1 indicate a high association in which as one variate grows the second grows also; a coefficient of nearly -1 also indicates a high degree of association but one in which as one variate grows larger the other becomes smaller.

TABLE 4

COMPARISON OF TIMINGS OF THE THREE ALGORITHMS

NETWORK DESCRIPTION					TIMINGS (MILLISECONDS)**					
					LABCOR		LABSET		TIMEXD	
NO.	TYPE	SIZE *	NODES	RUNS	μ	σ	μ	σ	μ	σ
1	G	15 15	225	285	143	27	88	31	48	15
2	G	15 15	225	288	161	13	129	19	68	12
3	G	12 14	168	234	71	34	53	25	59	25
4	G	12 14	168	328	104	4	72	23	42	9
5	R	6 11	40	800	29	4	16	7	31	16
6	R	6 11	40	800	25	3	16	8	35	15
7	R	6 11	40	960	26	4	17	8	39	17
8	R	6 11	40	960	28	4	16	8	34	15
9	R	4 16	50	480	28	2	18	7	24	11
10	R	4 16	50	480	28	1	17	6	24	12
11	G	10 20	200	280	124	18	99	33	52	18
12	G	5 40	200	410	151	14	130	42	61	23
13	G	15 15	225	285	162	17	103	33	59	20
14	G	15 15	225	285	128	32	95	30	62	19
15	G	15 15	225	285	143	27	94	38	62	26
16	G	8 8	64	448	40	3	31	10	31	13
17	G	10 8	80	252	43	7	26	11	40	16
18	R	8 7	80	334	58	15	40	15	40	19
19	G	10 12	120	220	70	9	34	16	34	12
20	R	9 6	120	192	89	7	32	17	24	13
21	G	9 9	81	270	56	4	41	15	36	14
22	G	9 9	81	270	64	3	35	11	45	17
23	G	9 9	81	270	71	7	37	10	51	18
24	G	9 9	81	270	55	3	36	12	47	19
25	G	9 9	81	270	62	7	41	14	46	20
26	G	9 9	81	270	56	4	36	12	42	18
AVERAGE					77.5	10.5	52.0	17.7	43.7	16.6
STD. DEV. OF AVG.					46.4	9.8	36.3	10.6	12.7	4.2

*For a grid network (G), 'size' is given by the numbers of horizontal and vertical grid elements; for a radial (R), by the numbers of radials and beltways.

**Statistics for 25 origin-destination pairs in each network.

TABLE 5

CORRELATION OF TIMINGS AND NETWORK

SIZE PARAMETERS

Correlation Coefficients

	LABCOR	LABSET	TIMEXD
NODES	.9653	.9343	.7867
RUNS	-.5122	-.4226	-.3711
NODES × RUNS	.8397	.8907	.7076

Spearman Rank Correlation Coefficients

	LABCOR	LABSET	TIMEXD
NODES	.9641	.9122	.7596
RUNS	-.3982	-.3082	-.2300
NODES × RUNS	.5498	.5329	.4981

The negative correlation between timings and the number of runs seems at first surprising, since one would expect that as the number of runs increases the number of possibilities to be checked at least remains the same and might (particularly for TIMEXD) grow larger. The negative correlation arises in this data set because we were attempting to test fairly large networks, where 'large' is measured relative to the amount of computer storage available to us, 65,000 words. Therefore the product of numbers of nodes and runs (vehicle departures) was relatively fixed. Networks with many nodes had fewer runs, and networks with few nodes had many runs. Timing is definitely positively correlated with the number of nodes, but because of our choice of test examples, the number of nodes was negatively correlated with the number of runs. Thus the timings are negatively correlated with the number of runs. That negative correlation also contributes to the lower correlation of timings with the product of nodes and runs than with the number of nodes alone.

The correlation of timing with the number of nodes is quite high, especially for LABCOR and LABSET as one would expect from the structure of the algorithms in which each step involves examining a new node. The correlation is less pronounced for TIMEXD which must examine time-expanded nodes whose number depends on both the number of network stops and vehicle departures. The unfortunate (for this purpose) design of the set of computer runs to be performed obscures this relationship because of the negative correlation between nodes and runs.

As an aid in interpolating and in further understanding the relationship between the number of nodes and the timings, linear regressions, with the timing as dependent variable and the number of nodes as independent variable, were performed. That is, we obtained values for the coefficients a_0 and a_1 in the following equation:

$$t = a_0 + a_1 N,$$

where t is the timing in milliseconds and N is the number of nodes. The coefficients, their standard deviations and the residual standard deviation of the fit are given in Table 6. This means, for example, that one may use the equation

$$t_{LABCOR} = 2.22 + .637N$$

to estimate the time required to calculate an average itinerary using the program LABCOR on the UNIVAC 1108. Two caveats must be mentioned. Since the networks used to fit these equations varied in size only from 40 nodes to 225 nodes, it is really not appropriate to use the equations for much larger networks. In addition, the timings reported here were obtained on a particular computer and while we have no reason to suppose <u>relative</u> performance would differ on another machine, <u>actual</u> time is likely to be quite different.

TABLE 6

LINEAR REGRESSION COEFFICIENTS FOR THE FIT OF TIMING AS A FUNCTION OF THE NUMBER OF NODES

	a_0	a_1	Std. dev. a_0	Std. dev. a_1	Lack of fit F ratio % point*
LABCOR	2.22	.637	4.81	.035	72.3
LABSET	-4.94	.482	5.14	.038	87.7
TIMEXD	26.82	.143	3.14	.023	88.5

*This statistic compares the standard deviations of repeated observations with the standard deviation of the fit. A value between 3% and 97% indicates an acceptable linear fit.

As an experiment to check algorithm performance on a larger network, LABCOR and LABSET were run on a 40 by 40 grid network, having therefore 1600 nodes. Actual running times for this network are given in Table 7. We also compared the timings actually obtained with those predicted by the linear regression equations. The estimates are 1021 milliseconds for LABCOR and 766 milliseconds for LABSET. The estimate for LABCOR is in error by more than a factor of 2, reinforcing the earlier warning about applying the regression equations outside the range of the original fit. However, the estimate for LABSET is only 14 percent in error, which leads greater credence to this equation in estimating timings for at least medium sized networks. The ratio of the time required by LABCOR to that required by LABSET is greater for this larger network than for all but one of the 26 test runs, indicating that the slope of the line for LABCOR should be greater than for LABSET, as is indeed the case in the regression equations. This means that the time for LABCOR increases at a greater rate with the number of nodes than does that for LABSET.

In [1], computation time for a large network (containing perhaps 3000 nodes) was estimated at approximately one second per itinerary, with the critical time factor being access and transfer of pages of the network and schedule data from peripheral storage to the main memory. Using the regression results in Table 6 we obtain estimates of 1.9 seconds for LABCOR, 1.4 seconds for LABSET, and 0.5 seconds for TIMEXD for the computation time per trip for a large transit system. These estimates do not contain any allowance for input/output, which was estimated in [1] to require an additional one half to one second. If some computations can be done in parallel with the input/output transfers, then the total time could be less than the sum of the two figures, but with the sum as an upper bound, the total time estimates for LABSET and TIMEXD then become 2.4 and 1.5 seconds respectively. Since we know from the timing on a medium-sized network that the regression equation seriously underestimates the time required by LABCOR, in a network of 3000 nodes, it might be expected to be perhaps more than 5 seconds per trip itinerary, a time which may be unacceptably long. The times for LABSET and TIMEXD, although slightly longer than the original estimates in [1], are well within those required to permit demand in a large city to be handled by the information center without irritating delays.

These timings were made on the UNIVAC 1108 computer here at NBS, and the magnitude of computation time on other computers is likelv to be different. However, since the UNIVAC 1108 is a relatively old computer, one would expect that computation times on most other machines would be less than those recorded here or at worst would be of the same order of magnitude.

TABLE 7

ALGORITHM TIMINGS (MILLISECONDS) FOR LABCOR AND LABSET

ON 40 x 40 GRID NETWORK

PAIR	ORG	DST	LABCOR	LABSET
1	432	539	2915	338
2	432	837	2900	1692
3	432	120	2899	1785
4	432	946	2927	717
5	432	336	2898	1007
6	1247	1454	2537	1326
7	1247	793	2537	1063
8	1247	761	2542	663
9	1247	1142	2570	365
10	1247	1278	2538	1446
11	1026	434	1765	1284
12	1026	698	1762	526
13	1026	1074	1767	803
14	1026	1300	1769	586
15	1026	855	1863	674
16	255	848	1856	810
17	255	666	1857	574
18	255	272	1852	748
19	255	43	1853	1639
20	255	1011	1851	591
21	1273	960	1905	545
22	1273	1179	1890	349
23	1273	1440	1882	878
24	1273	1026	1880	269
25	1273	1543	1885	1148
Average			2192	873
Std.dev.			462.3	443.3

4.4 Additional Analyses

Characteristics of the 26 test runs used in the analyses above are described more fully in Table 8. These parameters were chosen in such a way as to allow the analysis of the dependence of timing and path calculations on several factors, such as variability of the minimum transfer time requirement, the network shape, various frequency patterns, and the relative speed and frequency of express and local runs.

In assessing the effect of minimum transfer time on the computation of itineraries we will refer to Table 9. No clear pattern of variation of computation time with increase in transfer time holds for all the algorithms. The behavior of LABCOR timings seems to be exactly opposite to that of the other two. LABCOR computation time increases with an increase in the transfer time at the center node of a radial network and decreases as the transfer time increases in a grid. This pattern of increase and decrease is the same as the pattern of increasing and decreasing numbers of transfers, reflecting LABCOR computation time's dependence on the number of transfers. LABSET and TIMEXD times decrease with the long transfer at the center node of the radial network and increase as the grid transfer times increase. The increase in time for calculating paths in a grid network with longer transfer times may result from the generally longer trip lengths, which require examining more possible trips before finding a shortest one. The decrease in computer time for producing itineraries in a radial network with more time required to transfer at one (central) node may reflect the fact that although trips may be somewhat longer, the total number of competing itineraries is still reduced because many of those utilizing the central node are now effectively blocked.

The average number of transfers required increases when the minimum transfer time at the central node is increased, because some one-transfer trips traveling in one radial and out another become two-transfer trips utilizing a beltway. When the minimum transfer time is increased at <u>all</u> nodes in the grid network, it becomes less desirable to transfer since that requires significantly greater unit time, so the average number of transfers decreases.

A second analysis concerns the effect of grid shape on computation. In a previous study [2] of the performance of shortest-path algorithms, it was noted that some label-correcting algorithms performed very badly on elongated grid networks, because they depended critically on the length of the longest shortest path. In our grid transit network, there was some increase in the computation time as the grid became elongated but it is not as pronounced as with the standard shortest path computation, partly because the average number of transfers does not increase as much as the length of the longest shortest path. A comparison of computation times for networks 11 and 12, respectively a 10 x 20 grid and a 5 x 40 grid (both with 200 nodes), shows increases of 22, 31 and 17 percent respectively for LABCOR, LABSET and TIMEXD with the more

TABLE 8

DESCRIPTIONS OF THE NETWORKS FOR THE TESTS

NO.	TYPE	SIZE	NODES	DESCRIPTION
1	G	15X15	225	N → S: 6 runs; S → N: 3 runs; E → W: 5 runs; W → E: 5 runs
2	G	15X15	225	Edges and center every 5 min., others every 20 min
3	G	12X14	168	N → S and E → W frequently; S → N and W → E not so frequent
4	G	12X14	168	N → S: every 6 min.; E → W every 30 min.
5	R	6R,11B	40	2 min. frequency on radials, 3 min. frequency on beltways; 10 min. transfer time at center, 2 min. transfer elsewhere
6	R	6R,11B	40	2 min. frequency on radials, 3 min. frequency on beltways; 2 min. transfer everywhere
7	R	6R,11B	40	1 min. frequency on radials towards center and on full beltway, 10 min. frequency otherwise; 2 min. transfer everywhere
8	R	6R,11B	40	1 min. frequency on radials towards center and on full beltway, 10 min. frequency otherwise; 10 min. transfer at center, 2 min. transfer elsewhere
9	R	4R,16B	50	5 min. frequency everywhere; 2 min. transfer everywhere
10	R	4R,16B	50	5 min. frequency everywhere; 10 min. transfer at center, 2 min. elsewhere
11	G	10X20	200	N → S: 6 runs; S → N: 3 runs; E → W: 5 runs; W → E: 5 runs
12	G	5X40	200	N → S: 6 runs; S → N: 3 runs; E → W: 5 runs; W → E: 5 runs
13	G	15X15	225	N → S: 6 runs starting at 8:00 S → N: 3 runs starting at 8:15; E → W: 5 runs starting at 8:10 W → E: 5 runs starting at 8:05.
14	G	15X15	225	N → S: 6 runs; S → N: 3 runs; E → W: 5 runs; W → E: 5 runs; transfer time 5 min. everywhere

TABLE 8 (Continued)

NO.	TYPE	SIZE	NODES	DESCRIPTION
15	G	15X15	225	N → S: 6 runs; S → N: 3 runs; E → W: 5 runs; W → E: 5 runs; transfer time 3 min. everywhere
16	G	8X8	64	Multiple period run with N → S and E → W more frequent in period 1, all same frequency in period 2, S → N and W → E more frequent in period 3; speed greatest in period 2
17	G	10X8	80	Local: N → S and W → E have 7 runs, S → N and E → W have 4 runs; Express: N → S and W → E have 4 runs, S → N and E → W have 2 runs
18	R	8R,7B, 12S	80	Local: frequency every 20 min. out from center, every 30 min. in to center, every 15 min. on 2 beltways, every 20 min. on other beltways Express: frequency every 30 min. out from center, every 60 min. in to center, every 30 min. on beltways; 5 min. transfer at center, 2 min. elsewhere
19	G	12X10	120	Local: 4 runs, Express : 2 runs; transfers 2 min.
20	R	9R,6B	120	Local: 5 runs, Express : 3 4uns' 5 ,om/ tramsfer at cemter' 2 min. elsewhere
21	G	9X9	81	Local: 6 runs--10 min. apart, Express: 3 runs--20 min. apart; 2 min. transfer, ratio of express to local speed 1.1
22	G	9X9	81	Local: 6 runs--10 min. apart, Express: 3 runs--20 min. apart; 2 min. transfer; ratio of express to local speed 2.0
23	G	9X9	81	Local: 6 runs--10 min. apart, Express: 3 runs--20 min. apart, 2 min. transfer; ratio of express to local speed 3.0
24	G	9X9	81	Local: 6 runs--10 min. apart, Express: 3 runs--15 min. apart, 2 min. transfer; ratio of express to local speed 2.0
25	G	9X9	81	Local: 6 runs--10 min. apart, Express: 3 runs--25 min. apart, 2 min. transfer; ratio of express to local speed 2.0
26	G	9X9	81	Local: 6 runs--10 min. apart, Express: 3 runs--30 min. apart, 2 min. transfer; ratio of express to local speed 2.0

TABLE 9

INFLUENCE OF TRANSFER TIME

CASE NO.	TRANSFER TIME	COMPUTATION TIME*			AVG. NO. OF TRANSFERS		
		LABCOR	LABSET	TIMEXD	LABCOR	LABSET	TIMEXD
6	2	25	16	35	.68	.68	.72
5	10 (at center node)	29	16	31	.88	.92	.92
7	2	26	17	39	.68	.68	.68
8	10 (at center node)	28	16	34	.76	.76	.88
1	2	143	88	48	1.12	1.28	1.32
15	3	143	95	62	1.08	1.16	1.16
14	5	128	94	62	.84	1.00	.96

*Computation time is in milliseconds

elongated network. The average number of transfers is greater by 4, 31, and 4 percent respectively for LABCOR, LABSET and TIMEXD on the elongated network, although the number of stops in any diagonal trip (across the whole network) increases by 50 percent. Therefore, despite some increase in both computation time and the average number of transfers with an elongated network, the computations in a transit network are not as sensitive to the network shape as is a standard shortest-path calculation.

The sensitivity of computation time and number of transfers to the relative speeds of express and local service was examined, and the results are displayed in Table 10. As express service becomes more attractive (i.e., faster relative to the local service), the number of transfers increases since most trips try to take advantage of the express service, often transferring from local to express and then back to local. The local service effectively operates as a feeder service for the express service when the speed differential is fairly large. Computation time also increases commensurately with the number of transfers. When express service is provided by a fixed rail system, ratios of 3 to 1 in speed may easily hold. The system changeover from express bus to express rail, as for example in Washington, may bring an increase in the average number of transfers per trip and an increase in the computation time per trip itinerary, even with no increase in the number of stops.

Several other characteristics were examined for effects on either computation time or path output and were found to elicit no discernable pattern of response. Among these are the difference between grid-type and radial-type network structure, variations in the initial departure times (with constant frequency), variation of the time between express runs (again with frequency held constant), and various geographical frequency patterns, such as north-south routes frequent while east-west routes are less so, certain streets having frequent service while others have less, or radial runs frequent toward the center but not as frequent outwards. Although it might be expected that any of these parameters might affect algorithm performance, none had a great effect and no discernable pattern emerged.

TABLE 10

INFLUENCE OF RATIO OF EXPRESS
TO LOCAL SPEED

RUN	SPEED RATIO	COMPUTATION TIME*			AVG. NO. OF TRANSFERS		
		LABCOR	LABSET	TIMEXD	LABCOR	LABSET	TIMEXD
21	1.1	56	41	36	1.0	1.28	1.28
22	2.0	64	35	45	1.2	1.32	1.40
23	3.0	71	37	51	1.4	1.52	1.84

*Computation time is in milliseconds.

5. CONCLUSION AND RECOMMENDATIONS

This report has discussed the comparative performance of three algorithms for producing itineraries by computer for use in an automated transit information system. The test networks used in the analysis were designed specifically to highlight performance variation as a function of network size and type and ranges of parameter values. The results of the analysis are summarized below in three categories: path output, program size, and timing.

LABCOR is guaranteed to produce, among all trips arriving at the destination at the same time, that trip having fewest number of transfers. In 9 percent of the test cases LABSET produced a trip with more transfers than did LABCOR. TIMEXD had more transfers in 14 percent of the cases. TIMEXD also produced itineraries in which an extra transfer occurred because of an express overtaking a local vehicle. This situation does not occur with LABCOR. Thus in all cases LABCOR produced significantly more desirable itineraries about 10 percent of the time.

Computer storage requirements for LABCOR and LABSET are similar and depend mainly on storing the route and schedule information, which requires the list of stops on each route and the arrival time at each stop for each departure. Other arrays whose sizes depend primarily on the number of stops are required. The storage required by TIMEXD depends mainly on the number of arcs in the time-expanded network, which is determined by the number of vehicle route segments and the number of possible transfers. The storage for a square 15 by 15 grid with about 300 departures was estimated as 8200 locations for LABCOR, 10,170 for LABSET and 64,440 for TIMEXD. The square grid, which admittedly is a situation least favorable to TIMEXD, nonetheless illustrates the difference between the requirements of the two types of algorithm. Even in a more favorable situation, TIMEXD is likely to require substantially more computer storage than either of the other two algorithms. In addition, our programming of LABCOR and LABSET has not taken full advantage of most efficient storage practices, whereas TIMEXD has much less leeway.

TIMEXD is clearly fastest for larger networks, and as shown by the regression equation, its calculation time grows more slowly with an increase in network size. LABCOR, which produces better paths, averaged 77 percent longer computation times and for the larger networks included in the analysis, the 15 by 15 grids, was 2.5 times as slow as TIMEXD. The timing of each of the three algorithms was highly correlated with the number of nodes, with the rate of growth being greatest for LABCOR, moderate for LABSET and lowest for TIMEXD. Timing also depends on network characteristics, such as the number of transfers, the minimum transfer time requirements, the relative speeds of express and local service, and network shape.

A recommendation on the choice among the three algorithms depends in part on the appropriate tradeoff between speed and quality of the output itinerary. If the speed of the LABCOR algorithm is sufficient for the particular application, its clearly-more-desirable itinerary output makes it the algorithm of choice, but if speed is more important then TIMEXD becomes attractive. Our analysis did not attempt to fine-tune the networks, for instance, to examine individually the transfer arcs included in TIMEXD networks to see if undesirable transfers could be curtailed by removing some of these arcs as spurious candidates. Other heuristics or data manipulations are also possible and might improve the path output from TIMEXD without degrading its performance significantly. TIMEXD also has the disadvantage of requiring a large amount of core storage; both of the other algorithms require much less. LABSET produces paths which may be longer than those output by LABCOR, although this does not happen as frequently as with TIMEXD. LABSET is faster than LABCOR but not as fast as TIMEXD, and requires only slightly more storage than LABCOR. Thus in situations where speed is important but better quality path output than available from TIMEXD is desired, LABSET may be an acceptable compromise.

6. REFERENCES

1. Judith F. Gilsinn, Patsy B. Saunders, and Martin H. Pearl, Path Finding Algorithms and Data Structures for Point-to-Point Trip Management, National Bureau of Standards Report Number NBSIR 75-676, January 1975.

2. Judith Gilsinn and Christoph Witzgall, A Performance Comparison of Labeling Algorithms for Calculating Shortest Path Trees, National Bureau of Standards Technical Note 772, May 1973.

3. Douglas R. Shier and Judith F. Gilsinn, Cost/Benefit Analysis of Automated Transit Information Systems, National Bureau of Standards Report Number NBSIR 77-1253, June 1977.

APPENDIX A

DOCUMENTATION OF PROGRAMS FOR TEST-PROBLEM GENERATION AND ITINERARY-FINDING

The computer programs used in performing the analyses described in Section 4 are documented in this appendix. A flowchart of these programs appears in Figure A.1, with the programs shown in rectangular boxes, input files in rectangular boxes with a cut-off corner, intermediate data files in boxes with rounded sides, and output in the box labeled "TRIP ITINERARIES". The numbers in the upper right corners of program boxes are keyed to the section in which the program is documented. For instance, documentation of program RAD is in section A.1. The number in parentheses in either an input or an intermediate file box refers to the logical unit number used in referencing the file. Descriptions of the contents and formats of the files appear in Section A.10. Two programs not appearing in this chart have been included in the documentation, because they would be important parts of an actual information system's computer-program package and because they have been discussed both in [1] and in the text of this report. They are an arrival-oriented version of the time-expanded network algorithm, described in Section 2.2.2, and a program to remove extraneous, non-decision nodes from a transit network. Computer listings of all programs appear in Appendix B.

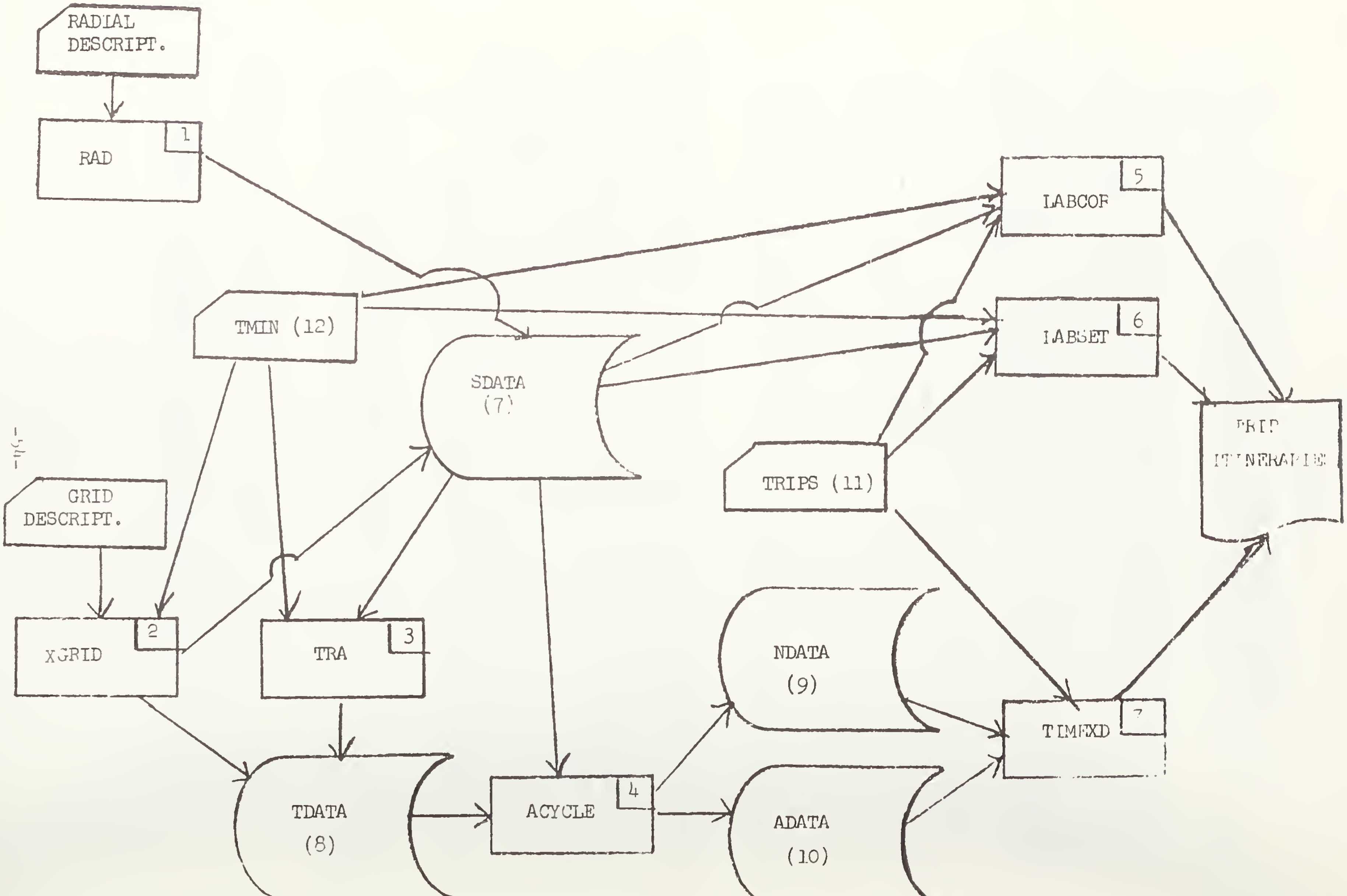
All programs are written in FORTRAN V, UNIVAC's enhanced version of FORTRAN IV, and were run on the UNIVAC 1108 at NBS under the EXEC 8 operating system. Generally, all programs for a single test case were executed in sequence in the same computer run with network generation followed by necessary preliminary processing, followed immediately by itinerary computation by each of the three algorithms in sequence. Thus for radial networks, the program execution sequence is RAD, TRA, ACYCLE, LABCOR, LABSET and TIMEXD; for grid networks the sequence is similar but omits TRA, thus becoming XGRID, ACYCLE, LABCOR, LABSET, and TIMEXD. Input files TMIN and TRIPS, together with the input required for either the grid or radial network generators, must be prepared in advance. All other files are generated by the programs as they are executed.

Since these programs were coded primarily for use in the analyses described in this report, no special effort was made to limit code to a portable subset of FORTRAN. Program characteristics which affect portability are listed below:

1. Card input is from logical unit 5, printer output is on logical unit 6, and units 7 through 12 are used for various data files.

2. In the path computation programs, a variable INF, representing a very large integer, is set equal to 999999999, which may be too large for machines with smaller word sizes.

FIGURE A.1
FLOWCHART OF THE TRANSIT PROGRAM SYSTEM



3. We have used the "PARAMETER =" statement to set program dimensions. This may be replaced by explicit numerical values wherever the PARAMETER variable occurs.

4. The UNIVAC FORTRAN V compiler ignores all remaining characters following the character @ on any line. This has been used to add short comments to lines of code, and all characters after and including the @ on any line.

5. The IMPLICIT INTEGER (A-Z) statement makes all variables of integer type, and may be replaced by a list of all variables appearing in the code.

6. The "END=" clause in a READ statement transfers control to the statement number given when an end-of-file condition is sensed in input.

7. The path-finding algorithm codes all call the special system subroutine CPUSUP which computes the CPU time in milliseconds since the start of the run. Any equivalent clock routine can be used.

Detailed documentation and user instructions for the programs shown in Figure A.1, together with the two additional codes discussed above, are included in the sections below. In each case there is a narrative describing the program's function, a list of the variables and arrays appearing in the code, a list of the input required and outputs produced together with their formats, and descriptions of subroutines. Listings of the codes appear in Appendix B.

A.1. PROGRAM RAD

This program generates a user-specified radial, or "spider web", network in which routes go outward from a central node along radial segments or go between radials in circular arcs or portions of such arcs. The user specifies the number of radials, the number of stops and distances between stops along each radial, the stops on radials connected by circular segments (or "beltways"), and the stops at which "spike" routes connect to radials. These latter are routes which proceed from the central node out along a radial but diverge from the radial at some node along it.

By user specification, any radial, beltway or spike can also have an express route. Each route, local or express, has a complementary route which traverses the same set of stops in reverse order. Express routes stop only at intersections with other express routes.

Schedules for each route are computed from input data which include the number of runs, the headway between runs, and the departure time of the first run for each route for each time period. Time between stops is computed from interstop distance by using a user-supplied conversion factor. Both local and express conversion factors are input for each time period.

The main program RAD serves both as a network-description input routine and as a calling routine to generate the radial transit route system. Subroutine LOCAL is called to create the local network and subroutine EXPRES is called to create the express routes. Subroutine RSCHED is called from LOCAL and EXPRES to compute schedules for each route.

A.1.1 Variables and Arrays Used in RAD

Time period input

IPD	-	number of time periods used in computing schedules. (Headways are constant during a time period.)
ZK(I), I=1, IPD	-	factor used in converting distance to time for local routes for each time period
XZK(I), I=1, IPD	-	factor used in converting distance to time for express routes for each time period

Radial input

- IRAD - number of radials
- ANGLE(r) - angle of radial r, measured counterclockwise from east
- ISTP(r) - number of stops on radial r not counting the center node
- DSTRAD(r,j) - distance between stops j and j + 1 on radial r
- REXP(r) - a flag set to 1 if radial r is an express route

Beltway input

- IBELT - number of beltways
- IBIN(b) - initial (east-most) radial connected by beltway b
- IBFIN(b) - final radial connected by beltway b
- NB(b) - number of radials intersected by beltway b. (Note that beltway b will intersect all radials between IBIN(b) and IBFIN(b). If IBFIN(b) $\leq$ IBIN(b) it will intersect radials IBIN(b) to IRAD and 1 to IBFIN(b). When IBIN(b) = IBFIN(b) the beltway is a full beltway encircling the center node; otherwise it is only a "partial circle".)
- IBSTP(b,i) i = 1, NB(b) - stop on radial IBIN(b) + i - 1 at beltway b, where the center is counted as stop 1 on each radial. (Note that the length of the beltway segment, between the ith and (i + 1)st radials it intersects, is calculated as the average of the circular arc length at the radius determined by stop IBSTP(b,i) on the ith radial and the circular arc length at the radius determined by stop IBSTP(b,i+1) on the (i+1) radial.)
- BEXP(b) - a flag set to 1 if beltway b is an express route

Spike input

- ISPIKE - number of spikes

ISRAD(s) - radial to which spike is connected

ISSTP(s) - stop on radial ISRAD to which spike is connected

DSPIKE(s) - length of spike

SEXP(s) - a flag set to 1 if spike s is an express spike

Node description

KRAD(k) - radial on which node k is located

KSTP(k) - stop on radial KRAD(k) which is node k

NODE(r,j) - node for stop j on radial r

Route description

MRTE(m,n) - nth stop on route m. (Note that MRTE(1,j) = MRTE(2,n - j + 1) since routes go in both directions and appear in pairs.)

DTEMP(m,n) - distance between stops n and n + 1 along route m. (As noted for MRTE, the distances in DTEMP(1,k) appear in the reverse order to DTEMP(2,k).)

Input schedule description

JRUNS - number of runs of this route this time period

JHEAD - headway between runs of this route this time period

PDTM - time of first run of this route this time period

Counters

K - the number of nodes

M1 - current route in one direction

M2 - route in the opposite direction to M1 traversing the same stops in reverse order

A.1.2 Program Input

Input to program RAD includes a description of the radial network to be generated and schedule data for each route. The input, read from cards on logical unit 5, consists of three main sections: (1) the topological and route structure of the radials, beltways and spikes, (2) the schedule information for local routes and (3) the schedule information for express routes. The actual card formats are given below in the order in which they are to appear.

Time period input

Contents	Format
IPD	I5
ZK(I),I = 1, IPD	8F10.1
XZK(I),I = 1, IPD	8F10.1

Radial input

IRAD	I5

For each radial:

REXP	I5
ANGLE, ISTP	F10.1, I5
(DSTRAD(radial, J), J=1, ISTP)	8F10.1

Beltway input

IBELT	I5

For each beltway:

BEXP	I5
IBIN, IBFIN	2I5
(IBSTP (beltway, J), J=1, NB)	16I5

Spike input

ISPIKE	I5

For each spike:

SEXP	I5
ISRAD, ISSTP, DSPIKE	2I5, F10.1

A description of the schedule input data follows. Within each group of schedule data, the order must be the same as the order of the topology input above.

Local radial schedule input

For each radial:

Route going outward from center, one card for each time period
JRUNS, JHEAD, JDTM 3I5

Route going inward toward center, one card for each time period
JRUNS, JHEAD, JDTM 3I5

Local beltway schedule input

For each beltway:

Route going counterclockwise, one card for each time period
JRUNS, JHEAD, JDTM 3I5

Route going clockwise, one card for each time period
JRUNS, JHEAD, JDTM 3I5

Local spike schedule input

For each spike:

Route going outward from center, one card for each time period
JRUNS, JHEAD, JDTM 3I5

Route going inward toward center, one card for each time period
JRUNS, JHEAD, JDTM 3I5

Express beltway schedule input

For each express beltway:

Route going counterclockwise, one card for each time period
JRUNS, JHEAD, JDTM 3I5

Route going clockwise, one card for each time period
JRUNS, JHEAD, JDTM 3I5

Express spike schedule input

For each express spike:

Route going outward from center, one card for each time period
JRUNS, JHEAD, JDTM 3I5

Route going inward toward center, one card for each time period
JRUNS, JHEAD, JDTM 3I5

Express radial schedule input

For each express radial:

Route going outward from center, one card for each time period
JRUNS, JHEAD, JDTM 3I5

Route going inward toward center, one card for each time period
JRUNS, JHEAD, JDTM 3I5

A.1.3 Program Output

Output consists of the file SDATA (on unit 7) which gives for each route the number of stops, the number of runs, the list of stops and the times at each stop for each run. This information is also printed.

A.1.4 Subroutine LOCAL

Subroutine LOCAL generates the nodes and local routes of a radial transit route system. The network topology description is available through three common blocks - RADIAL, BELTWY and SPIKE. The subroutine constructs the node description arrays KRAD, KSTP and NODE as it computes the local radial routes. Local beltway and spike routes are then computed. After each route and its complement are computed, subroutine RSCHED is called to compute the schedules.

A.1.5 Subroutine EXPRES

This subroutine generates the express routes of a radial transit route system. Two-way express routes are computed for radials, beltways and spikes designated by the user to be major routes. Subroutine RSCHED is called to compute the schedule of each route and its complement.

A.1.6 Variables and Arrays Used in LOCAL and EXPRES

Radial description

IRAD - number of radials

ANGLE(r) - angle of radial r, measured counterclockwise from east

ISTP(r) - number of stops on radial r not counting the center node

DSTRAD(r,j) - distance between stops j and j + 1 on radial r

REXP(r) - a flag set to 1 if radial r is an express route

Beltway description

IBELT	-	number of beltways
IBIN(b)	-	initial radial connected by beltway b
IBFIN(b)	-	final radial connected by beltway b
DSTBLT(b,j)	-	distance between stops j and j + 1 on beltway b
NB(b)	-	number of radials intersected by beltway b
IBSTP(b,i)	-	stop on radial IBIN(b) + i - 1 on beltway b, where the center is counted as stop 1
BEXP(b)	-	a flag set to 1 if beltway b is an express route

Spike description

ISPIKE	-	number of spikes
ISRAD(s)	-	radial to which spike is connected
ISSTP(s)	-	stop on radial ISRAD to which spike is connected
DSPIKE(s)	-	length of spike
NDSPK(s)	-	node number of stop at end of spike
SEXP(s)	-	a flag set to 1 if spike s is an express route

Node description

KRAD(k)	-	radial on which node k is located
KSTP(k)	-	stop on radial KRAD(k) which is node k
NODE(r,j)	-	node for stop j on radial r
KEXP(k)	-	true/false express node indicator

Route description

MRTE(m,n)	-	nth stop on route. (Note that MRTE (1,j) = MRTE (2, n-j+1) since routes go in both directions and appear in pairs.)

DTEMP(m,n) - distance between stops n and n+1 along route. (As noted for MRTE, the distances in DTEMP (1,k) appear in the reverse order to DTEMP (2,k).)

Counters

K - the number of nodes

M1 - current route in one direction

M2 - route in the opposite direction to M1 traversing the same stops in reverse order

A.1.7 Subroutine RSCHED

This subroutine computes and outputs schedule information for routes M1 and M2. Input, which is transmitted through calling arguments and the common block GENRAL, includes the stops along routes M1 and M2 stored in MRTE, the distance between stops in DTEMP, the number of stops along these routes in N, and time period information stored in IPD, ZK and XZK. After schedules for both routes are computed, RSCHED outputs the route number, the number of stops, the number of runs, the list of stops and the times at each stop for each run both to file SDATA and to the printer.

A.1.8 Variables and Arrays Used in RSCHED

N - number of stops along either of the current pair of routes

MRTE(m,j) - jth stop along the route (Stops in MRTE(1,-) appear in the reverse order to those in MRTE(2,-).)

DTEMP(m,j) - distance between stops MRTE(m,j) and MRTE(m,j + 1)

M1 - number of the first of the pair of routes being considered

IPD - number of time periods for which schedules are to be constructed

ZK(i) - factor for converting distance to time for local routes in period i

XZK(i) - factor for converting distance to time for express routes in period i

JRUNS(i)	-	number of runs of the current route in period i
JHEAD(i)	-	headway between runs of the current route in period i
JDTM(i)	-	time of first run of the current route in period i
NRUNS	-	number of runs of the current route over all periods
MM	-	keeps track of which of the pair of routes is the current route
JTIME(ℓ)	-	the time that the current vehicle stops at stop MRTE(MM,ℓ)
LOE	-	Equal to 1 if RSCHED is being called from LOCAL. Equal to 2 if RSCHED is being called from EXPRES

A.2. PROGRAM XGRID

This program generates a P x Q grid transit network with routes running in the horizontal (west-east, east-west) and the vertical (north-south, south-north) directions. Stops are numbered consecutively from left to right and from top to bottom. Routes are numbered consecutively in the order: W-E, E-W, N-S and S-N. Express horizontal and vertical routes are numbered in a similar order beginning with route number 2P + 2Q + 1. Any stop is allowed to be a transfer point between routes which stop at that point. The minimum transfer time between any two routes is considered to depend only on the stop at which the transfer occurs. Routes are assumed to follow regular schedules (constant headways) during each of a number of time periods. A different conversion factor may apply for converting distance into travel time during different periods, and different factors apply to local and to express routes.

The program XGRID makes calls to four subroutines: GRID (which creates the local grid network), XPRESS (which creates express routes), XSCHED (which produces the complete schedule information) and TRANS (which outputs the appropriate transfer information). These subroutines are described in fuller detail below. Input to XGRID includes the number of stops P and the number of stops Q which define, respectively, the vertical and horizontal dimensions of the P x Q grid. In addition, the user must specify the interstop distances between successive "rows" and "columns" of the grid, as well as the number of time periods and conversion factors for each period for local and for express routes. Subroutine XPRESS requires designation of the horizontal and vertical elements to be used for express routes, subroutine XSCHED requires the input of abbreviated schedule information and subroutine TRANS requires the input of (minimum) transfer times at each stop. Output of the grid generator consists of detailed schedule information and transfer information.

A.2.1 Variables and Arrays Used in XGRID, GRID, and XPRESS

Input parameters

P	-	vertical dimension of grid
Q	-	horizontal dimension of grid
L(i)	-	distance between rows i and i - 1
W(i)	-	distance between columns i and i - 1
PDS	-	number of periods
ZK(j)	-	converts distance into time for local routes for period j
XZK(j)	-	converts distance into time for express routes for period j

Route description

NN(r)	-	number of nodes (stops) on route r
NODE(r,i)	-	the ith node on route r
D(r,i)	-	the ith interstop distance along route r
RBASE	-	convenient reference base for absolute route numbers

Transfer node description

TNODE(j)	-	the jth transfer node
RT1(j)	-	route from which a transfer is made at TNODE(j)
RT2(j)	-	route to which a transfer is made at TNODE(j)
NTRANS	-	number of transfers

A.2.2 Program Input

Input to the grid generator consists of five types:

1. Structural parameters of the local grid, read in from cards (unit 5) by XGRID.

2. Structural parameters of the express routes, read in from cards (unit 5) by XPRESS.

3. Conversion factors for each period, read in from cards (unit 5) by PGRID.

4. Abbreviated schedule information, read in from cards (unit 5) by TSCHED.

5. Minimum transfer times at each node, found on file TMIN (unit 12) and read by TRANS.

Specific formats for data types 1, 2, 3 and 4 are given below.

Local Structural Parameters

Contents	Format
P,Q,(L(I),I = 2,P), (W(I),I = 2,Q)	(16I5)

Express Structural Parameters

NMP,NMQ	2I5
(MAINP(I), I=1,NMP)	(16I5)
(MAINQ(J), J=1,NMQ)	(16I5)

Conversion Factors

PDS,(ZK(I), I=1,PDS)	I5, (15F5.2)
(XZK(I),I=1,PDS)	(5X,15F5.2)

Abbreviated Schedule Information

NR,(ROUTE(J), J=1,NR) -- one card for each group of routes, followed by --	(16I5)
RUNS(I), HEAD(I), (DTIME(I,J) J=1, NR) -- one card per group for each time period -- (applies to all routes in the group)	(16I5)

@EOF

A.2.3 <u>Program Output</u>

Output from the grid generator consists of two files:

1. Detailed schedule information is produced by XSCHED and is written onto file SDATA (unit 7).

2. Transfer information is produced by TRANS and is written onto file TDATA (unit 8).

A.2.4 <u>Subroutine GRID</u>

This subroutine generates the nodes, local routes and transfer data for a P x Q grid network, where P,Q > 1. The variables P, Q, RBASE and the arrays LL, W are transmitted from PGRID. The subroutine defines the variable NTRANS and constructs the arrays NN, NODE, D, TNODE, RT1 and RT2; these quantities are then made available to other (sub) programs through COMMON.

A.2.5 <u>Variables and Arrays used in GRID</u>

Input parameters

P	-	vertical dimension of grid
Q	-	horizontal dimension of grid
LL(i)	-	distance between rows i and i - 1
W(i)	-	distance between columns i and i - 1
RBASE	-	convenient reference base for absolute route numbers

Route description

NN(r)	-	number of nodes on route r
NODE(r,i)	-	the ith node on route r
D(r,i)	-	the ith interstop distance along route r

Transfer node description

TNODE(j)	-	the jth transfer node
RT1(j)	-	route from which a transfer is made at TNODE(j)
RT2(j)	-	route to which a transfer is made at TNODE(j)

NTRANS	-	number of transfers

A.2.6 Subroutine XPRESS

This subroutine generates the express routes for a PxQ grid. The variables P, Q, RBASE and the arrays LL and W are transmitted from XGRID. The subroutine increments the variable NTRANS and constructs the arrays NN, NODE, D, TNODE, RT1 and RT2 for the express routes; these quantities are then made available to other (sub) programs through COMMON. XPRESS reads from cards (unit 5) NMP and NMQ and the arrays MAINP and MAINQ identifying those vertical and horizontal elements used by express routes. XPRESS calls subroutine XTRANS to construct transfers.

A.2.7 Variables and Arrays used in XPRESS

Input parameters

P	-	vertical dimension of grid
Q	-	horizontal dimension of grid
LL(i)	-	distance between rows i and i - 1
W(i)	-	distance between columns i and i - 1
RBASE	-	convenient reference base for absolute route numbers

Route description

NN(r)	-	number of nodes on route r
NODE(r,i)	-	the ith node on route r
D(r,i)	-	the ith interstop distance along route r

Transfer node description

TNODE(j)	-	the jth transfer node
RT1(j)	-	route from which a transfer is made at TNODE(j)
RT(j)	-	route to which a transfer is made at TNODE(j)
NTRANS	-	number of transfers

Express route input

NMP	-	number of horizontal express routes
NMQ	-	number of vertical express routes
MAINP(i)	-	ith main horizontal element (express routes travel along the main elements)
MAINQ(i)	-	ith main vertical element

Express route descriptors

SP(i)	-	ith main horizontal element, including endpoints if they are not in MAINP
SQ(i)	-	ith main vertical element, including endpoints if they are not in MAINQ
NSP	-	number of entries in SP
NSQ	-	number of entries in SQ
DP(i)	-	distance between ith and i + 1st elements in SP
DQ(i)	-	distance between ith and i + 1st elements in SQ
FLAG(i)	-	an array designating which transfers are allowed between the current route r and other routes stopping at the same node. FLAG = 0 indicated no transfer; FLAG = 1 indicates transfer is allowed. The transfers controlled by FLAG are as follows for each value of i: i=1: r to local West East route i=2: local West East route to r i=3: r to local East West route i=4: local East West route to r i=5: r to local North South route i=6: local North South route to r i=7: r to local South North route i=8: local South North route to r i=9: r to express East West route i=10: r to express West East route i=11: r to express North South route i=12: r to express South North route

A.2.8 Subroutine XTRANS

This subroutine creates the express route transfers indicated in the array FLAG. The subroutine increments the variable NTRANS and constructs the COMMON arrays TNODE, RT1 and RT2 for express routes at nodes at which vertical and horizontal express routes intersect. (Analogous arrays for nodes at the ends of express routes, along the periphery of the grid, are constructed in XPRESS.)

A.2.9 Variables and Arrays Used in XTRANS

Input parameters

P	-	vertical dimension of grid
Q	-	horizontal dimension of grid
FLAG(i)	-	array designating which transfers are allowed between the current route r and other routes stopping at the same node. FLAG = 0 indicates no transfer; FLAG = 1 indicates transfer is allowed. The transfers controlled by FLAG are as follows for each value of i: i=1: r to local East West route i=2: local East West route to r i=3: r to local West East route i=4: local West East route to r i=5: r to local North South route i=6: local North South route to r i=7: r to local South North route i=8: local South North route to r i=9: r to express West East route i=10: r to express East West route i=11: r to express North South route i=12: r to express South North route
R	-	current express route for which transfers are being calculated
I	-	local grid horizontal element of current node
J	-	local grid vertical element of current node
L1	-	express horizontal element of current node
L2	-	express vertical element of current node
NMP	-	number of horizontal express routes
NMQ	-	number of vertical express routes

A.2.10 Subroutine XSCHED

This subroutine reads in (from unit 5) a group of routes together with abbreviated schedule information and then produces detailed schedule information for each route and period. The detailed schedule information is written out onto file SDATA (unit 7). The variables PDS, RBASE, PQ and the arrays NN, NODE, D, ZK, XZK are transmitted from XGRID.

A.2.11 Variables and Arrays Used in XSCHED

Input variables and arrays

PDS	-	number of periods
RBASE	-	convenient reference base for absolute route numbers
PQ	-	the number of local routes (2*P+2*Q)
ZK(j)	-	converts distance into time for period j for local routes
XZK(j)	-	converts distance into time for period j for express routes
NN(r)	-	number of nodes on route r
NODE(r,i)	-	the ith node on route r
D(r,i)	-	the ith interstop distance along route r

Additional variables and arrays (read from unit 5)

NR	-	number of routes in a group
ROUTE(j)	-	the jth route of the group
RUNS(i)	-	number of runs for period i
HEAD(i)	-	headway for routes in period i
DTIME(i,j)	-	initial departure time for route j in period i

Working arrays

SCHED(k)	-	schedule time for kth node along route

A.2.12 Subroutine TRANS

This subroutine writes out onto file TDATA (unit 8) the transfer information previously generated. The transfer time array TMIN is read from unit IN = 12. The variables PQ, RBASE, NTRANS and the arrays TNODE, RT1, RT2 are transmitted from XGRID.

A.2.13 Variables and Arrays Used in TRANS

Input variables and arrays

PQ - number of nodes in grid network

RBASE - convenient reference base for absolute route numbers

NTRANS - number of transfers

TNODE(j) - the jth transfer node

RT1(j) - route from which a transfer is made at TNODE(j)

RT2(j) - route to which a transfer is made at TNODE(j)

TMIN(i) - minimum transfer time between any two routes at node i

A.3. PROGRAM TRA

This program produces a list of allowable transfers between routes from the route descriptions and minimum transfer times for each node. It is assumed that routes occur in pairs, with routes i and i + 1 having the same stops in reverse order. Transfers are allowed between all routes stopping at a node except that:

1. One cannot transfer from the first stop on a route,

2. One cannot transfer to the last stop on a route, and

3. One cannot transfer from a route to its reverse counterpart.

A.3.1 Variables and Arrays Used in TRA

MINTRA(i) - minimum time required to transfer between routes at node i

STP(k)	-	the kth stop along the current route (used in reading the route information)
NSTP(i)	-	the number of routes stopping at node i
RTE(i,j)	-	the jth route stopping at node i
BE(i,j)	-	1 if node i is the first node on route RTE(i,j). 2 if node i is the last node on route RTE(i,j). 0 otherwise.
NODE	-	the number of nodes
M	-	the number of stops on the current route
N	-	the number of routes stopping at the current node

A.3.2 Program Input

Input to TRA consists of one card and two files:

1. Card with number of nodes (I5 format).
2. Schedules from file SDATA on unit 7.
3. Minimum transfer times in file TMIN on unit 12.

A.3.3 Program Output

Program output is the file TDATA which gives, for each node, the route pairs between which transfers are allowed and the minimum time required to transfer between those routes at that node.

A.4. PROGRAM ACYCLE

This program produces an appropriate time-expanded network from given schedule information and transfer data. Each node of the time-expanded network that is constructed represents a particular (stop, time) pair. Transfers are accommodated by using transfer arcs between nodes; such transfer arcs are labelled with the fictitious route number 9999. Output consists of the time-expanded node and arc data. Nodes are sorted by their time component, while arcs are sorted by their origin node. This program makes use of the subroutine SORTP having arguments X, N, Y, XPOS. That routine sorts the N elements of the array X into nondecreasing order, thus forming array Y. The ith element of array XPOS indicates what position of the original array X corresponds to the ith ordered observation Y(i).

A.4.1 Variables and Arrays Used in ACYCLE

Input variables and arrays

RT	-	route number
NN	-	number of stops on route
RUNS	-	number of runs
NODE(i)	-	the ith stop along the route
SCHED(i)	-	the ith schedule time along the route
TNODE	-	stop at which transfer occurs
RT1	-	route from which transfer at TNODE
RT2	-	route to which transfer at TNODE
TMIN	-	minimum transfer time at TNODE

Constructed arrays

N(i)	-	stop associated with network node i
T(i)	-	time associated with network node i
START(r)	-	first position where information may be found for route r on node list
END(r)	-	last position where information may be found for route r on node list
LLEN	-	length of network node list
FROM(j)	-	starting node of arc in position j of arc list
TO(j)	-	ending node of arc in position j of arc list
RTE(j)	-	route number corresponding to arc in position j of arc list
MLEN	-	length of network arc list

Working arrays

TT(i)	-	the ith ordered element of T

TIND(i) - the position in T of the ith element of TT

NEW(i) - the position in TT of the ith element of T

FF(i) - the ith ordered element of FROM

FIND(i) - the position in FROM of the ith element of FF

A.4.2 Program Input

Input to ACYCLE consists of the following two files:

1. SDATA (unit 7) contains detailed schedule information for each route.

2. TDATA (unit 8) contains transfer information for each transfer point and routes connecting there.

A.4.3 Program Output

Output from ACYCLE consists of the following two files:

1. NDATA (unit 9) contains the node data for the time-expanded network, sorted by time.

2. ADATA (unit 10) contains the arc data for the time-expanded network, sorted by origin node.

A.5. PROGRAM LABCOR

This program calculates trip itineraries using the label-correcting bipartite route/stop scheme described in Section 1.1. Program input consists of the route and schedule data, minimum transfer times, and a list of trips to be calculated. Output is the itinerary for each trip, the calculation time in milliseconds for each trip, and the average and standard deviation of the calculation times for all trips.

A.5.1 Variables and Arrays used in LABCOR

Stop information

NR(s) - number of routes stopping at s

ROUTE(s,j) - jth route stopping at s

MINTRA(s) - minimum time required to transfer between routes at s

Route information

NS(r)	-	number of stops on route r
STOP(r,i)	-	ith stop on route r
SCHED(k,i)	-	arrival time at the ith stop of the kth departure
SBEG(r)	-	location in SCHED of the first scheduled departure for route r
SEND(r)	-	location in SCHED of the last scheduled departure for route r

Arrays used in the algorithm

L(s)	-	sequence list of stops to fan out from
F(s)	-	position of stop s in list L
T(s)	-	arrival time at stop s
TB(s)	-	boarding time for vehicle arriving at s at T(s)
PS(s)	-	stop preceding s in the path to s
PR(s)	-	route from PS(s) to s

Arrays used in printing the path

SPRT(j)	-	stop
RPRT(j)	-	route
TPRT(j)	-	arrival time
TBPRT(j)	-	boarding time

Variables used in timing calculations

RUNTIM	-	CPU time (in milliseconds) used in calculating one trip
RTIME	-	sum of RUNTIM's
RTSQ	-	sum of squares of RUNTIM's

NRUN	-	number of trips calculated
ROUT	-	used in printing average and standard deviations

Variables describing the trip to be calculated

ORG	-	trip origin stop
DST	-	trip destination stop
TIME	-	desired departure time

A.5.2 Program Input

Input to LABCOR consists of three files:

1. Schedules from file SDATA on unit 7.
2. Minimum transfer times in file TMIN on unit 12.
3. Trips to be found in file TRIPS on unit 11.

A.5.3 Program Output

All program output is on the printer and consists, for each desired trip, of the origin, destination and departure time, the trip itself with route, boarding and alighting stops, and times for each segment of the trip and computation time. At the end of the run the average computation time for all trips and its standard deviation are also printed.

A.6. PROGRAM LABSET

This program calculates trip itineraries using the label-setting bipartite route/stop scheme described in Section 2.2. Program input consists of the route and schedule data, minimum transfer times, and a list of trips to be calculated. Output is the itinerary for each trip, the calculation time in milliseconds for each trip, and the average and standard deviation of the calculation times for all trips.

A.6.1 Variables and Arrays used in LABSET

Stop information

NR(s)	-	number of routes stopping at s
ROUTE(s,j)	-	jth route stopping at s
MINTRA(s)	-	minimum time required to transfer between routes at s

Route information

NS(r)	-	number of stops on route r
STOP(r,i)	-	ith stop on route r
SCHED(k,i)	-	arrival time at the ith stop of the kth departure
SBEG(r)	-	location in SCHED of the first scheduled departure for route r
SEND(r)	-	location in SCHED of the last scheduled departure for route r

Arrays used in the algorithm

L(s)	-	sequence list of stops to fan out from
LPRED(s)	-	predecessor node to node s in chain of nodes representing a level in sequence list L. If s heads the chain (i.e. HEAD(s) is true), this pointer gives the position of s in the chain of nodes.
LSUCC(s)	-	successor node to node s in chain of nodes representing a level in list L
HEAD(s)	-	logical variable used to indicate whether s heads a chain in L
T(s)	-	arrival time at stop s
TB(s)	-	boarding time for vehicle arriving at s at T(s)
PS(s)	-	stop preceding s in the path to s
PR(s)	-	route from PS(s) to s

Arrays used in printing the path

SPRT(j)	-	stop
RPRT(j)	-	route
TPRT(j)	-	arrival time
TBPRT(j)	-	boarding time

Variables used in timing calculations

RUNTIM	-	CPU time (in milliseconds) used in calculating one trip
RTIME	-	sum of RUNTIM's
RTSQ	-	sum of squares of RUNTIM's
NRUN	-	number of trips calculated
ROUT	-	used in printing average and standard deviations

Variables describing the trip to be calculated

ORG	-	trip origin stop
DST	-	trip destination stop
TIME	-	desired departure time

A.6.2 Program Input

Input to LABCOR consists of three files:

1. Schedules from file SDATA on unit 7.
2. Minimum transfer times in file TMIN on unit 12.
3. Trips to be found in file TRIPS on unit 11.

A.6.3 Program Output

All program output is on the printer and consists, for each desired trip, of the origin, destination and departure time, the trip itself with route, boarding and alighting stops, and times for each segment of the trip and computation time. At the end of the run the average computation time for all trips and its standard deviation are also printed.

A.7. PROGRAM TIMEXD

This program uses a time-expanded representation (see Section 2.2) in order to calculate a "best" itinerary from a given origin to a given destination. The trip produced must not depart the origin before some specified time and must arrive at the destination as early as possible. Program input consists of the node and arc data for the time-expanded network together with a list of trips for which itineraries are required. Schedule times are assumed to be given for 1 through 1600 minutes. Program output consists of an itinerary for each trip, the calculation time (in milliseconds) for each trip as well as the average and standard deviation of calculation times for all trips in the list.

A.7.1 Variables and Arrays used in TIMEXD

Node and arc data

N(i)	-	stop associated with network node i
T(i)	-	time associated with network node i
ARC(i)	-	last position where information may be found for node i in arc list
TO(j)	-	ending node of arc in position j of arc list
RTE(j)	-	route number corresponding to arc in position j of arc list
NN(t)	-	node corresponding to the first occurrence of time t or later
NODE	-	number of network nodes

Input variables for trip

ORG	-	desired origin stop of trip
DST	-	desired destination stop of trip
TIME	-	time at or after which trip is to begin

Variables and arrays used in the algorithm

DONE	-	first node for which the destination stop has been encountered
P(i)	-	predecessor node to node i along the current path
PRTE(i)	-	route into node i along the current path

Arrays used in printing the path

PATHN(k)	-	stop in position k along path
PATHT(k)	-	time in position k along path

PATHR(k) - route in position k along path

Variables used in timing calculations

DIFF	-	CPU time (in milliseconds) used in calculating one trip
RTIME	-	cumulative sum of DIFF's
RTSQ	-	cumulative sum of squares of DIFF's
NRUN	-	number of trips calculated
ROUT	-	used in printing average and standard deviation of trip calculation times

A.7.2 Program Input

Input to TIMEXD consists of three files:

1. Node data in time-expanded form, sorted in increasing order by time and found on file NDATA (unit 9).

2. Arc data in time-expanded form, sorted by origin node and found on file ADATA (unit 10).

3. Trips to be found, on file TRIPS (unit 11).

A.7.3 Program Output

All program output is produced on the line printer and consists of the following information:

1. The origin, destination and departure time for each trip.

2. The trip itinerary with route, boarding stop, boarding time, alighting stop and alighting time for each segment of the trip.

3. Computation time for the trip calculation.

4. Mean and standard deviation of computation times for all trips.

A.8. PROGRAM TIMEXA

This program uses a time-expanded representation (see Section 2.2) in order to calculate a "best" itinerary from a given origin to a given destination. The trip produced must arrive at the destination by a specified time and must depart from the origin as late as possible. Program input consists of the node and arc data for the time-expanded network together with a list of trips for which itineraries are required. Schedule times are assumed to be given for 1 through 1600 minutes. Program output consists of an itinerary for each trip, the calculation time (in milliseconds) for each trip as well as the average and standard deviation of calculation times for all trips in the list.

A.8.1 Variables and Arrays used in TIMEXA

Node and arc data

- N(i) - stop associated with network node i
- T(i) - time associated with network node i
- ARC(i) - last position where information may be found for node i in arc list
- TO(j) - ending node of arc in position j or arc list
- RTE(j) - route number corresponding to arc in position j of arc list
- NN(t) - node corresponding to last occurrence of time t or earlier
- NODE - number of network nodes

Input variables for trip

- ORG - desired origin stop of trip
- DST - desired destination stop of trip
- TIME - time by which trip is to be completed

Variables and arrays used in the algorithm

- S(i) - successor node to node i along the current path
- SRTE(i) - route out of node i along the current path

Arrays used in printing the path

- PATHN(k) - stop in position k along path
- PATHT(k) - time in position k along path

PATHR(k) - route in position k along path

Variables used in timing calculations

DIFF - CPU time (in milliseconds) used in calculating one trip

RTIME - cumulative sum of DIFF's

RTSQ - cumulative sum of squares of DIFF's

NRUN - number of trips calculated

ROUT - used in printing average and standard deviation of trip calculation times

A.8.2 Program Input

Input to TIMEXA consists of three files:

1. Node data in time-expanded form, sorted in increasing order by time and found on file NDATA (unit 9).
2. Arc data in time-expanded form, sorted by origin node and found on file ADATA (unit 10).
3. Trips to be found, on file TRIPS (unit 11).

A.8.3 Program Output

All program output is produced on the line printer and consists of the following information:

1. The origin, destination and departure time for each trip.
2. The trip itinerary with route, boarding stop, boarding time, alighting stop and alighting time for each segment of the trip.
3. Computation time for the trip calculation.
4. Mean and standard deviation of computation times for all trips.

A.9. PROGRAM REMOVE

This program removes unnecessary nodes from a transit network, leaving only those nodes at which transferring is possible and likely. All nodes which are on only one route and which are not the first or last node on that route are deleted since no transferring is possible at any of these nodes. When several routes have a segment in common, intermediate nodes on that segment can be deleted, since any transfers can take place at the initial or final node of the segment. Thus a node is removed whenever it lies between the same two stops on all routes which stop at that node.

Program input consists of the stops on each route from file SDATA. The program then sorts the nodes on the number of routes stopping at the node and deletes all nodes serviced by only one route, as long as the node is not the first or last stop on the route. Finally other nodes are examined to see if they lie on the same segment on all routes. The program prints a list of deleted nodes and the revised network.

A.9.1 Variables and Arrays used in REMOVE

Route Input

- SRTE(i) - stops on each route. REND is used to indicate which section of SRTE refers to a particular route
- REND(i) - position in SRTE of the last stop of route i

Stop Description

- RSTOP(i,j) - jth route stopping at node i
- NRTE(i) - number of routes stopping at node i
- NIN(i) - is true if node i has not been removed (i.e., node i is in the network) and is false if node i has been removed.

Temporary storage

- TEMP(j) - used for input and output of the nodes on a route. Also used as an intermediate array storing the position in each route stopping at a node of that node.
- ORDER(i) - used in sorting; the original position of the ith entry in sorted order

SORTN(i)	-	used in sorting; contains the sorted array in sorted order
TIME(j)	-	used for storing schedule data
INDEX(j)	-	used while printing the revised network data. INDEX(j) is the position in the old route stop list of the jth stop in the revised network.

Counters

NRTE	-	number of routes
NR	-	total number of stops on routes
NEND	-	number of nodes which begin or end a route
NDEL	-	number of nodes deleted
NS	-	number of stops on the current route
NT	-	number of runs in the schedule for the current route

A.9.2 Program Input

Input to program REMOVE consists of the file SDATA (on unit 7) containing route descriptions and schedules for the network.

A.9.3 Program Output

Program output is a set of revised routes and schedules written on logical unit 13 in the format for file SDATA. This file can be used instead of SDATA in all subsequent runs (and thus be assigned to unit 7 for other runs).

In addition to creating a revised SDATA file, program REMOVE prints out a list of the deleted nodes and also the revised routes. Two error prints may occur, most commonly because of improper dimensions.

A.10 FILE FORMATS

A.10.1 Format for File SDATA (unit 7)

Contents	Format
route number, number of stops, number of runs, stops on route	20I5

For each run on the route:

time at each stop 20I5

A.10.2 Format for File TDATA (unit 8)

For each node and possible transfer:

node,
from route,
to route,
minimum transfer time 4I5

A.10.3 Format for File NDATA (unit 9)

stop, 2I5
time

A.10.4 Format for File ADATA (unit 10)

arc origin node,
arc destination node,
route number 3I5

A.10.5 Format for File TRIPS (unit 11)

For each desired trip itinerary:

trip origin,
trip destination,
desired departure time 3I5

@EOF

A.10.6 Format for File TMIN (unit 12)

minimum transfer time
at each node 16I5

@EOF

APPENDIX B

LISTINGS OF PROGRAMS FOR TEST-PROBLEM GENERATION AND ITINERARY-FINDING

B.1 PROGRAM RAD

```
C THIS PROGRAM SERVES BOTH AS AN INPUT ROUTINE AND AS A DRIVER
C ROUTINE TO GENERATE A RADIAL TRANSIT ROUTE SYSTEM.
C THE USER SPECIFIES THE NUMBER OF RADIALS, THE DIRECTION OF EACH
C RADIAL. THE NUMBER OF STOPS AND DISTANCES BETWEEN STOPS ALONG EACH
C RADIAL.  THE USER ALSO MUST SPECIFY BELTWAYS WHICH CONNECT STOPS ON
C DIFFERENT RADIALS AND SPIKES WHICH CONNECT A STOP ON A RADIAL WITH A
C POINT NOT ON ANY RADIAL.
C
C---------
C
C NATIONAL BUREAU OF STANDARDS      APRIL, 1976
C          REVISED DECEMBER 1976 BY E. LEYENDECKER
C
C---------
C
      PARAMETER  MRADII=20    @ MAX NUMBER OF RADIALS
      PARAMETER  MSTPS =50    @ NUMBER OF STOPS PER RADIAL
      PARAMETER  MNODES=MSTPS*MRADII    @ NUMBER OF NODES ON RADIALS
      PARAMETER  MSTOPS=50    @ NUMBER OF STOPS PER ROUTE
      PARAMETER  MBELT =10    @ MAX NUMBER OF BELTWAYS
      PARAMETER  MSPIKE=20    @ MAX NUMBER OF SPIKES
      LOGICAL KEXP
      INTEGER REXP,BEXP,SEXP
      COMMON/NODES/KRAD(MNODES),KSTP(MNODES),KEXP(MNODES),
     1               NODE(MRADII,MSTPS)
      COMMON/GENRAL/MRTE(2,MSTOPS),DTEMP(2,MSTOPS),ZK(20),XZK(20),IPD
      COMMON/RADIAL/ANGLE(MRADII),ISTP(MRADII),DSTRAD(MRADII,MSTPS),
     1                REXP(MRADII),IRAD
      COMMON/BELTWY/IBIN(MBELT),IBFIN(MBELT),IBSTP(MBELT,MRADII),
     1                NB(MBELT),BEXP(MBELT),DSTBLT(MBELT,MRADII),IBELT
      COMMON/SPIKE/ISRAD(MSPIKE),ISSTP(MSPIKE),DSPIKE(MSPIKE),
     1               SEXP(MSPIKE),NDSPK(MSPIKE),ISPIKE
C
C---------
C
C VARIABLES AND ARRAYS USED IN THIS PROGRAM
C
C    RADIAL INPUT
C      IRAD - NUMBER OF RADIALS
C      ANGLE(I) - ANGLE OF RADIAL I, MEASURED COUNTERCLOCKWISE FROM EAST
C      ISTP(I) - NUMBER OF STOPS ON RADIAL I, NOT COUNTING THE CENTER
C      DSTRAD(I,J) - DISTANCE BETWEEN STOPS J AND J+1 ON RADIAL I
C      REXP(I) - A FLAG SET TO 1 IF RADIAL I IS AN EXPRESS ROUTE
C    BELTWAY INPUT
C      IBELT - NUMBER OF BELTWAYS
C      IBIN(I) - INITIAL RADIAL CONNECTED BY BELTWAY I
C      IBFIN(I) - FINAL RADIAL CONNECTED BY BELTWAY I
C      IBSTP(I,J) - STOP ON RADIAL J ON BELTWAY I
C      BEXP(I) - A FLAG SET TO 1 IF BELTWAY I IS AN EXPRESS ROUTE
```

```
C     SPIKE INPUT
C       ISPIKE - NUMBER OF SPIKES
C       ISRAD(I) - RADIAL TO WHICH SPIKE I IS CONNECTED
C       ISSTP(I) - STOP ON RADIAL TO WHICH SPIKE I IS CONNECTED
C       DSPIKE(I) - LENGTH OF SPIKE I
C       SEXP(I) - A FLAG SET TO 1 IF SPIKE IS AN EXPRESS ROUTE
C     NODE DESCRIPTION
C       KRAD(K) - RADIAL NODE K IS ON
C       KSTP(K) - STOP ON RADIAL KRAD(K) WHICH IS NODE K
C       NODE(I,J) - THE NODE FOR STOP J ON RADIAL I
C     ROUTE DESCRIPTION
C       MRTE(M,N) - NTH STOP ON ROUTE M.  MRTE(1,J)=MRTE(2,N-J+1) SINCE
C                   ROUTES GO IN BOTH DIRECTIONS
C       DTEMP(N,M) - DISTANCE BETWEEN STOPS N AND N+1 ALONG ROUTE
C
C----------
C
C      INITIALIZE
C
       K=1       @ NODE COUNTER
       M1=-1     @ ROUTE COUNTER IN ONE DIRECTION
       M2=0      @ ROUTE COUNTER IN OTHER DIRECTION
       P=3.14159265/180.    @DEGREES TO RADIANS
       DO 5 I=1,MNODES
       KEXP(I)=.FALSE.
     5 CONTINUE
C
C      READ IN NUMBER OF TIME PERIODS AND CONVERSTION FACTORS
C
       READ(5,900) IPD
       READ(5,902) (ZK(I),I=1,IPD)
       READ(5,902) (XZK(I),I=1,IPD)
C
C      READ IN RADIAL DESCRIPTION
C
       READ(5,900) IRAD
       DO 10 I=1,IRAD
       READ(5,900) REXP(I)
       READ(5,901) ANGLE(I),ISTP(I)
       ANGLE(I)=ANGLE(I)*P
       JSTP=ISTP(I)
       READ(5,902) (DSTRAD(I,J),J=1,JSTP)
    10 CONTINUE
C
C      READ BELTWAY DESCRIPTION
C
       READ(5,900) IBELT
       DO 20 I=1,IBELT
       READ(5,900) BEXP(I)
       READ(5,900) IBIN(I),IBFIN(I)
```

```
      NB(I)=IBFIN(I)-IBIN(I)+1
      IF(IBIN(I).GE.IBFIN(I)) NB(I)=IBFIN(I)+IRAD-IBIN(I)+1
      N=NB(I)
      READ(5,900) (IBSTP(I,J),J=1,N)
   20 CONTINUE
C
C     READ SPIKE DESCRIPTION
C
      READ(5,900) ISPIKE
      IF(ISPIKE.EQ.0)GO TO 40
      DO 30 I=1,ISPIKE
      READ(5,900) SEXP(I)
      READ(5,903) ISRAD(I),ISSTP(I),DSPIKE(I)
   30 CONTINUE
C
C     CALL SUBROUTINE LOCAL TO GENERATE LOCAL ROUTES
C
   40 CALL LOCAL(K,M1,M2)
C
C     CALL SUBROUTINE EXPRES TO GENERATE EXPRESS ROUTES
C
      CALL EXPRES(K,M1,M2)
C
C     ENDFILE UNIT 7 - SCHEDULE INFORMATION FILE
C
      ENDFILE 7
      STOP
  900 FORMAT(16I5)
  901 FORMAT(F10.1,I5)
  902 FORMAT(8F10.1)
  903 FORMAT(2I5,F10.1)
      END
```

B.1.1 Subroutine LOCAL

```
      SUBROUTINE LOCAL(K,M1,M2)
C THIS SUBROUTINE GENERATES THE LOCAL ROUTES OF A RADIAL TRANSIT ROUTE
C SYSTEM.  TWO-WAY ROUTES ARE CONSTRUCTED CONNECTING THE CENTER WITH THE
C END OF EACH RADIAL, THE CENTER WITH THE ENDPOINT OF EACH SPIKE, AND
C THE ENDPOINTS OF EACH BELTWAY. PROGRAM OUTPUT INCLUDES THE SCHEDULE
C FOR EACH ROUTE, THE STOPS ON EACH ROUTE AND THE ROUTES STOPPING AT
C EACH STOP.
C
C---------
C
C NATIONAL BUREAU OF STANDARDS      APRIL, 1976
C          REVISED DECEMBER 1976 BY E. LEYENDECKER
C
C---------
C
      PARAMETER  MRADII=20    @ MAX NUMBER OF RADIALS
      PARAMETER  MSTPS =50    @ NUMBER OF STOPS PER RADIAL
      PARAMETER  MNODES=MSTPS*MRADII    @ NUMBER OF NODES ON RADIALS
      PARAMETER  MSTOPS=50    @ NUMBER OF STOPS PER ROUTE
      PARAMETER  MBELT =10    @ MAX NUMBER OF BELTWAYS
      PARAMETER  MSPIKE=20    @ MAX NUMBER OF SPIKES
      LOGICAL KEXP
      INTEGER REXP,BEXP,SEXP
      COMMON/NODES/KRAD(MNODES),KSTP(MNODES),KEXP(MNODES),
     1             NODE(MRADII,MSTPS)
      COMMON/GENRAL/MRTE(2,MSTOPS),DTEMP(2,MSTOPS),ZK(20),XZK(20),IPD
      COMMON/RADIAL/ANGLE(MRADII),ISTP(MRADII),DSTRAD(MRADII,MSTPS),
     1               REXP(MRADII),IRAD
      COMMON/BELTWY/IBIN(MBELT),IBFIN(MBELT),IBSTP(MBELT,MRADII),
     1               NB(MBELT),BEXP(MBELT),DSTBLT(MBELT,MRADII),IBELT
      COMMON/SPIKE/ISRAD(MSPIKE),ISSTP(MSPIKE),DSPIKE(MSPIKE),
     1             SEXP(MSPIKE),NDSPK(MSPIKE),ISPIKE
C
C---------
C
C VARIABLES AND ARRAYS USED IN THIS PROGRAM
C
C    RADIAL
C      IRAD - NUMBER OF RADIALS
C      ANGLE(I) - ANGLE OF RADIAL I, MEASURED COUNTERCLOCKWISE FROM EAST
C      ISTP(I) - NUMBER OF STOPS ON RADIAL I, NOT COUNTING THE CENTER
C      DSTRAD(I,J) - DISTANCE BETWEEN STOPS J AND J+1 ON RADIAL I
C      REXP(I) - A FLAG SET TO 1 IF RADIAL I IS AN EXPRESS ROUTE
C    BELTWAY
C      IBELT - NUMBER OF BELTWAYS
C      IBIN(I) - INITIAL RADIAL CONNECTED BY BELTWAY I
C      IBFIN(I) - FINAL RADIAL CONNECTED BY BELTWAY I
C      IBSTP(I,J) - STOP ON RADIAL J ON BELTWAY I
C      DSTBLT(I,J) - DISTANCE BETWEEN STOPS J AND J+1 ON BELTWAY I
C      NB(I) - NUMBER OF RADIALS INTERSECTED BY BELTWAY I
C      BEXP(I) - A FLAG SET TO 1 IF BELTWAY I IS AN EXPRESS ROUTE
```

```
C     SPIKE
C       ISPIKE - NUMBER OF SPIKES
C       ISRAD(I) - RADIAL TO WHICH SPIKE I IS CONNECTED
C       ISSTP(I) - STOP ON RADIAL TO WHICH SPIKE I IS CONNECTED
C       DSPIKE(I) - LENGTH OF SPIKE I
C       NDSPK(I) - NODE NUMBER FOR STOP AT END OF SPIKE
C       SEXP(I) - A FLAG SET TO 1 IF SPIKE IS AN EXPRESS ROUTE
C     NODE DESCRIPTION
C       KRAD(K) - RADIAL NODE K IS ON
C       KSTP(K) - STOP ON RADIAL KRAD(K) WHICH IS NODE K
C       KEXP(K) - NODE IS/IS NOT AN EXPRESS NODE
C       NODE(I,J) - THE NODE FOR STOP J ON RADIAL I
C     ROUTE DESCRIPTION
C       MRTE(M,N) - NTH STOP ON ROUTE M.  MRTE(1,J)=MRTE(2,N-J+1) SINCE
C                   ROUTES GO IN BOTH DIRECTIONS
C       DTEMP(N,M) - DISTANCE BETWEEN STOPS N AND N+1 ALONG ROUTE
C
C----------
C      INITIALIZE
C----------
C
      PI=3.14159265
C
C----------
C COMPUTE RADIALS
C----------
C
      DO 2 I=1,IRAD
      NODE(I,1)=1
      M1=M1+2
      M2=M2+2
      ISTP1=ISTP(I)+1
      MRTE(1,1)=1
      MRTE(2,ISTP1)=1
      DO 1 J=2,ISTP1
C COMPUTE ROUTES
      K=K+1
      MRTE(1,J)=K
      MRTE(2,ISTP1+1-J)=K
      JJ=J-1
      D=DSTRAD(I,JJ)
      DTEMP(1,JJ)=D
      DTEMP(2,ISTP1-JJ)=D
C COMPUTE NODE DATA
      KRAD(K)=I
      KSTP(K)=J
      NODE(I,J)=K
1     CONTINUE
      CALL RSCHED(ISTP1,M1,1)
2     CONTINUE
      KMAX=K    ONUMBER OF NODES ON RADIALS
```

```
C
C---------
C COMPUTE BELTWAYS
C---------
C
      DO 3 IJ=1,IBELT
      M1=M1+2
      M2=M2+2
      I=IBIN(IJ)
      N=NB(IJ)
      DO 4 II=1,N
C COMPUTE ROUTES
      J=IBSTP(IJ,II)
      N1=NODE(I,J)
      MRTE(1,II)=N1
      MRTE(2,N+1-II)=N1
      IF (II.EQ.N) GO TO 4
      IT=II+1
      J1=IBSTP(IJ,IT)
      I1=I+1
      IF (I1.GT.IRAD) I1=I1-IRAD
      N2=NODE(I1,J1)
C COMPUTE THE ARC LENGTH AS THE AVERAGE OF THE ARC LENGTHS BETWEEN THE
C TWO RADIALS AT RADII OF STOP J ON RADIAL I AND STOP J1 ON RADIAL I1
      A=ANGLE(I1)-ANGLE(I)
      IF (A.LT.0.) A=A+2.*PI
      R1=0.
      JEND=J-1
      DO 31 JJ=1,JEND
      R1=R1+DSTRAD(I,JJ)
31    CONTINUE
      R2=0.
      JEND=J1-1
      DO 32 JJ=1,JEND
      R2=R2+DSTRAD(I1,JJ)
32    CONTINUE
      R=(R2+R1)/2.
      R=ABS(R)
      D=R*A
      DTEMP(1,II)=D
      DTEMP(2,N-II)=D
      DSTBLT(IJ,II)=D
      I=I+1
      IF (I.GT.IRAD) I=1
4     CONTINUE
      CALL RSCHED(NB(IJ),M1,1)
    3 CONTINUE
```

```
C
C---------
C COMPUTE SPIKE ROUTES
C---------
C
      IF(ISPIKE.EQ.0)RETURN
      DO 5 I=1,ISPIKE
      K=K+1
      NDSPK(I)=K
      M1=M1+2
      M2=M2+2
C COMPUTE SPIKE ARCS
C COMPUTE ROUTES
      MM=ISSTP(I)+1
      MRTE(1,1)=1
      MRTE(2,MM)=1
      MRTE(1,MM)=K
      MRTE(2,1)=K
      LSTP=ISSTP(I)
      DTEMP(1,LSTP)=DSPIKE(I)
      DTEMP(2,1)=DSPIKE(I)
      LRAD=ISRAD(I)
      J1=NODE(LRAD,2)
      N1=NODE(LRAD,LSTP)
      MT=MM
      II=1
      DO 6 J=J1,N1
      DTEMP(1,II)=DSTRAD(LRAD,II)
      DTEMP(2,MM-II)=DSTRAD(LRAD,II)
      II=II+1
      MT=MT-1
      MRTE(1,II)=J
      MRTE(2,MT)=J
6     CONTINUE
      CALL RSCHED(MM,M1,1)
    5 CONTINUE
      RETURN
      END
```

B.1.2 Subroutine EXPRES

```
      SUBROUTINE EXPRES(K,M1,M2)
C THIS SUBROUTINE GENERATES THE EXPRESS ROUTES OF A RADIAL TRANSIT ROUTE
C SYSTEM.  TWO-WAY ROUTES ARE CONSTRUCTED FOR RADIAL, BELTWAY AND SPIKE
C EXPRESS ROUTES.  PROGRAM OUTPUT INCLUDES THE SCHEDULE FOR EACH EXPRESS
C ROUTE, THE STOPS ON EACH EXPRESS ROUTE AND THE ROUTES STOPPING AT EACH
C STOP. EXPRESS ROUTES STOP ONLY AT THE OTHER MAJOR ROUTES.
C
C---------
C
C NATIONAL BUREAU OF STANDARDS   JANUARY 1977   E.LEYENDECKER
C
C---------
C
      PARAMETER  MRADII=20    @ MAX NUMBER OF RADIALS
      PARAMETER  MSTPS =50    @ NUMBER OF STOPS PER RADIAL
      PARAMETER  MNODES=MSTPS*MRADII    @ NUMBER OF NODES ON RADIALS
      PARAMETER  MSTOPS=50    @ NUMBER OF STOPS PER ROUTE
      PARAMETER  MBELT =10    @ MAX NUMBER OF BELTWAYS
      PARAMETER  MSPIKE=20    @ MAX NUMBER OF SPIKES
      LOGICAL KEXP
      INTEGER REXP,BEXP,SEXP
      COMMON/NODES/KRAD(MNODES),KSTP(MNODES),KEXP(MNODES),
     1            NODE(MRADII,MSTPS)
      COMMON/GENRAL/MRTE(2,MSTOPS),DTEMP(2,MSTOPS),ZK(20),XZK(20),IPD
      COMMON/RADIAL/ANGLE(MRADII),ISTP(MRADII),DSTRAD(MRADII,MSTPS),
     1            REXP(MRADII),IRAD
      COMMON/BELTWY/IBIN(MBELT),IBFIN(MBELT),IBSTP(MBELT,MRADII),
     1            NB(MBELT),BEXP(MBELT),DSTBLT(MBELT,MRADII),IBELT
      COMMON/SPIKE/ISRAD(MSPIKE),ISSTP(MSPIKE),DSPIKE(MSPIKE),
     1            SEXP(MSPIKE),NDSPK(MSPIKE),ISPIKE
C
C---------
C
C VARIABLES AND ARRAYS USED IN THIS PROGRAM
C
C    RADIAL
C      IRAD - NUMBER OF RADIALS
C      ANGLE(I) - ANGLE OF RADIAL I, MEASURED COUNTERCLOCKWISE FROM EAST
C      ISTP(I) - NUMBER OF STOPS ON RADIAL I, NOT COUNTING THE CENTER
C      DSTRAD(I,J) - DISTANCE BETWEEN STOPS J AND J+1 ON RADIAL I
C      REXP(I) - A FLAG SET TO 1 IF RADIAL I IS AN EXPRESS ROUTE
C    BELTWAY
C      IBELT - NUMBER OF BELTWAYS
C      IBIN(I) - INITIAL RADIAL CONNECTED BY BELTWAY I
C      IBFIN(I) - FINAL RADIAL CONNECTED BY BELTWAY I
C      IBSTP(I,J) - STOP ON RADIAL J ON BELTWAY I
C      DSTBLT(I,J) - DISTANCE BETWEEN STOPS J AND J+1 ON BELTWAY I
C      NB(I) - NUMBER OF RADIALS INTERSECTED BY BELTWAY I
C      BEXP(I) - A FLAG SET TO 1 IF BELTWAY I IS AN EXPRESS ROUTE
```

```
C     SPIKE
C        ISPIKE - NUMBER OF SPIKES
C        ISRAD(I) - RADIAL TO WHICH SPIKE I IS CONNECTED
C        ISSTP(I) - STOP ON RADIAL TO WHICH SPIKE I IS CONNECTED
C        DSPIKE(I) - LENGTH OF SPIKE I
C        NDSPK(I) - NODE NUMBER FOR STOP AT END OF SPIKE
C        SEXP(I) - A FLAG SET TO 1 IF SPIKE IS AN EXPRESS ROUTE
C     NODE DESCRIPTION
C        KRAD(K) - RADIAL NODE K IS ON
C        KSTP(K) - STOP ON RADIAL KRAD(K) WHICH IS NODE K
C        KEXP(K) - NODE IS/IS NOT AN EXPRESS NODE
C        NODE(I,J) - THE NODE FOR STOP J ON RADIAL I
C     ROUTE DESCRIPTION
C        MRTE(M,N) - NTH STOP ON ROUTE M.   MRTE(1,J)=MRTE(2,N-J+1) SINCE
C                    ROUTES GO IN BOTH DIRECTIONS
C        DTEMP(N,M) - DISTANCE BETWEEN STOPS N AND N+1 ALONG ROUTE
C
C---------
C      INITIALIZE
C---------
C
      KSTOP=0          @ KSTOP COUNTS STOPS ON AN EXPRESS ROUTE
      DIST=0.0         @ DISTANCE ACCUMULATOR FOR EXPRESS ROUTES
C
C---------
C     COMPUTE EXPRESS BELTWAYS
C---------
C
      DO 200 I=1,IBELT
      IF(BEXP(I).NE.1)GO TO 200
      M1=M1+2
      M2=M2+2
      IR=IBIN(I)
      NUMB=NB(I)
C*****CALCULATE EXPRESS STOPS AND INTER-STOP DISTANCES IN ONE DIRECTION
      DO 150 II=1,NUMB
      J=IBSTP(I,II)
      ND=NODE(IR,J)
      IF(IR.EQ.IBIN(I).OR.IR.EQ.IBFIN(I))GO TO 110
      IF(REXP(IR).EQ.1)GO TO 110
      DO 100 IS=1,ISPIKE
      IF(ISRAD(IS).NE.IR)GO TO 100
      IF(SEXP(IS).EQ.1.AND.ISSTP(IS).GE.J)GO TO 110
  100 CONTINUE
      GO TO 120
  110 KSTOP=KSTOP+1
      MRTE(1,KSTOP)=ND
      KEXP(ND)=.TRUE.
```

```
  120 IF(II.EQ.1.OR..NOT.KEXP(ND))GO TO 130
      DTEMP(1,KSTOP-1)=DIST
      DIST=0.0
  130 IF(II.EQ.NUM )GO TO 140
      DIST=DIST+DSTPLT(I,II)
  140 IR=IR+1
      IF(IR.GT.IRAD)IR=1
  150 CONTINUE
C*****FILL MRTE(2, ) AND DTEMP(2, )
      DO 160 II=1,KSTOP
      MRTE(2,KSTOP+1-II)=MRTE(1,II)
      IF(II.NE.KSTOP) DTEMP(2,KSTOP-II)=DTEMP(1,II)
  160 CONTINUE
C*****CALL RSCHED
      CALL RSCHED(KSTOP,M1,2)
      KSTOP=0
  200 CONTINUE
C
C---------
C     COMPUTE EXPRESS SPIKES
C---------
C
      IF(ISPIKE.EQ.0)GO TO 500
      DO 400 I=1,ISPIKE
      IF(SEXP(I).NE.1)GO TO 400
      LRAD=ISRAD(I)
      M1=M1+2
      M2=M2+2
      MM=ISSTP(I)
C*****CALCULATE EXPRESS STOPS AND INTER-STOP DISTANCES IN ONE DIRECTION
      DO 350 II=1,MM
      ND=NODE(LRAD,II)
      IF(II.EQ.1)GO TO 310
      IF(KEXP(ND))GO TO 310
      IF(II.NE.MM)GO TO 320
      IF(REXP(LRAD).EQ.1)GO TO 310
      DO 300 IS=1,ISPIKE
      IF(I.EQ.IS.OR.ISRAD(IS).NE.LRAD)GO TO 300
      IF(SEXP(IS).EQ.1.AND.ISSTP(IS).GE.MM)GO TO 310
  300 CONTINUE
      GO TO 320
  310 KSTOP=KSTOP+1
      MRTE(1,KSTOP)=ND
      KEXP(ND)=.TRUE.
  320 IF(II.EQ.1.OR..NOT.KEXP(ND))GO TO 330
      DTEMP(1,KSTOP-1)=DIST
      DIST=0.0
  330 IF(II.LT.MM) DIST=DIST+DSTRAD(LRAD,II)
      IF(II.EQ.MM) DIST=DIST+DSPIKE(I)
  350 CONTINUE
```

```
C*****CONSIDER STOP AT END OF SPIKE
      KSTOP=KSTOP+1
      MRTE(1,KSTOP)=NDSPK(I)
      DTEMP(1,KSTOP-1)=DIST
      DIST=0.0
C*****FILL MRTE(2, ) AND DTEMP(2, )
      DO 360 II=1,KSTOP
      MRTE(2,KSTOP+1-II)=MRTE(1,II)
      IF(II.NE.KSTOP) DTEMP(2,KSTOP-II)=DTEMP(1,II)
  360 CONTINUE
C*****CALL RSCHED
      CALL RSCHED(KSTOP,M1,2)
      KSTOP=0
  400 CONTINUE
C
C---------
C     COMPUTE EXPRESS RADIALS
C---------
C
  500 DO 600 I=1,IRAD
      IF(REXP(I).NE.1) GO TO 600
      M1=M1+2
      M2=M2+2
      MM=ISTP(I)+1
C*****CALCULATE EXPRESS STOPS AND INTER-STOP DISTANCES IN ONE DIRECTION
      DO 650 II=1,MM
      ND=NODE(I,II)
      IF(II.EQ.1.OR.II.EQ.MM)GO TO 610
      IF(KEXP(ND))GO TO 610
      GO TO 620
  610 KSTOP=KSTOP+1
      MRTE(1,KSTOP)=ND
  620 IF(II.EQ.1)GO TO 630
      IF(II.NE.MM.AND..NOT.KEXP(ND))GO TO 630
      DTEMP(1,KSTOP-1)=DIST
      DIST=0.0
  630 IF(II.LT.MM) DIST=DIST+DSTRAD(I,II)
  650 CONTINUE
C*****FILL MRTE(2, ) AND DTEMP(2, )
      DO 660 II=1,KSTOP
      MRTE(2,KSTOP+1-II)=MRTE(1,II)
      IF(II.NE.KSTOP) DTEMP(2,KSTOP-II)=DTEMP(1,II)
  660 CONTINUE
C*****CALL RSCHED
      CALL RSCHED(KSTOP,M1,2)
      KSTOP=0
  600 CONTINUE
      RETURN
      END
```

B.1.3 Subroutine RSCHED

```
      SUBROUTINE RSCHED(N,M1,LOF)
C THIS SUBROUTINE COMPUTES AND PRINTS OUT SCHEDULE INFORMATION FOR
C M1 AND M1+1.  INPUT INCLUDES THE STOPS ALONG ROUTES M1 AND M1+1,
C STORED IN MRTE(1,I) AND MRTE(2,I) RESPECTIVELY, THE NUMBER OF
C STOPS ALONG THESE ROUTES IN N, AND THE DISTANCE BETWEEN STOPS IN D.
C INPUT ALSO INCLUDES THE NUMBER OF TIME PERIODS IPD AND THE FACTOR
C USED TO TRANSLATE DISTANCE TO TIME IN EACH PERIOD STORED IN ZK AND XZK.
      PARAMETER  MSTOPS=50
      PARAMETER  MPDS=10
      COMMON/GENRAL/MRTE(2,MSTOPS),DTEMP(2,MSTOPS),ZK(20),XZK(20),IPD
      DIMENSION  JRUNS(MPDS),JHEAD(MPDS),JDTM(MPDS),JTIME(MSTOPS)
      MM=M1
      DO 3 IROUTE=1,2
      NRUNS=0
      DO 10 IPERD=1,IPD
C READ THE NUMBER OF RUNS OF THIS ROUTE THIS PERIOD (JRUNS), THE
C HEADWAY BETWEEN RUNS (JHEAD), AND THE TIME OF THE FIRST RUN (JDTM).
      READ (5,901) JRUNS(IPERD),JHEAD(IPERD),JDTM(IPERD)
901   FORMAT (20I5)
      NRUNS=NRUNS+JRUNS(IPERD)
10     CONTINUE
      WRITE (6,900) MM,N,NRUNS,(MRTE(IROUTE,J),J=1,N)
900   FORMAT('0ROUTE',I4,' HAS',I3,' STOPS,',I4,' RUNS'/5X,'STOPS',20I5)
      WRITE (7,901) MM,N,NRUNS,(MRTE(IROUTE,J),J=1,N)
      DO 21 IPERD=1,IPD
      JJ=JRUNS(IPERD)
      IF (JJ.EQ.0) GO TO 21
      DO 2 IRUN=1,JJ
      JTIME(1)=JDTM(IPERD)+(IRUN-1)*JHEAD(IPERD)
      DO 1 L=2,N
      IF(LOF.EQ.1)JTIME(L)=JTIME(L-1)+DTEMP(IROUTE,L-1)*ZK(IPERD)
      IF(LOE.EQ.2)JTIME(L)=JTIME(L-1)+DTEMP(IROUTE,L-1)*XZK(IPERD)
1     CONTINUE
      WRITE (6,902) (JTIME(L),L=1,N)
902   FORMAT (10X,20I5)
      WRITE (7,901) (JTIME(L),L=1,N)
2     CONTINUE
21    CONTINUE
      MM=MM+1
3     CONTINUE
      RETURN
      END
```

B.2 PROGRAM XGRID

```
C  **************** XGRID : PURE GRID GENERATOR ****************
C  THIS PROGRAM GENERATES A PURE GRID TRANSIT SYSTEM WITH NORTH-SOUTH,
C  EAST-WEST ROUTES.  THE USER SPECIFIES THE NUMBER OF STOPS P AND THE
C  NUMBER OF STOPS Q WHICH DEFINE,RESPECTIVELY, THE VERTICAL AND
C  HORIZONTAL DIMENSIONS OF THE P-BY-Q GRID.  IN ADDITION THE USER
C  MUST SPECIFY THE INTERSTOP DISTANCES L(I) AND W(I) BETWEEN SUCCESSIVE
C  'ROWS' AND 'COLUMNS' OF THE GRID.  STOPS WILL BE NUMBERED FROM LEFT
C  TO RIGHT, TOP TO BOTTOM IN SEQUENCE.  ANY STOP IS ALLOWED TO BE A
C  TRANSFER NODE BETWEEN ROUTES, WHICH ARE ASSUMED TO RUN WEST-EAST,
C  EAST-WEST, NORTH-SOUTH AND SOUTH-NORTH ONLY.  THE TRANSFER TIME
C  BETWEEN ROUTES IS ASSUMED TO BE CONSTANT AT EACH TRANSFER STOP.
C  ROUTES FOLLOW REGULAR SCHEDULES THROUGHOUT EACH PERIOD, AND THERE
C  MAY BE A DIFFERENT CONVERSION FACTOR FOR CONVERTING DISTANCE INTO
C  TIME FOR EACH PERIOD.  THIS VERSION ALLOWS EXPRESS ROUTES
C  BY THE SPECIFICATION OF MAJOR X AND Y ROUTES.  THESE ROUTES
C  STOP ONLY AT THE OTHER MAJOR ROUTES.
C
C--------
C
C  NATIONAL BUREAU OF STANDARDS          MAY, 1976
C
C--------
C
      PARAMETER MAXP=50         @ DIMENSION P OF GRID
      PARAMETER MAXQ=50         @ DIMENSION Q OF GRID
      PARAMETER MAXPD=10        @ NUMBER OF PERIODS
      PARAMETER NRTES=200       @ NUMBER OF ROUTES
      PARAMETER MAXN=50         @ NUMBER OF NODES/ROUTE
      PARAMETER TRANSF=5000     @ NUMBER OF TRANSFERS
      DIMENSION  L(MAXP),W(MAXQ),ZK(MAXPD),XZK(MAXPD)
      COMMON  NN(NRTES),NODE(NRTES,MAXN),D(NRTES,MAXN),
     1        TNODE(TRANSF),RT1(TRANSF),RT2(TRANSF),NTRANS
C
C--------
C
C  VARIABLES AND ARRAYS USED IN THIS PROGRAM
C
C    INPUT PARAMETERS
C      P - VERTICAL DIMENSION OF GRID
C      Q - HORIZONTAL DIMENSION OF GRID
C      L(I) - DISTANCE BETWEEN VERTICAL ROWS I,I-1
C      W(I) - DISTANCE BETWEEN HORIZONTAL COLUMNS I,I-1
C      PDS - NUMBER OF PERIODS
C      ZK(J) - CONVERTS DISTANCE INTO TIME FOR PERIOD J
C
C    ROUTE DESCRIPTION
C      NN(R) - NUMBER OF NODES ON ROUTE R
C      NODE(R,I) - THE I-TH NODE ON ROUTE R
C      D(R,I) - THE I-TH INTERSTOP DISTANCE ALONG ROUTE R
C
```

```
C     TRANSFER NODE DESCRIPTION
C        TNODE(L) - THE L-TH TRANSFER NODE
C        RT1(L) - ROUTE FROM WHICH TRANSFER IS MADE AT TNODE(L)
C        RT2(L) - ROUTE TO WHICH TRANSFER IS MADE AT TNODE(L)
C        NTRANS - NUMBER OF TRANSFERS
C
C--------
C
      IMPLICIT INTEGER (A-Y)
      REAL XZK
C
C  READ STRUCTURAL PARAMETERS OF GRID.
C
      READ(5,900) P,Q,(L(I),I=2,P),(W(I),I=2,Q)
  900 FORMAT(16I5)
      RBASE=0     @ CONVENIENT REFERENCE BASE FOR ABSOLUTE ROUTE NUMBERS
C
C  CREATE NODES,ROUTES,TRANSFER DATA.
C
      CALL GRID(P,Q,L,W,RBASE)
      CALL XPRESS(P,Q,L,W,RBASE)
C
C  READ CONVERSION FACTORS FOR EACH PERIOD.
C
      READ(5,901) PDS,(ZK(I),I=1,PDS)
  901 FORMAT(I5,(15F5.2))
      READ (5,902) (XZK(I),I=1,PDS)
902   FORMAT (5X,15F5.2)
C
C  COMPUTE COMPLETE SCHEDULE INFORMATION FOR EACH ROUTE AND PERIOD.
C
      PQ=2*P+2*Q
      CALL XSCHED(NN,NODE,D,ZK,XZK,PDS,RBASE,PQ)
C
C  THE TRANSFER DATA GENERATED IN GRID IS PRINTED.
C
      PQ=P*Q
      CALL TRANS(NTRANS,TNODE,RT1,RT2,PQ,RBASE)
      STOP
      END
```

B.2.1 Subroutine GRID

```
      SUBROUTINE GRID(P,Q,LL,W,RBASE)
C   THIS SUBROUTINE GENERATES THE NODES,ROUTES AND TRANSFER
C   DATA FOR A P-BY-Q GRID NETWORK.  THE VARIABLES P,Q,RBASE AND
C   THE ARRAYS LL,W ARE TRANSMITTED FROM THE MAIN PROGRAM.  IT IS
C   ASSUMED THAT P,Q > 1.  NODES ARE NUMBERED FROM LEFT TO RIGHT,
C   TOP TO BOTTOM IN SEQUENCE.  ROUTES ARE ASSUMED TO RUN WEST-
C   EAST,EAST-WEST,NORTH-SOUTH,SOUTH-NORTH ONLY.  TRANSFERS ARE
C   ALLOWED TO OCCUR BETWEEN ROUTES AT ANY NODE.
C
C--------
C
C   NATIONAL BUREAU OF STANDARDS        MAY,1976
C
C--------
C
      PARAMETER MAXP=50         @ DIMENSION P OF GRID
      PARAMETER MAXQ=50         @ DIMENSION Q OF GRID
      PARAMETER NRTES=200       @ NUMBER OF ROUTES
      PARAMETER MAXN=50         @ NUMBER OF NODES/ROUTE
      PARAMETER TRANSF=5000     @ NUMBER OF TRANSFERS
      DIMENSION LL(MAXP),W(MAXQ)
      COMMON NN(NRTES),NODE(NRTES,MAXN),D(NRTES,MAXN),
     1        TNODE(TRANSF),RT1(TRANSF),RT2(TRANSF),NTRANS
C
C--------
C
C   VARIABLES AND ARRAYS USED IN THIS SUBROUTINE
C
C     INPUT PARAMETERS
C       P - VERTICAL DIMENSION OF GRID
C       Q - HORIZONTAL DIMENSION OF GRID
C       LL(I) - DISTANCE BETWEEN VERTICAL ROWS I,I-1
C       W(I) - DISTANCE BETWEEN HORIZONTAL COLUMNS I,I-1
C       RBASE - CONVENIENT REFERENCE BASE FOR ABSOLUTE ROUTE NUMBERS
C
C     ROUTE DESCRIPTION
C       NN(R) - NUMBER OF NODES ON ROUTE R
C       NODE(R,I) - THE I-TH NODE ON ROUTE R
C       D(R,I) - THE I-TH INTERSTOP DISTANCE ALONG ROUTE R
C
C     TRANSFER NODE DESCRIPTION
C       TNODE(L) - THE L-TH TRANSFER NODE
C       RT1(L) - ROUTE FROM WHICH TRANSFER IS MADE AT TNODE(L)
C       RT2(L) - ROUTE TO WHICH TRANSFER IS MADE AT TNODE(L)
C       NTRANS - NUMBER OF TRANSFERS
C
```

```
C--------
C
      IMPLICIT INTEGER (A-Y)
C
C  GENERATE FOR EACH ROUTE THE NODES ON IT.  ALSO CREATE INTERSTOP DISTANCES
C
      PP=2*P
      PPQ=PP+Q
      DO 110 R=1,P
      NN(R)=Q
      IS=Q*(R-1)
      NODE(R,1)=IS+1
      DO 100 I=2,Q
      NODE(R,I)=IS+I
      D(R,I-1)=W(I)
  100 CONTINUE
  110 CONTINUE
      DO 130 RR=1,P
      R=RR+P
      NN(R)=Q
      IF=Q*RR+1
      NODE(R,1)=IF-1
      DO 120 I=2,Q
      NODE(R,I)=IF-I
      D(R,I-1)=W(Q-I+2)
  120 CONTINUE
  130 CONTINUE
      DO 150 RR=1,Q
      R=RR+PP
      NN(R)=P
      NODE(R,1)=RR
      DO 140 I=2,P
      NODE(R,I)=RR+(I-1)*Q
      D(R,I-1)=LL(I)
  140 CONTINUE
  150 CONTINUE
      DO 170 RR=1,Q
      R=RR+PPQ
      NN(R)=P
      IF=(P-1)*Q+RR
      NODE(R,1)=IF
      DO 160 I=2,P
      NODE(R,I)=IF-(I-1)*Q
      D(R,I-1)=LL(P-I+2)
  160 CONTINUE
  170 CONTINUE
```

```
C
C  CREATE TRANSFER DATA.
C
      N=0
      L=0
      DO 240 I=1,P
      DO 230 J=1,Q
      N=N+1
      IF(I.EQ.1) GO TO 210
      IF(J.EQ.1) GO TO 200
      L=L+1
      TNODE(L)=N
      RT1(L)=PP+J
      RT2(L)=P+I
      L=L+1
      TNODE(L)=N
      RT1(L)=I
      RT2(L)=PPQ+J
  200 IF(J.EQ.Q) GO TO 210
      L=L+1
      TNODE(L)=N
      RT1(L)=PP+J
      RT2(L)=I
      L=L+1
      TNODE(L)=N
      RT1(L)=P+I
      RT2(L)=PPQ+J
  210 IF(J.EQ.1) GO TO 220
      IF(I.EQ.P) GO TO 220
      L=L+1
      TNODE(L)=N
      RT1(L)=I
      RT2(L)=PP+J
      L=L+1
      TNODE(L)=N
      RT1(L)=PPQ+J
      RT2(L)=P+I
  220 IF(J.EQ.Q) GO TO 230
      IF(I.EQ.P) GO TO 230
      L=L+1
      TNODE(L)=N
      RT1(L)=P+I
      RT2(L)=PP+J
      L=L+1
      TNODE(L)=N
      RT1(L)=PPQ+J
      RT2(L)=I
  230 CONTINUE
  240 CONTINUE
      NTRANS=L
      RETURN
      END
```

B.2.2 Subroutine XPRESS

```
      SUBROUTINE XPRESS(P,Q,LL,W,RBASE)
       PARAMETER  MP=25
       PARAMETER  MQ=25
       PARAMETER  NRTES=200
       PARAMETER  MAXN=50
       PARAMETER  TRANSF=5000
      DIMENSION  LL(1),W(1)
      DIMENSION  MAINP(MP),MAINQ(MQ),DP(MP),DQ(MQ),SP(MP),SQ(MQ),
     1  FLAG(12)
      COMMON  NN(NRTES),NODE(NRTES,MAXN),D(NRTES,MAXN),TNODE(TRANSF),
     1  RT1(TRANSF),RT2(TRANSF),NTRANS
      IMPLICIT INTEGER(A-Y)
      READ (5,900) NMP,NMQ
900   FORMAT (16I5)
      IF (NMP.LE.0.OR.NMQ.LE.0) RETURN
      READ (5,900) (MAINP(I),I=1,NMP)
      READ (5,900) (MAINQ(I),I=1,NMQ)
      L=1
      J=1
      IF (MAINP(1).EQ.1) GO TO 1
      SP(1)=1
      L=2
1     SP(L)=MAINP(J)
      L=L+1
      J=J+1
      IF (J.LE.NMP) GO TO 1
      IF (MAINP(NMP).EQ.P) GO TO 2
      SP(L)=P
      L=L+1
2     NSP=L-1
      L=1
      J=1
      IF (MAINQ(1).EQ.1) GO TO 3
      SQ(1)=1
      L=2
3     SQ(L)=MAINQ(J)
      L=L+1
      J=J+1
      IF (J.LE.NMQ) GO TO 3
      IF (MAINQ(NMQ).EQ.Q) GO TO 131
      SQ(L)=Q
      L=L+1
131   NSQ=L-1
      IEND=1
      DO 5 I=2,NSP
      II=I-1
      DP(II)=0
      IBEG=IEND+1
      IEND=SP(I)
```

```
      DO 4 J=IBEG,IEND
      DP(II)=DP(II)+LL(J)
4     CONTINUE
5     CONTINUE
      IEND=1
      DO 7 I=2,NSQ
      II=I-1
      DQ(II)=0
      IBEG=IEND+1
      IEND=SQ(I)
      DO 6 J=IBEG,IEND
      DQ(II)=DQ(II)+W(J)
6     CONTINUE
7     CONTINUE
      R1=2*(P+Q)
      Q1=NSQ+1
      DO 9 I=1,NMP
      R1=R1+1
      R2=R1+NMP
      BASE=(MAINP(I)-1)*Q
      DO 8 J=1,NSQ
      NOD=BASE+SQ(J)
      NODE(R1,J)=NOD
      NODE(R2,Q1-J)=NOD
      IF (J.EQ.NSQ) GO TO 8
      D(R1,J)=DQ(J)
      D(R2,NSQ-J)=DQ(J)
8     CONTINUE
      NN(R1)=NSQ
      NN(R2)=NSQ
9     CONTINUE
      R1=2*(P+Q+NMP)
      P1=NSP+1
      DO 11 I=1,NMQ
      R1=R1+1
      R2=R1+NMQ
      BASE=MAINQ(I)
      DO 10 J=1,NSP
      NOD=BASE+(SP(J)-1)*Q
      NODE(R1,J)=NOD
      NODE(R2,P1-J)=NOD
      IF (J.EQ.NSP) GO TO 10
      D(R1,J)=DP(J)
      D(R2,NSP-J)=DP(J)
10    CONTINUE
      NN(R1)=NSP
      NN(R2)=NSP
11     CONTINUE
```

```
      PQ=2*P+2*Q
      DO 38 L1=1,NMP
      I=MAINP(L1)
      DO 38 L2=1,NMQ
      J=MAINQ(L2)
      R=PQ+L1
      DO 12 L=1,12
      FLAG(L)=1
12    CONTINUE
      FLAG(9)=0
      FLAG(10)=0
      IF (J.NE.1) GO TO 13
      FLAG(1)=0
      FLAG(2)=0
      FLAG(3)=0
      FLAG(5)=0
      FLAG(7)=0
      FLAG(11)=0
      FLAG(12)=0
13     IF (J.NE.Q) GO TO 14
      FLAG(1)=0
      FLAG(2)=0
      FLAG(4)=0
      FLAG(6)=0
      FLAG(8)=0
14    IF (I.NE.1) GO TO 15
      FLAG(7)=0
      FLAG(6)=0
      FLAG(12)=0
15    IF (I.NE.P) GO TO 16
      FLAG(5)=0
      FLAG(8)=0
      FLAG(11)=0
16    CALL XTRANS(P,Q,FLAG,R,I,J,L1,L2,NMP,NMQ)
      IF (L2.GT.1) GO TO 18
      IF (SQ(1).EQ.MAINQ(1)) GO TO 17
      NOD=(I-1)*Q+1
      IF (I.EQ.1) GO TO 161
      NTRANS=NTRANS+1
      RT1(NTRANS)=2*P+1
      RT2(NTRANS)=R
      TNODE(NTRANS)=NOD
161   IF (I.EQ.P) GO TO 17
      NTRANS=NTRANS+1
      RT1(NTRANS)=2*P+Q+1
      RT2(NTRANS)=R
      TNODE(NTRANS)=NOD
17    IF (SQ(NSQ).EQ.MAINQ(NMQ)) GO TO 18
      NOD=I*Q
      IF (I.EQ.1) GO TO 171
```

```
      NTRANS=NTRANS+1
      RT1(NTRANS)=P
      RT2(NTRANS)=PQ
      TNODE(NTRANS)=NOD
171   IF (I.EQ.P) GO TO 18
      NTRANS=NTRANS+1
      RT1(NTRANS)=P
      RT2(NTRANS)=2*P+Q
      TNODE(NTRANS)=NOD
18    R=PQ+NMP+L1
      DO 19 L=1,12
      FLAG(L)=1
19    CONTINUE
      FLAG(9)=0
      FLAG(10)=0
      IF (J.NE.1) GO TO 20
      FLAG(2)=0
      FLAG(3)=0
      FLAG(4)=0
      FLAG(6)=0
      FLAG(8)=0
20    IF (J.NE.Q) GO TO 21
      FLAG(1)=0
      FLAG(3)=0
      FLAG(4)=0
      FLAG(5)=0
      FLAG(7)=0
      FLAG(11)=0
      FLAG(12)=0
21    IF (I.NE.1) GO TO 22
      FLAG(7)=0
      FLAG(6)=0
      FLAG(12)=0
22    IF (I.NE.P) GO TO 23
      FLAG(5)=0
      FLAG(8)=0
      FLAG(11)=0
23    CALL XTRANS(P,Q,FLAG,R,I,J,L1,L2,NMP,NMQ)
      IF (L2.GT.1) GO TO 25
      IF (SQ(NSQ).EQ.MAINQ(NMQ)) GO TO 24
      NOD=I*Q
      IF (I.EQ.1) GO TO 231
      NTRANS=NTRANS+1
      RT1(NTRANS)=2*P+Q
      RT2(NTRANS)=R
      TNODE(NTRANS)=NOD
231   IF (I.EQ.P) GO TO 24
      NTRANS=NTRANS+1
      RT1(NTRANS)=2*P+2*Q
      RT2(NTRANS)=R
      TNODE(NTRANS)=NOD
```

```
24    IF (SQ(1).EQ.MAINQ(1)) GO TO 25
      NOD=(J-1)*Q+1
      IF (I.EQ.1) GO TO 241
      NTRANS=NTRANS+1
      RT1(NTRANS)=R
      RT2(NTRANS)=2*P+Q+1
      TNODE(NTRANS)=NOD
241   IF (I.EQ.P) GO TO 25
      NTRANS=NTRANS+1
      RT1(NTRANS)=R
      RT2(NTRANS)=2*P+1
      TNODE(NTRANS)=NOD
25    R=PQ+2*NMP+L2
      DO 26 L=1,12
      FLAG(L)=1
26    CONTINUE
      FLAG(11)=0
      FLAG(12)=0
      IF (J.NE.1) GO TO 27
      FLAG(2)=0
      FLAG(3)=0
      FLAG(10)=0
27    IF (J.NE.Q) GO TO 28
      FLAG(1)=0
      FLAG(4)=0
      FLAG(9)=0
28    IF (I.NE.1) GO TO 29
      FLAG(1)=0
      FLAG(3)=0
      FLAG(5)=0
      FLAG(6)=0
      FLAG(7)=0
      FLAG(9)=0
      FLAG(10)=0
29    IF (I.NE.P) GO TO 30
      FLAG(2)=0
      FLAG(4)=0
      FLAG(5)=0
      FLAG(6)=0
      FLAG(8)=0
30    CALL XTRANS(P,Q,FLAG,R,I,J,L1,L2,NMP,NMQ)
      IF (L1.GT.1) GO TO 32
      IF (SP(1).EQ.MAINP(1)) GO TO 31
      NOD=J
      IF (J.EQ.1) GO TO 301
      NTRANS=NTRANS+1
      RT1(NTRANS)=1
      RT2(NTRANS)=R
      TNODE(NTRANS)=NOD
```

```
301   IF (J.EQ.Q) GO TO 31
      NTRANS=NTRANS+1
      RT1(NTRANS)=P+1
      RT2(NTRANS)=P
      TNODE(NTRANS)=NOD
31    IF (SP(NSP).EQ.MAINP(NMP)) GO TO 32
      NOD=(P-1)*Q+J
      IF (J.EQ.1) GO TO 311
      NTRANS=NTRANS+1
      RT1(NTRANS)=P
      RT2(NTRANS)=2*P
      TNODE(NTRANS)=NOD
311   IF (J.EQ.Q) GO TO 32
      NTRANS=NTRANS+1
      RT1(NTRANS)=P
      RT2(NTRANS)=P
      TNODE(NTRANS)=NOD
32    R=PQ+2*NMP+NMQ+L2
      DO 321 L=1,12
      FLAG(L)=1
321   CONTINUE
      FLAG(11)=0
      FLAG(12)=0
      IF (J.NE.1) GO TO 33
      FLAG(2)=0
      FLAG(3)=0
      FLAG(10)=0
33    IF (J.NE.Q) GO TO 34
      FLAG(1)=0
      FLAG(4)=0
      FLAG(9)=0
34    IF (I.NE.1) GO TO 35
      FLAG(2)=0
      FLAG(4)=0
      FLAG(6)=0
      FLAG(7)=0
      FLAG(8)=0
35    IF (I.NE.P) GO TO 36
      FLAG(1)=0
      FLAG(3)=0
      FLAG(5)=0
      FLAG(7)=0
      FLAG(8)=0
      FLAG(9)=0
      FLAG(10)=0
36    CALL XTRANS(P,Q,FLAG,R,I,J,L1,L2,NMP,NMQ)
      IF (L1.GT.1) GO TO 38
      IF (SP(NSP).EQ.MAINP(NMP)) GO TO 37
      NOD=(P-1)*Q+J
```

```
      IF (J.EQ.1) GO TO 361
      NTRANS=NTRANS+1
      RT1(NTRANS)=P
      RT2(NTRANS)=R
      TNODE(NTRANS)=NOD
361   IF (J.EQ.Q) GO TO 37
      NTRANS=NTRANS+1
      RT1(NTRANS)=2*P
      RT2(NTRANS)=R
      TNODE(NTRANS)=NOD
37    IF (SP(1).EQ.MAINP(1)) GO TO 38
      NOD=J
      IF (J.EQ.1) GO TO 371
      NTRANS=NTRANS+1
      RT1(NTRANS)=R
      RT2(NTRANS)=P+1
      TNODE(NTRANS)=NOD
371   IF (J.EQ.Q) GO TO 38
      NTRANS=NTRANS+1
      RT1(NTRANS)=R
      RT2(NTRANS)=1
      TNODE(NTRANS)=NOD
38    CONTINUE
      RETURN
      END
```

B.2.3 Subroutine XTRANS

```
      SUBROUTINE XTRANS(P,Q,FLAG,R,I,J,L1,L2,NMP,NMQ)
      PARAMETER  NRTES=200
      PARAMETER  MAXN=50
      PARAMETER  TRANSF=5000
      COMMON  NN(NRTES),NODE(NRTES,MAXN),D(NRTES,MAXN),TNODE(TRANSF),
     1  RT1(TRANSF),RT2(TRANSF),NTRANS
      IMPLICIT INTEGER (A-Y)
      DIMENSION  FLAG(1)
       NOD=(I-1)*Q+J
      PQ=2*P+2*Q
      LL=NTRANS
      IF (FLAG(1).EQ.0) GO TO 1
      LL=LL+1
      RT1(LL)=R
      RT2(LL)=I
1     IF (FLAG(2).EQ.0) GO TO 2
      LL=LL+1
      RT1(LL)=I
      RT2(LL)=R
2     IF (FLAG(3).EQ.0) GO TO 3
      LL=LL+1
      RT1(LL)=R
      RT2(LL)=P+I
3     IF (FLAG(4).EQ.0) GO TO 4
      LL=LL+1
      RT1(LL)=P+I
      RT2(LL)=R
4     IF (FLAG(5).EQ.0) GO TO 5
      LL=LL+1
      RT1(LL)=R
      RT2(LL)=2*P+J
5     IF (FLAG(6).EQ.0) GO TO 6
      LL=LL+1
      RT1(LL)=2*P+J
      RT2(LL)=R
6     IF (FLAG(7).EQ.0) GO TO 7
      LL=LL+1
      RT1(LL)=R
      RT2(LL)=2*P+Q+J
7     IF (FLAG(8).EQ.0) GO TO 8
      LL=LL+1
      RT1(LL)=2*P+Q+J
      RT2(LL)=R
8     IF (FLAG(9).EQ.0) GO TO 9
      LL=LL+1
      RT1(LL)=R
      RT2(LL)=PQ+L1
9     IF (FLAG(10).EQ.0) GO TO 10
      LL=LL+1
      RT1(LL)=R
      RT2(LL)=PQ+NMP+L1
```

```
10    IF (FLAG(11).EQ.0) GO TO 11
      LL=LL+1
      RT1(LL)=R
      RT2(LL)=PQ+2*NMP+L2
11    IF (FLAG(12).EQ.0) GO TO 12
      LL=LL+1
      RT1(LL)=R
      RT2(LL)=PQ+2*NMP+NMQ+L2
12    LBEG=NTRANS+1
      DO 13 L=LBEG,LL
      TNODE(L)=NOD
13    CONTINUE
      NTRANS=LL
      RETURN
      END
```

B.2.4 Subroutine XSCHED

```
      SUBROUTINE XSCHED(NN,NODE,D,ZK,XZK,PDS,RBASE,PQ)
C  THIS SUBROUTINE READS IN A GROUP OF ROUTES TOGETHER WITH
C  ABBREVIATED SCHEDULE INFORMATION AND PRODUCES COMPLETE SCHEDULE
C  INFORMATION FOR ROUTES IN EACH PERIOD.  THE VARIABLES PDS,RBASE
C  AND THE ARRAYS NN,NODE,D,ZK ARE TRANSMITTED FROM THE MAIN PROGRAM.
C  THE PROGRAM WRITES OUT THE DETAILED SCHEDULE INFORMATION USING
C  UNIT OUT = 7.
C
C--------
C
C  NATIONAL BUREAU OF STANDARDS      MAY,1976
C
C--------
C
      PARAMETER MAXPD=10        @ NUMBER OF PERIODS
      PARAMETER NRTES=200       @ NUMBER OF ROUTES
      PARAMETER MAXN=50         @ NUMBER OF NODES/ROUTE
      PARAMETER NGROUP=100      @ NUMBER OF ROUTES/GROUP
      DIMENSION NN(NRTES),NODE(NRTES,MAXN),D(NRTES,MAXN),ZK(MAXPD),
     1          ROUTE(NGROUP),RUNS(MAXPD),HEAD(MAXPD),
     2          DTIME(MAXPD,NGROUP),SCHED(MAXN),XZK(MAXPD)
C
C--------
C
C  VARIABLES AND ARRAYS USED IN THIS SUBROUTINE
C
C    INPUT PARAMETERS
C      PDS - NUMBER OF PERIODS
C      RBASE - CONVENIENT REFERENCE BASE FOR ABSOLUTE ROUTE NUMBERS
C      ZK(J) - CONVERTS DISTANCE INTO TIME FOR PERIOD J
C
C    ROUTE DESCRIPTION
C      NN(R) - NUMBER OF NODES ON ROUTE R
C      NODE(R,I) - THE I-TH NODE ON ROUTE R
C      D(R,I) - THE I-TH INTERSTOP DISTANCE ALONG ROUTE R
C
C    ADDITIONAL VARIABLES AND ARRAYS (FROM UNIT 5)
C      NR - NUMBER OF ROUTES IN A GROUP
C      ROUTE(J) - THE J-TH ROUTE OF THE GROUP
C      RUNS(I) - NUMBER OF RUNS FOR PERIOD I
C      HEAD(I) - HEADWAY FOR ROUTES IN PERIOD I
C      DTIME(I,J) - INITIAL DEPARTURE TIME FOR ROUTE J IN PERIOD I
C
C    WORKING ARRAYS
C      SCHED(K) - SCHEDULE TIME FOR K-TH NODE ALONG ROUTE
C
```

```
C--------
C
      IMPLICIT INTEGER (A-Y)
      REAL XZK
C
C  DEFINE OUTPUT UNIT.
C
      OUT=7
C
C  READ IN A GROUP OF ROUTES (SIMILAR IN RUNS,HEADWAYS).
C
   10 READ(5,900,END=80) NR,(ROUTE(J),J=1,NR)
  900 FORMAT(16I5)
C
C  FOR EACH ROUTE IN THE GROUP, READ IN THE SCHEDULE PARAMETERS BY TIME
C  PERIOD.
C
      NRUNS=0
      DO 20 I=1,PDS
      READ(5,900) RUNS(I),HEAD(I),(DTIME(I,J),J=1,NR)
      NRUNS=NRUNS+RUNS(I)
   20 CONTINUE
C
C  BEGIN CONSIDERATION OF EACH ROUTE IN THE GROUP.
C
      DO 70 J=1,NR
      R=ROUTE(J)
      NNR=NN(R)
      RT=R+RBASE
C
C  PRINT ROUTES AND NODES ON ROUTES.
C
      WRITE(OUT,800)RT,NNR,NRUNS,(NODE(R,I),I=1,NNR)
  800 FORMAT(20I5)
C
C  COMPUTE SCHEDULE INFORMATION.
C
      DO 60 II=1,PDS
      SCHED(1)=DTIME(II,J)
      Z=ZK(II)
      IF (R.GT.PQ) Z=XZK(II)
      HD=HEAD(II)
      RNS=RUNS(II)-1
      DO 30 I=2,NNR
      SCHED(I)=SCHED(I-1)+(Z*D(R,I-1))
   30 CONTINUE
```

```
C
C  PRINT SCHEDULE INFORMATION FOR FIRST DEPARTURE IN PERIOD.
C
      WRITE(OUT,800) (SCHED(I),I=1,NNR)
      IF(RNS.EQ.0) GO TO 60
      DO 50 K=1,RNS
      DO 40 I=1,NNR
      SCHED(I)=SCHED(I)+HD
   40 CONTINUE
C
C  PRINT SCHEDULE INFORMATION FOR REST OF PERIOD.
C
      WRITE(OUT,800) (SCHED(I),I=1,NNR)
   50 CONTINUE
   60 CONTINUE
   70 CONTINUE
      GO TO 10
   80 ENDFILE OUT
      RETURN
      END
```

B.2.5 Subroutine TRANS

```
      SUBROUTINE TRANS(NTRANS,TNODE,RT1,RT2,PQ,RBASE)
C  THIS SUBROUTINE WRITES OUT USING UNIT OUT = 8 THE TRANSFER
C  INFORMATION PREVIOUSLY GENERATED BY GRID. THE ROUTE NUMBERS
C  PRINTED ARE ABSOLUTE (I.E. NUMBERS RUN CONSECUTIVELY FROM RBASE).
C  THE VARIABLES PQ,RBASE,NTRANS AND THE ARRAYS TNODE,RT1,RT2
C  ARE TRANSMITTED FROM THE MAIN PROGRAM. THE ARRAY TMIN IS READ FROM
C  UNIT IN = 12.
C
C--------
C
C  NATIONAL BUREAU OF STANDARDS        MAY,1976
C
C--------
C
      PARAMETER TRANSF=5000     @ NUMBER OF TRANSFERS
      PARAMETER MNODE=1000      @ NUMBER OF NODES
      DIMENSION TNODE(TRANSF),RT1(TRANSF),RT2(TRANSF),TMIN(MNODE)
C
C--------
C
C  VARIABLES AND ARRAYS USED IN THIS SUBROUTINE
C
C    INPUT PARAMETERS
C      RBASE - CONVENIENT REFERENCE BASE FOR ABSOLUTE ROUTE NUMBERS
C      TMIN(I) - MINIMUM TRANSFER TIME BETWEEN ANY TWO ROUTES AT NODE I
C
C    TRANSFER NODE DESCRIPTION
C      TNODE(L) - THE L-TH TRANSFER NODE
C      RT1(L) - ROUTE FROM WHICH TRANSFER IS MADE AT TNODE(L)
C      RT2(L) - ROUTE TO WHICH TRANSFER IS MADE AT TNODE(L)
C      NTRANS - NUMBER OF TRANSFERS
C
C--------
C
      IMPLICIT INTEGER (A-Y)
C
C  DEFINE INPUT AND OUTPUT UNITS.
C
      IN=12
      OUT=8
C
C  READ IN MINIMUM TRANSFER TIMES AT EACH NODE.
C
      READ(IN,900) (TMIN(I),I=1,PQ)
  900 FORMAT(16I5)
```

```
C
C  PRINT TRANSFER MODE INFORMATION.
C
      DO 10 L=1,NTRANS
      R1=RT1(L)+RBASE
      R2=RT2(L)+RBASE
      TN=TNODE(L)
      WRITE(OUT,800) TN,R1,R2,TMIN(TN)
  800 FORMAT(4I5)
   10 CONTINUE
      ENDFILE OUT
      RETURN
      END
```

B.3 PROGRAM TRA

```
      PARAMETER  MNODE=200
      PARAMETER  MM=20
      PARAMETER  MN=50
      DIMENSION  RTE(MNODE,MN),STP(MM),MINTRA(MNODE),NSTP(MNODE),
     1           BE(MNODE,MN)
      IMPLICIT INTEGER (A-Z)
      DO 1 I=1,MNODE
      NSTP(I)=0
      DO 1 J=1,MN
      BE(I,J)=0
1     CONTINUE
      READ (5,900) NODE
900   FORMAT (20I5)
      READ (12,903) (MINTRA(I),I=1,NODE)
903   FORMAT (16I5)
2     READ (7,900,END=6) R,M,NRUNS,(STP(I),I=1,M)
      DO 4 I=1,M
      J=STP(I)
      N=NSTP(J)+1
      RTE(J,N)=R
      NSTP(J)=N
      IF (I.GT.1) GO TO 3
      BE(J,N)=1
      GO TO 4
3     IF (I.LT.M) GO TO 4
      BE(J,N)=2
4     CONTINUE
      DO 5 L=1,NRUNS
      READ (7,901) DUMMY
901   FORMAT (A6)
5     CONTINUE
      GO TO 2
    6 CONTINUE
      DO 10 L=1,NODE
      N=NSTP(L)
      DO 10 II=1,N
      IF (BE(L,II).EQ.1) GO TO 10
      I=RTE(L,II)
      J1=II-1
      IF (MOD(II,2).EQ.0) J1=II-2
      IF (J1.LT.1) GO TO 8
      DO 7 JJ=1,J1
      IF (BE(L,JJ).EQ.2) GO TO 7
      J=RTE(L,JJ)
      WRITE (8,900) L,I,J,MINTRA(L)
7     CONTINUE
```

```
8       J1=II+1
        IF (MOD(II,2).EQ.1) J1=II+2
        IF (J1.GT.N) GO TO 10
        DO 9 JJ=J1,N
        IF (RE(L,JJ).EQ.2) GO TO 9
        J=RTE(L,JJ)
        WRITE (8,900) L,I,J,MINTRA(L)
9       CONTINUE
10      CONTINUE
        ENDFILE 8
        STOP
        END
```

```
C  ****************** ACYLE : ACYCLIC TRANSLATOR *****************
C  THIS PROGRAM PRODUCES AN APPROPRIATE TIME-EXPANDED NETWORK FROM
C  GIVEN SCHEDULE INFORMATION AND TRANSFER DATA.  EACH NODE OF THE
C  TIME-EXPANDED NETWORK REPRESENTS A PARTICULAR (STOP,TIME) PAIR.
C  TRANSFERS ARE ACCOMMODATED USING TRANSFER ARCS WITH THE FICTITIOUS
C  ROUTE 9999.
C
C
C  ----------
C
C  NATIONAL BUREAU OF STANDARDS        JULY, 1976
C
C  ----------
C
      COMPILER (XM=1)
      PARAMETER MAXS=100      @ NUMBER OF STOPS/ROUTE
      PARAMETER NRTES=200     @ NUMBER OF ROUTES
      PARAMETER MAXN=4500     @ NUMBER OF NODES
      PARAMETER MAXA=13000    @ NUMBER OF ARCS
      DIMENSION NODE(MAXS),SCHED(MAXS),START(NRTES),END(NRTES)
      COMMON /BLK1/ N(MAXN),T(MAXN),TT(MAXN),TIND(MAXN),NEW(MAXN)
      COMMON /BLK2/ FROM(MAXA),TO(MAXA),RTE(MAXA),FF(MAXA),FIND(MAXA)
C
C  ----------
C
C  VARIABLES AND ARRAYS USED IN THIS PROGRAM
C
C    INPUT VARIABLES AND ARRAYS
C
C      RT - ROUTE NUMBER
C      NN - NUMBER OF STOPS ON ROUTE
C      RUNS - NUMBER OF RUNS
C      NODE(I) - THE I-TH STOP ALONG THE ROUTE
C      SCHED(I) - THE I-TH SCHEDULE TIME ALONG THE ROUTE
C      TNODE - STOP AT WHICH TRANSFER OCCURS
C      RT1 - ROUTE FROM WHICH TRANSFER AT TNODE
C      RT2 - ROUTE TO WHICH TRANSFER AT TNODE
C      TMIN - MINIMUM TRANSFER TIME AT TNODE
C
C    CONSTRUCTED ARRAYS
C
C      N(I) - STOP ASSOCIATED WITH NETWORK NODE I
C      T(I) - TIME ASSOCIATED WITH NETWORK NODE I
C      START(R) - FIRST POSITION WHERE INFORMATION MAY BE FOUND
C                 FOR ROUTE R ON NODE LIST
C      END(R) - LAST POSITION WHERE INFORMATION MAY BE FOUND
C               FOR ROUTE R ON NODE LIST
```

```
C       LLEN - LENGTH OF NETWORK NODE LIST
C       FROM(J) - STARTING NODE OF ARC IN POSITION J OF ARC LIST
C       TO(J) - ENDING NODE OF ARC IN POSITION J OF ARC LIST
C       RTE(J) - ROUTE NUMBER CORRESPONDING TO ARC IN POSITION J
C                OF ARC LIST
C       MLEN - LENGTH OF NETWORK ARC LIST
C
C     WORKING ARRAYS
C
C       TT(I) - THE I-TH ORDERED ELEMENT OF T
C       TIND(I) -  THE POSITION IN T OF THE I-TH ELEMENT OF TT
C       NEW(I) - THE POSITION IN TT OF THE I-TH ELEMENT OF T
C       FF(I) - THE I-TH ORDERED ELEMENT OF FROM
C       FIND(I) - THE POSITION IN FROM OF THE I-TH ELEMENT OF FF
C
C   ----------
C
      IMPLICIT INTEGER (A-Z)
C
C  DEFINE INPUT UNITS.
C
      IN1=7    @ INPUT FILE FOR SCHEDULE DATA
      IN2=8    @ INPUT FILE FOR TRANSFER DATA
C
C  BEGIN PROCESSING SCHEDULE INFORMATION.  CREATE NODES AND ARCS
C  OF THE TIME-EXPANDED NETWORK.
C
      L=0
      M=0
C
C  READ IN ROUTE AND SCHEDULE DESCRIPTION. CONSTRUCT NODE AND ARC LIST.
C  NOTE THAT ONE MUST HAVE NN > 1 AND RUNS > 0.
C
   10 READ(IN1,900,END=40) RT,NN,RUNS,(NODE(I),I=1,NN)
  900 FORMAT(20I5)
      START(RT)=L+1
      DO 30 JJ=1,RUNS
      READ(IN1,900) (SCHED(I),I=1,NN)
      L=L+1
      N(L)=NODE(1)
      T(L)=SCHED(1)
      DO 20 J=2,NN
      M=M+1
      FROM(M)=L
      TO(M)=L+1
      RTE(M)=RT
      L=L+1
      N(L)=NODE(J)
      T(L)=SCHED(J)
   20 CONTINUE
   30 CONTINUE
```

```
      END(RT)=L
      GO TO 10
   40 LLEN=L
C
C  READ IN TRANSFER DATA AND UPDATE ARC LIST.
C
   50 READ(IN2,900,END=80) TNODE,RT1,RT2,TMIN
      LS1=START(RT1)
      LF1=END(RT1)
      DO 70 L=LS1,LF1
      IF(N(L).NE.TNODE) GO TO 70
      TM=T(L)+TMIN
      LS2=START(RT2)
      LF2=END(RT2)
      DO 60 LL=LS2,LF2
      IF(N(LL).NE.TNODE) GO TO 60
      IF(T(LL).LT.TM) GO TO 60
      IF(LL.EQ.LF2)GO TO 55
      IF(N(LL).EQ.N(LL+1))GO TO 60
   55 M=M+1
      FROM(M)=L
      TO(M)=LL
      RTE(M)=9999
      GO TO 70
   60 CONTINUE
   70 CONTINUE
      MLEN=M
      GO TO 50
C
C  SORT NODE ARRAY BY TIME.
C
   80 CALL SORTP(T,LLEN,TT,TIND)
      DO 90 I=1,LLEN
      J=TIND(I)
      NEW(J)=I
   90 CONTINUE
C
C  RENUMBER NODES IN ARC LIST.
C
      DO 100 I=1,MLEN
      K1=FROM(I)
      K2=TO(I)
      FROM(I)=NEW(K1)
      TO(I)=NEW(K2)
  100 CONTINUE
C
C  SORT ARC ARRAY BY ORIGIN NODE.
C
      CALL SORTP(FROM,MLEN,FF,FIND)
```

```
C
C  WRITE OUT NETWORK NODE DATA, SORTED ON T.
C
      OUT1=9       @ DEFINE OUTPUT UNIT FOR NODE DATA.
      DO 110 I=1,LLEN
      J=TIND(I)
      WRITE(OUT1,901) N(J),T(J)
  901 FORMAT(3I5)
  110 CONTINUE
      ENDFILE OUT1
C
C  WRITE OUT ARC DATA, SORTED BY ORIGIN NODE.
C
      OUT2=10      @ DEFINE OUTPUT UNIT FOR ARC DATA.
      DO 120 I=1,MLEN
      J=FIND(I)
      WRITE(OUT2,901) FROM(J),TO(J),RTE(J)
  120 CONTINUE
      ENDFILE OUT2
      STOP
      END
```

8.4.1 Subroutine SORTP

```
SUBROUTINE SORTP(X,N,Y,XPOS)
THIS ROUTINE SORTS THE ELEMENTS OF THE INPUT VECTOR X AND PUTS THE SORTED
ELEMENTS INTO THE VECTOR Y.  IT ALSO CARRIES ALONG THE INDEX NUMBER
OF EACH ORDERED OBSERVATION--THAT IS, IT CARRIES ALONG THE POSITION OF
THE I-TH ORDERED OBSERVATION (FOR EACH I) AS IT WAS IN THE ORIGINAL
UNORDERED DATA VECTOR X.  THESE POSITIONS ARE PLACED IN THE VECTOR XPOS.
THIS ROUTINE IS USEFUL IN ATTEMPTING TO LOCATE THE MINIMUM, THE MAXIMUM,
OR SOME OTHER ORDERED OBSERVATION OF INTEREST IN THE ORIGINAL UNORDERED
INPUT VECTOR X.
THE INPUT TO THIS ROUTINE IS THE SINGLE PRECISION VECTOR X OF
(UNSORTED) OBSERVATIONS, THE INTEGER VALUE N (= SAMPLE SIZE),
AN EMPTY SINGLE PRECISION VECTOR Y INTO WHICH THE SORTED OBSERVATIONS
WILL BE PLACED, AND AN EMPTY SINGLE PRECISION VECTOR XPOS INTO WHICH THE
POSITIONS OF THE SORTED OBSERVATIONS WILL BE PLACED.
THE OUTPUT FROM THIS ROUTINE IS THE SINGLE PRECISION VECTOR Y INTO WHICH
THE SORTED OBSERVATIONS HAVE BEEN PLACED, AND THE SINGLE PRECISION VECTOR
XPOS INTO WHICH THE POSITIONS OF THE SORTED OBSERVATIONS HAVE BEEN PLACED.
RESTRICTIONS ON THE MAXIMUM ALLOWABLE VALUE OF N--THE DIMENSIONS
OF VECTORS IU AND IL (DEFINED AND USED INTERNALLY WITHIN THIS ROUTINE)
DETERMINE THE MAXIMUM ALLOWABLE VALUE OF N FOR THIS
ROUTINE.  IF IU AND IL EACH HAVE DIMENSION K, THEN N MAY NOT EXCEED
2**(K+1) - 1.  FOR THIS ROUTINE AS WRITTEN, THE DIMENSIONS OF IU AND IL
HAVE BEEN SET TO 36, THUS THE MAXIMUM ALLOWABLE VALUE OF N IS
APPROXIMATELY 137 BILLION.  SINCE THIS EXCEEDS THE MAXIMUM ALLOWABLE
VALUE FOR AN INTEGER VARIABLE IN MANY COMPUTERS, AND SINCE A SORT OF 137
BILLION ELEMENTS IS PRESENTLY IMPRACTICAL AND UNLIKELY, THEREFORE NO
TEST FOR WHETHER THE INPUT SAMPLE SIZE N EXCEEDS 137 BILLION HAS BEEN
INCORPORATED INTO THIS ROUTINE.  IT IS THUS ASSUMED THAT THERE IS NO
(PRACTICAL) RESTRICTION ON THE MAXIMUM VALUE OF N FOR THIS ROUTINE.
PRINTING--NONE UNLESS AN ERROR CONDITION EXISTS
THIS ROUTINE IS SINGLE PRECISION IN INTERNAL OPERATION.
SUBROUTINES NEEDED--NONE
SORTING METHOD--BINARY SORT
REFERENCE--CACM MARCH 1969, PAGE 186 (BINARY SORT ALGORITHM BY RICHARD
           C. SINGLETON.
         --CACM JANUARY 1970, PAGE 54.
         --CACM OCTOBER 1970, PAGE 624.
         --JACM JANUARY 1961, PAGE 41.
WRITTEN BY JAMES J. FILLIBEN, STATISTICAL ENGINEERING LABORATORY (205.03)
NATIONAL BUREAU OF STANDARDS, WASHINGTON, D.C. 20234       JUNE 1972

DIMENSION X(1),Y(1),XPOS(1)
DIMENSION IU(36),IL(36)
IMPLICIT INTEGER (A-Z)

CHECK THE INPUT ARGUMENTS FOR ERRORS
```

```
      IPR=6
      IF(N.LT.1)GOTO50
      IF(N.EQ.1)GOTO55
      HOLD=X(1)
      DO60I=2,N
      IF(X(I).NE.HOLD)GOTO90
   60 CONTINUE
      WRITE(IPR, 9)HOLD
      DO61I=1,N
      Y(I)=X(I)
      XPOS(I)=I
   61 CONTINUE
      RETURN
   50 WRITE(IPR,15)
      WRITE(IPR,47)N
      RETURN
   55 WRITE(IPR,18)
      Y(1)=X(1)
      XPOS(1)=1.0
      RETURN
   90 CONTINUE
    9 FORMAT(1H ,108H***** NON-FATAL DIAGNOSTIC--THE FIRST  INPUT ARGUME
     1NT (A VECTOR) TO THE SORTP  SUBROUTINE HAS ALL ELEMENTS = ,E15.8,6
     1H *****)
   15 FORMAT(1H , 91H***** FATAL ERROR--THE SECOND INPUT ARGUMENT TO THE
     1 SORTP  SUBROUTINE IS NON-POSITIVE *****)
   18 FORMAT(1H ,100H***** NON-FATAL DIAGNOSTIC--THE SECOND INPUT ARGUME
     1NT TO THE SORTP  SUBROUTINE HAS THE VALUE 1 *****)
   47 FORMAT(1H , 35H***** THE VALUE OF THE ARGUMENT IS ,I8   ,6H *****)
C
C     COPY THE VECTOR X INTO THE VECTOR Y
      DO100I=1,N
      Y(I)=X(I)
  100 CONTINUE
C
C     DEFINE THE XPOS (POSITION) VECTOR.  BEFORE SORTING, THIS WILL
C     BE A VECTOR WHOSE I-TH ELEMENT IS EQUAL TO I.
C
      DO150I=1,N
      XPOS(I)=I
  150 CONTINUE
C
C     CHECK TO SEE IF THE INPUT VECTOR IS ALREADY SORTED
C
      NM1=N-1
      DO200I=1,NM1
      IP1=I+1
      IF(Y(I).LE.Y(IP1))GOTO200
      GOTO250
  200 CONTINUE
      RETURN
```

```
250 M=1
    I=1
    J=N
305 IF(I.GE.J)GOTO370
310 K=I
    MID=(I+J)/2
    AMED=Y(MID)
    BMED=XPOS(MID)
    IF(Y(I).LE.AMED)GOTO320
    Y(MID)=Y(I)
    XPOS(MID)=XPOS(I)
    Y(I)=AMED
    XPOS(I)=BMED
    AMED=Y(MID)
    BMED=XPOS(MID)
320 L=J
    IF(Y(J).GE.AMED)GOTO340
    Y(MID)=Y(J)
    XPOS(MID)=XPOS(J)
    Y(J)=AMED
    XPOS(J)=BMED
    AMED=Y(MID)
    BMED=XPOS(MID)
    IF(Y(I).LE.AMED)GOTO340
    Y(MID)=Y(I)
    XPOS(MID)=XPOS(I)
    Y(I)=AMED
    XPOS(I)=BMED
    AMED=Y(MID)
    BMED=XPOS(MID)
    GOTO340
330 Y(L)=Y(K)
    XPOS(L)=XPOS(K)
    Y(K)=TT
    XPOS(K)=ITT
340 L=L-1
    IF(Y(L).GT.AMED)GOTO340
    TT=Y(L)
    ITT=XPOS(L)
350 K=K+1
    IF(Y(K).LT.AMED)GOTO350
    IF(K.LE.L)GOTO330
    LMI=L-I
    JMK=J-K
    IF(LMI.LE.JMK)GOTO360
    IL(M)=I
    IU(M)=L
    I=K
    M=M+1
    GOTO380
```

```
360 IL(M)=K
    IU(M)=J
    J=L
    M=M+1
    GOTO380
370 M=M-1
    IF(M.EQ.0)RETURN
    I=IL(M)
    J=IU(M)
380 JMI=J-I
    IF(JMI.GE.11)GOTO310
    IF(I.EQ.1)GOTO305
    I=I-1
390 I=I+1
    IF(I.EQ.J)GOTO370
    AMED=Y(I+1)
    BMED=XPOS(I+1)
    IF(Y(I).LE.AMED)GOTO390
    K=I
395 Y(K+1)=Y(K)
    XPOS(K+1)=XPOS(K)
    K=K-1
    IF(AMED.LT.Y(K))GOTO395
    Y(K+1)=AMED
    XPOS(K+1)=BMED
    GOTO390
    END
```

B.5 PROGRAM LABCOR

```
      PARAMETER  MR=100          @ MAX NUMBER OF ROUTES
      PARAMETER  MMR=MR+1
      PARAMETER  MS=120          @ MAX NUMBER OF STOPS
      PARAMETER  ML=MS           @ MAX POSITIONS IN L
      PARAMETER  MSR=12          @ MAX NUMBER OF STOPS PER ROUTE
      PARAMETER  MRS=20          @ MAX NUMBER OF ROUTES PER STOP
      PARAMETER  MD=3000         @ MAX NUMBER OF VEHICLE DEPARTURES
      PARAMETER  MP=6            @ MAX NUMBER OF SEGMENTS PER PATH
      PARAMETER  MMP=MP+1
      IMPLICIT INTEGER (A-Z)
      REAL ROUT
      DIMENSION  T(MS),TB(MS),PS(MS),PR(MS),L(ML),F(MS),NR(MS),
     1           NS(MR),SBEG(MMR),STOP(MR,MSR),MINTRA(MS),
     2           ROUTE(MS,MRS),SCHED(MD,MSR),SPPT(MMP),RPRT(MP),
     3           TPRT(MP),TBPRT(MP),SEND(MMR)
C
C ARRAYS USED IN THIS PROGRAM
C
C    STOP INPUT
C       NR(S)       - NUMBER OF ROUTES STOPPING AT S
C       ROUTE(S,J)  - JTH ROUTE STOPPING AT S
C       MINTRA(S)   - MINIMUM TIME REQUIRED TO TRANSFER BETWEEN ROUTES
C                     AT NODE S
C    ROUTE INPUT
C       NS(R)       - NUMBER OF STOPS ON ROUTE R
C       STOP(R,I)   - ITH STOP ON ROUTE R
C       SBEG(R)     - LOCATION IN THE SCHEDULE LIST OF THE FIRST
C                     SCHEDULED DEPARTURE FOR ROUTE R
C       SEND(R)     - LOCATION IN THE SCHEDULE LIST OF THE LAST
C                     SCHEDULED DEPARTURE FOR ROUTE R
C       SCHED(K,I)  - ARRIVAL TIME AT THE ITH STOP OF THE KTH DEPARTURE
C    ALGORITHM
C       L(S)        - SEQUENCE LIST OF STOPS TO FAN OUT FROM
C       F(S)        - POSITION OF STOP S IN LIST L
C       T(S)        - ARRIVAL TIME AT STOP S
C       TB(S)       - BOARDING TIME FOR VEHICLE ARRIVING AT S
C       PS(S)       - STOP PRECEDING S IN PATH TO S
C       PR(S)       - ROUTE FROM PS(S) TO S
C    PRINTING THE PATH
C       SPRT(J)     - STOP
C       RPRT(J)     - ROUTE
C       TPRT(J)     - ARRIVAL TIME
C       TBPRT(J)    - BOARDING TIME
C
      INF=99999999               @ INFINITY USED IN THE PATH CALCULATION
      RTSO=0
      RTIME=0
      NRUN=0
```

```
C
      DO 1 S=1,MS
      NR(S)=0
1     CONTINUE
C
C READ ROUTE INPUT
C
      NSTOP=0
      NROUTE=0
      K=1         @ NUMBER OF DEPARTURES IN THE SCHEDULE
2     READ (7,901,END=6) R,M,KK,(STOP(R,J),J=1,M)
901   FORMAT (20I5)
      NS(R)=M
      IF (R.GT.NROUTE) NROUTE=R
C
C READ SCHEDULES FOR ROUTE R
C
      SBEG(R)=K
      DO 3 KS=1,KK
      READ (7,901) (SCHED(K,J), J=1,M)
      K=K+1
3     CONTINUE
      SEND(R)=K-1
C
C ADD ROUTE R TO LIST OF ROUTES STOPPING AT EACH STOP IN R'S STOP LIST
C
4     DO 5 I=1,M
      S=STOP(R,I)
      IF (S.GT.NSTOP) NSTOP=S
      NR(S)=NR(S)+1
      J=NR(S)
      ROUTE(S,J)=R
5     CONTINUE
      GO TO 2
6     READ (12,907) (MINTRA(I),I=1,NSTOP)
907   FORMAT (16I5)
C
C READ ORIGIN AND DESTINATION
C
7     READ (11,903,END=24) ORG, DST,TIME
903   FORMAT( 3I5)
      WRITE (6,904)  ORG,DST,TIME
904   FORMAT ('0'///'0TRIP FROM',I5,' TO',I5,' DEPARTING ON OR AFTER',
     1I5/)
      IF (TIME.GE.1.AND.TIME.LE.1440) GO TO 8
      WRITE (6,992)
992   FORMAT (5X,'NO TRIP.  DEPARTURE TIME IS OUT OF RANGE.'/)
      GO TO 7
8     CALL CPUSUP(START)
```

```
C
C INITIALIZE ARRAYS USED IN THE ALGORITHM
C
      DO 9 S=1,MS
      T(S)=INF
      TB(S)=INF
      PR(S)=0
      PS(S)=0
      F(S)=0
9     CONTINUE
      T(ORG)=TIME
      U=0
      V=0
      I=ORG
      TT=T(I)
C
C START ALGORITHM
C
10    N=NR(I)
      DO 18 J=1,N
      R=ROUTE(I,J)
      M=NS(R)
      DO 11 JJ=1,M
      IF (STOP(R,JJ).EQ.I) GO TO 12
11    CONTINUE
C IF STOP(I) NOT FOUND IN ROUTE R, ERROR AND STOP
      WRITE (6,990) I,R
990   FORMAT ('0*** ERROR ***    STOP',I5,' NOT FOUND ON ROUTE',I5)
      STOP
12    IF (JJ.EQ.M) GO TO 18
C IF JJ IS LAST STOP ON ROUTE R CANNOT DEPART STOP I ON ROUTE R
      IBEG=SBEG(R)
      IEND=SEND(R)
      DO 13 K=IBEG,IEND
      IF (SCHED(K,JJ).GE.TT) GO TO 14
13    CONTINUE
C IF NO DEPARTURES ARE AFTER TT, TRY ANOTHER ROUTE
      GO TO 18
14    TIM=SCHED(K,JJ)
      JJ=JJ+1
C TEST ARRIVAL TIMES AT STOPS AFTER STOP I ON ROUTE R
      DO 17 SS=JJ,M
      S=STOP(R,SS)
      IF (SCHED(K,SS).GE.T(S)) GO TO 17
      PR(S)=R
      PS(S)=I
      IF (T(S).GE.INF) GO TO 15
```

```
C IF S IS ALREADY ON THE LIST L, MOVE S DOWN TO A NEW POSITION AND ZERO
C THE OLD POSITION
      KK=F(S)
      IF (KK.GT.0) L(KK)=0
15    T(S)=SCHED(K,SS)
      TB(S)=TIM
C ADD S TO LIST L
16    V=V+1
      IF (V.GT.ML) V=1
      IF (V.NE.U) GO TO 161
C
C COMPACTIFY LIST L
C
25    W=U+1
      DO 26 IPOS=W,ML
      IF (L(IPOS).EQ.0) GO TO 26
      V=V+1
      ST=L(IPOS)
      L(V)=ST
      F(ST)=V
26    CONTINUE
      W=U-1
      IF (W.EQ.0) GO TO 16
      DO 27 IPOS=1,W
      IF (L(IPOS).EQ.0) GO TO 27
      V=V+1
      IF (V.GT.ML) V=1
      ST=L(IPOS)
      L(V)=ST
      F(ST)=V
27    CONTINUE
      GO TO 16
161   L(V)=S
      F(S)=V
17    CONTINUE
18    CONTINUE
      F(I)=0
      IF (U.NE.0) L(U)=0
C STOP WHEN THE LAST STOP PROCESSED WAS THE LAST ON THE LIST L
19    IF (U.EQ.V) GO TO 20
      U=U+1
      IF (U.GT.ML) U=1
      I=L(U)
      IF (I.EQ.0) GO TO 19
      TT=T(I)+MINTRA(I)
      GO TO 10
```

```
C
C WRITE OUT TRIP
C
20    CALL CPUSUP(FINISH)
      RUNTIM=FINISH-START
      RTIME=RTIME+RUNTIM
      RTSQ=RTSQ+RUNTIM*RUNTIM
      NRUN=NRUN+1
      WRITE (6,908) RUNTIM
908   FORMAT (10X,'RUN TIME FOR LABCOR',I10,' MILLISECONDS'/)
      IF (PS(DST).EQ.0) GO TO 23
      I=DST
      SPRT(MMP)=DST
      K=MP
21    II=PR(I)
      RPRT(K)=II
      TBPRT(K)=TB(I)
      TPRT(K)=T(I)
      I=PS(I)
      SPRT(K)=I
      K=K-1
      IF (I.NE.ORG) GO TO 21
      K=K+1
      DO 22 J=K,MP
      JJ=J+1
      WRITE (6,905) RPRT(J),SPRT(J),TBPRT(J),SPRT(JJ),TPRT(J)
905   FORMAT (5X,'BOARD ROUTE ',I5,' AT STOP ',I5,' AT TIME',I5,
     1' ARRIVE AT STOP ',I5,' AT TIME' I5)
22    CONTINUE
      GO TO 7
23    WRITE (6,906)
906   FORMAT (5X,'NO TRIP FOUND')
      GO TO 7
24    ROUT=FLOAT(RTIME)/FLOAT(NRUN)
      WRITE (6,909) ROUT
909   FORMAT ('0'///'0AVERAGE RUN TIME FOR LABCOR IS',F12.2,
     1' MILLISECONDS')
      ROUT=FLOAT(NRUN)*ROUT*ROUT
      NRUN=NRUN-1
      ROUT=(FLOAT(RTSQ)-ROUT)/FLOAT(NRUN)
      ROUT=SQRT(ROUT)
      WRITE (6,910) ROUT
910   FORMAT ('0STANDARD DEVIATION OF RUN TIME FOR LABCOR IS',F12.2,
     1' MILLISECONDS')
      STOP
      END
```

```
      PARAMETER MR=100          @ MAX NUMBER OF ROUTES
      PARAMETER MMR=MR+1
      PARAMETER MS=120          @ MAX NUMBER OF STOPS
      PARAMETER MSR=12          @ MAX NUMBER OF STOPS PER ROUTE
      PARAMETER MRS=20          @ MAX NUMBER OF ROUTES PER STOP
      PARAMETER MD=3000         @ MAX NUMBER OF VEHICLE DEPARTURES
      PARAMETER MP=6            @ MAX NUMBER OF SEGMENTS PER PATH
      PARAMETER MAT=1441        @ MAX ARRIVAL TIME+1
      PARAMETER MMP=MP+1
      IMPLICIT INTEGER (A-Z)
      REAL ROUT
      LOGICAL HEAD
      DIMENSION T(MS),TB(MS),PS(MS),PR(MS),L(MAT),NR(MS),NS(MR),
     2 SBEG(MMR),SEND(MMR),STOP(MR,MSR),MINTRA(MS),ROUTE(MS,MRS),
     3 SCHED(MD,MSR),SPRT(MMP),RPRT(MP),TPRT(MP),TBPRT(MP),LSUCC(MS),
     4 LPRED(MS),HEAD(MS)
C
C ARRAYS USED IN THIS PROGRAM
C
C    STOP INPUT
C       NR(S)       - NUMBER OF ROUTES STOPPING AT S
C       ROUTE(S,J)  - JTH ROUTE STOPPING AT S
C       MINTRA(S)   - MINIMUM TIME TO TRANSFER BETWEEN ROUTES AT NODE S
C    ROUTE INPUT
C       NS(R)       - NUMBER OF STOPS ON ROUTE R
C       STOP(R,I)   - ITH STOP ON ROUTE R
C       SBEG(R)     - LOCATION IN THE SCHEDULE LIST OF THE FIRST
C                     SCHEDULED DEPARTURE FOR ROUTE R
C       SEND(R)     - LOCATION IN THE SCHEDULE LIST OF THE LAST
C                     SCHEDULED DEPARTURE FOR ROUTE R.
C       SCHED(K,I)  - ARRIVAL TIME AT THE ITH STOP OF THE KTH DEPARTURE
C    ALGORITHM
C       L(S)        - SEQUENCE LIST OF STOPS TO FAN OUT FROM
C       LPRED(S)    - PREDECESSOR NODE TO NODE S IN CHAIN OF NODES
C                     REPRESENTING A LEVEL IN SEQUENCE LIST L.  IF S
C                     HEADS THE CHAIN (HEAD(S)=.TRUE.), THIS POINTER
C                     GIVES THE POSITION OF S IN THE SEQUENCE LIST.
C       LSUCC(S)    - SUCCESSOR NODE TO NODE S IN CHAIN OF NODES
C                     REPRESENTING A LEVEL IN LIST L.
C       HEAD(S)     - LOGICAL VARIABLE USED TO INDICATE WHETHER S
C                     HEADS A CHAIN IN L.
C       T(S)        - ARRIVAL TIME AT STOP S
C       TB(S)       - BOARDING TIME FOR VEHICLE ARRIVING AT S
C       PS(S)       - STOP PRECEDING S IN PATH TO S
C       PR(S)       - ROUTE FROM PS(S) TO S
```

```
C     PRINTING THE PATH
C        SPRT(J)      - STOP
C        RPRT(J)      - ROUTE
C        TPRT(J)      - ARRIVAL TIME
C        TBPRT(J)     - BOARDING TIME
C
      INF=999999999             @ INFINITY USED IN THE PATH CALCULATION
      RTIME=0
      NRUN=0
      RTSQ=0
C
      DO 10 S=1,MS
      NR(S)=0
10    CONTINUE
C
C READ ROUTE INPUT
C
      NSTOP=0
      NROUTE=0
      K=1          @ NUMBER OF DEPARTURES IN THE SCHEDULE
20    READ (7,270,END=50) R,M,KK,(STOP(R,J),J=1,M)
      NS(R)=M
      IF (R.GT.NROUTE) NROUTE=R
C
C READ SCHEDULES FOR ROUTE R
C
      SBEG(R)=K
      DO 30 KS=1,KK
      READ (7,270) (SCHED(K,J),J=1,M)
      K=K+1
30    CONTINUE
      SEND(R)=K-1
C
C ADD ROUTE R TO LIST OF ROUTES STOPPING AT EACH STOP IN R'S STOP LIST
C
      DO 40 I=1,M
      S=STOP(R,I)
      IF (S.GT.NSTOP) NSTOP=S
      NR(S)=NR(S)+1
      J=NR(S)
      ROUTE(S,J)=R
40    CONTINUE
      GO TO 20
50    SBEG(NROUTE+1)=K
      READ (12,280) (MINTRA(I),I=1,NSTOP)
C
C READ ORIGIN AND DESTINATION
C
```

```
60     READ (11,290,END=260) ORG,DST,TIME
       WRITE (6,300) ORG,DST,TIME
       IF (TIME.GE.1.AND.TIME.LE.1440) GO TO 70
       WRITE (6,300) ORG,DST,TIME
       WRITE (6,310)
       GO TO 60
70     CALL CPUSUP (START)
C
C INITIALIZE ARRAYS USED IN THE ALGORITHM
C
       DO 80 S=1,NS
       T(S)=INF
       TB(S)=INF
       PR(S)=0
       PS(S)=0
       LSUCC(S)=0
       LPRED(S)=0
       HEAD(S)=.FALSE.
80     CONTINUE
       T(ORG)=TIME
       I=ORG
       TT=T(I)
       U=TIME+1
       DO 90 S=U,MAT
90     L(S)=0
C
C START ALGORITHM
C
100    N=NR(I)
       DO 200 J=1,N
       R=ROUTE(I,J)
       M=NS(R)
       DO 110 JJ=1,M
       IF (STOP(R,JJ).EQ.I) GO TO 120
110    CONTINUE
C
C IF STOP(I) NOT FOUND IN ROUTE R, ERROR AND STOP
C
       WRITE (6,320) I,R
       STOP
120    IF (JJ.EQ.M) GO TO 200
C
C IF JJ IS LAST STOP ON ROUTE R CANNOT DEPART STOP I ON ROUTE R
C
       IBEG=SBEG(R)
       IEND=SEND(R)
       DO 130 K=IBEG,IEND
       IF (SCHED(K,JJ).GE.TT) GO TO 140
130    CONTINUE
```

```
C
C IF NO DEPARTURES ARE AFTER TT, TRY ANOTHER ROUTE
C
      GO TO 200
140   TIM=SCHED(K,JJ)
      JJ=JJ+1
C
C TEST ARRIVAL TIMES AT STOPS AFTER STOP I ON ROUTE R
C
      DO 190 SS=JJ,M
      S=STOP(R,SS)
      IF (SCHED(K,SS).GE.T(S)) GO TO 190
      IF (T(S).GE.INF) GO TO 170
C
C IF S IS ALREADY ON THE LIST, REMOVE IT.
C
      X=LSUCC(S)
      Y=LPRED(S)
      IF (HEAD(S)) GO TO 150
      LSUCC(Y)=X
      IF (X.NE.0) GO TO 160
      GO TO 170
150   L(Y)=X
      IF (X.EQ.0) GO TO 170
      HEAD(X)=.TRUE.
160   LPRED(X)=Y
C
C PUT S ON THE SEQUENCE LIST.
C
170   Y=SCHED(K,SS)+1
      X=L(Y)
      L(Y)=S
      LSUCC(S)=X
      HEAD(S)=.TRUE.
      LPRED(S)=Y
      IF (X.EQ.0) GO TO 180
      LPRED(X)=S
      HEAD(X)=.FALSE.
180   T(S)=SCHED(K,SS)
      TB(S)=TIM
      PR(S)=R
      PS(S)=I
190   CONTINUE
200   CONTINUE
C
C STOP WHEN POINTER REACHES DST OR WHEN A COMPLETE PASS
C PRODUCES NO NEW ADDITIONS TO THE SHORTEST PATH TREE.
C
```

```
210    I=L(U)
       IF (I.NE.0) GO TO 211
       U=U+1
       IF (U.GT.MAT) GO TO 220
       GO TO 210
211    IF (I.EQ.DST) GO TO 220
       TT=T(I)+MINTRA(I)
       V=LSUCC(I)
       L(U)=V
       IF (V.EQ.0) GO TO 100
       LPRED(V)=U
       HEAD(V)=.TRUE.
       GO TO 100
C
C WRITE OUT TRIP
C
220    CALL CPUSUP (FINISH)
       RUNTIM=FINISH-START
       RTIME=RTIME+RUNTIM
       RTSQ=RTSQ+RUNTIM*RUNTIM
       NRUN=NRUN+1
       WRITE (6,340) RUNTIM
       IF (PS(DST).EQ.0) GO TO 250
       I=DST
       SPRT(MP)=DST
       K=MP
230    II=PR(I)
       RPRT(K)=II
       TBPRT(K)=TB(I)
       TPRT(K)=T(I)
       I=PS(I)
       SPRT(K)=I
       K=K-1
       IF (I.NE.ORG) GO TO 230
       K=K+1
       DO 240 J=K,MP
       JJ=J+1
       WRITE (6,350) RPRT(J),SPRT(J),TBPRT(J),SPRT(JJ),TPRT(J)
240    CONTINUE
       GO TO 60
250    WRITE (6,360)
       GO TO 60
260    ROUT=FLOAT(RTIME)/FLOAT(NRUN)
       WRITE (6,370) ROUT
       ROUT=FLOAT(NRUN)*ROUT*ROUT
       NRUN=NRUN-1
       ROUT=(FLOAT(RTSQ)-ROUT)/FLOAT(NRUN)
       ROUT=SQRT(ROUT)
       WRITE (6,380) ROUT
       STOP
```

```
C
C
270   FORMAT (20I5)
280   FORMAT (16I5)
290   FORMAT (3I5)
300   FORMAT ('0'///'0TRIP FROM',I5,' TO',I5,' DEPARTING ON OR AFTER',I5
     2/)
310   FORMAT (5X,'NO TRIP.  DEPARTURE TIME IS OUT OF RANGE.'/)
320   FORMAT ('0*** ERROR ***      STOP',I5,' NOT FOUND ON ROUTE',I5)
340   FORMAT (10X,'RUN TIME FOR LABSET',I10,' MILLISECONDS'/)
350   FORMAT (5X,'BOARD ROUTE ',I5,' AT STOP ',I5,' AT TIME',I5,
     2 ' ARRIVE AT STOP ',I5,' AT TIME'I5)
360   FORMAT (5X,'NO TRIP FOUND')
370   FORMAT ('0'///'0AVERAGE RUN TIME FOR LABSET IS',F12.2,
     1' MILLISECONDS')
380   FORMAT ('0STANDARD DEVIATION OF RUN TIME FOR LABSET IS',F12.2,
     1' MILLISECONDS')
C
C
      END
```

```
C    ********** TIMEXD: DEPARTURE ORIENTED CRITERION **********
C    THIS PROGRAM USES A TIME-EXPANDED REPRESENTATION IN ORDER TO
C    CALCULATE THE BEST PATH FROM A GIVEN ORIGIN TO A GIVEN DESTIN-
C    ATION TO DEPART AFTER A SPECIFIED TIME (AND ARRIVE AS EARLY AS
C    POSSIBLE).  INPUT CONSISTS OF THE NODE AND ARC DATA FOR THE
C    TIME-EXPANDED NETWORK TOGETHER WITH A LIST OF TRIPS FOR WHICH
C    TRIP ITINERARIES ARE REQUIRED.  OUTPUT CONSISTS OF AN ITINERARY
C    FOR EACH TRIP, THE CALCULATION TIME IN MILLISECONDS FOR EACH
C    TRIP AS WELL AS THE AVERAGE AND STANDARD DEVIATION OF CALCULATION
C    TIMES FOR ALL TRIPS IN THE LIST.
C
C    ----------
C
C    NATIONAL BUREAU OF STANDARDS          JULY, 1976
C
C    ----------
C
      PARAMETER MT=1600         @ NUMBER OF MINUTES
      PARAMETER MAX=100         @ NUMBER OF ROUTES/PATH
      PARAMETER NODES=4500      @ NUMBER OF NODES
      PARAMETER ARCS=13000      @ NUMBER OF ARCS
      DIMENSION N(NODES),T(NODES),P(NODES),ARC(NODES),PRTE(NODES),NN(MT)
     1          ,TO(ARCS),RTE(ARCS),PATHN(MAX),PATHT(MAX),PATHR(MAX)
      REAL ROUT
C
C    ----------
C
C    VARIABLES AND ARRAYS USED IN THIS PROGRAM
C
C      NODE AND ARC DATA
C
C        N(I) - STOP ASSOCIATED WITH NETWORK NODE I
C        T(I) - TIME ASSOCIATED WITH NETWORK NODE I
C        ARC(I) - LAST POSITION WHERE INFORMATION MAY BE
C                 FOUND FOR NODE I IN ARC LIST
C        TO(J) - ENDING NODE OF ARC IN POSITION J OF ARC LIST
C        RTE(J) - ROUTE NUMBER CORRESPONDING TO ARC IN POSITION
C                 J OF ARC LIST
C        NN(T) - NODE CORRESPONDING TO THE FIRST OCCURRENCE OF
C                TIME T OR LATER
C        NODE - NUMBER OF NETWORK NODES
C
C      INPUT VARIABLES FOR TRIP
C
C        ORG - DESIRED ORIGIN STOP OF TRIP
C        DST - DESIRED DESTINATION STOP OF TRIP
C        TIME - TIME AT OR AFTER WHICH TRIP IS TO BEGIN
C
```

```
C     VARIABLES AND ARRAYS USED IN THE ALGORITHM
C
C       DONE - FIRST NODE FOR WHICH THE DESTINATION STOP HAS
C              BEEN ENCOUNTERED
C       P(I) - PREDECESSOR NODE TO NODE I ALONG CURRENT PATH
C       PRTE(I) - ROUTE INTO NODE I ALONG THE CURRENT PATH
C
C     ARRAYS USED IN PRINTING THE PATH
C
C       PATHN(K) - STOP IN POSITION K ALONG PATH
C       PATHT(K) - TIME IN POSITION K ALONG PATH
C       PATHR(K) - ROUTE IN POSITION K ALONG PATH
C
C     VARIABLES USED IN TIMING CALCULATIONS
C
C       DIFF - CPU TIME (IN MSECS.) USED IN CALCULATING ONE TRIP
C       RTIME - CUMULATIVE SUM OF DIFF'S
C       RTSQ - CUMULATIVE SUM OF SQUARES OF DIFF'S
C       NRUN - NUMBER OF TRIPS CALCULATED
C       ROUT - USED IN PRINTING AVERAGE AND STANDARD DEVIATION
C              OF TRIP CALCULATION TIMES
C
        IMPLICIT INTEGER (A-Z)
C
C     READ THE NETWORK NODE DATA, SORTED ON T.   ASSUME ALL T(I) > 0.
C     THE SCHEDULE TIMES ARE GIVEN UP TO 1600 MINUTES.
C
C  ----------
C
        I=1
        TT=0
        IN1=9       @ INPUT UNIT FOR NODE DATA
  100   READ(IN1,900,END=103) N(I),T(I)
  900   FORMAT(3I5)
        IF(T(I).EQ.TT) GO TO 102
        TN=TT+1
        TM=T(I)
        DO 101 J=TN,TM
        NN(J)=I
  101   CONTINUE
        TT=T(I)
  102   I=I+1
        GO TO 100
  103   NODE=I-1
C
C     READ THE ARC DATA, SORTED BY ORIGIN NODE.
C
        L=1
        K=1
        IN2=10      @ INPUT UNIT FOR ARC DATA
```

```
  104 READ(IN2,900,END=107) I,TO(L),RTE(L)
      IF(I.EQ.K) GO TO 106
      KM=I-1
      DO 105 J=K,KM
      ARC(J)=L-1
  105 CONTINUE
      K=I
  106 L=L+1
      GO TO 104
  107 DO 108 J=I,NODE
      ARC(J)=L-1
  108 CONTINUE
      WRITE (6,910)
  910 FORMAT(1H1)
      NRUN=0
      RTIME=0
      RTSQ=0
C
C    READ ORIGIN,DESTINATION AND DESIRED STARTING TIME(ASSUMED TO BE
C    BETWEEN 1 AND 1440, INCLUSIVE).
C
      IN3=11     @ INPUT UNIT FOR TRIP DATA
  200 READ(IN3,900,END=500) ORG,DST,TIME
      WRITE(6,911) ORG,DST,TIME
  911 FORMAT(//1H0,'TRIP FROM',I5,' TO',I5,' DEPARTING ON OR AFTER',
     1I5/)
      IF(TIME.LT.1.OR.TIME.GT.1440) GO TO 405
      IF(TIME.GT.TT) GO TO 406
      CALL CPUSUP(STIME)
C
C    INITIALIZE PATH ARRAY.
C
      DO 201 I=1,NODE
      P(I)=0
      PRTE(I)=0
  201 CONTINUE
C
C    BEGIN CALCULATION OF ROUTES FROM ORG TO DST.
C
      DONE=NODE+1
      I=NN(TIME)
  202 IF(N(I).EQ.ORG) GO TO 203
      I=I+1
      IF(I.GT.NODE) GO TO 406
      GO TO 202
  203 L=ARC(I-1)+1
      IF(I.EQ.1) L=1
      END=ARC(I)
      PRTI=PRTE(I)
```

```
  204 IF(END.LT.L) GO TO 206
      IF(RTE(L).EQ.9999.AND.PRTI.EQ.9999) GO TO 205
      J=TO(L)
      P(J)=I
      PRTE(J)=RTE(L)
      IF(N(J).NE.DST) GO TO 205
      IF(DONE.GT.J) DONE=J
  205 L=L+1
      GO TO 204
  206 I=I+1
      IF(I.GT.NODE) GO TO 406
      IF(I.EQ.DONE) GO TO 300
      IF(P(I).EQ.0.AND.N(I).NE.ORG) GO TO 206
      GO TO 203
C
C     STORE INFORMATION ABOUT SELECTED ROUTE FROM ORG TO DST.
C
  300 M=DONE
      K=MAX+1
  301 K=K-1
      R=PRTE(M)
      PATHN(K)=N(M)
      PATHT(K)=T(M)
      PATHR(K)=R
  302 M=P(M)
      IF(N(M).EQ.ORG) GO TO 303
      IF(PRTE(M).NE.R) GO TO 301
      GO TO 302
  303 K=K-1
      PATHN(K)=ORG
      PATHT(K)=T(M)
      CALL CPUSUP(FTIME)
C
C     PRINT ROUTE INFORMATION.
C
      KM=MAX-1
      DO 304 L=K,KM
      LL=L+1
      IF(PATHN(L).EQ.PATHN(LL)) GO TO 304
      WRITE(6,901) PATHR(LL),PATHN(L),PATHT(L),PATHN(LL),PATHT(LL)
  901 FORMAT(5X,'BOARD ROUTE',I5,' AT STOP',I5,' AT TIME',I5,
     1' ARRIVE AT STOP',I5,' AT TIME',I5)
  304 CONTINUE
  305 DIFF=FTIME-STIME
      NRUN=NRUN+1
      RTIME=RTIME+DIFF
      RTSQ=RTSQ+DIFF*DIFF
      WRITE(6,902) DIFF
  902 FORMAT(1H0,'RUN TIME FOR TIMEXD = ',I6,' MILLISECONDS')
      GO TO 202
```

```
405 WRITE(6,905) TIME
905 FORMAT(1X,'TIME',I5,2X,'SHOULD BE BETWEEN 1 AND 1440 --- TRY AGAIN
   1')
    GO TO 200
406 CALL CPUSUP(FTIME)
    WRITE(6,906)
906 FORMAT(    1X,'NO TRIP EXISTS FROM ORIGIN TO DESTINATION LEAVING W
   1ITHIN 160 MINUTES AFTER THE REQUESTED TIME --- TRY ANOTHER DEPARTU
   2RE TIME')
    GO TO 305
500 ROUT=FLOAT(RTIME)/FLOAT(NRUN)
    WRITE(6,907) ROUT
907 FORMAT(///1H0,'AVERAGE RUN TIME =',F12.2,' MILLISECONDS')
    ROUT=FLOAT(NRUN)*ROUT*ROUT
    NRUN=NRUN-1
    ROUT=(FLOAT(RTSQ)-ROUT)/FLOAT(NRUN)
    ROUT=SQRT(ROUT)
    WRITE(6,908) ROUT
908 FORMAT(///1H0,'STANDARD DEVIATION OF RUN TIMES =',F12.2,
   1' MILLISECONDS')
    STOP
    END
```

B.8 PROGRAM TIMEXA

```
C   ********** TIMEXA: ARRIVAL ORIENTED CRITERION **********
C   THIS PROGRAM USES A TIME-EXPANDED REPRESENTATION IN ORDER TO
C   CALCULATE THE BEST PATH FROM A GIVEN ORIGIN TO A GIVEN DESTIN-
C   ATION TO ARRIVE BEFORE A SPECIFIED TIME (AND DEPART AS LATE AS
C   POSSIBLE).  INPUT CONSISTS OF THE NODE AND ARC DATA FOR THE
C   TIME-EXPANDED NETWORK TOGETHER WITH A LIST OF TRIPS FOR WHICH
C   TRIP ITINERARIES ARE REQUIRED.  OUTPUT CONSISTS OF AN ITINERARY
C   FOR EACH TRIP, THE CALCULATION TIME IN MILLISECONDS FOR EACH
C   TRIP AS WELL AS THE AVERAGE AND STANDARD DEVIATION OF CALCULATION
C   TIMES FOR ALL TRIPS IN THE LIST.
C
C   ----------
C
C   NATIONAL BUREAU OF STANDARDS       JULY, 1976
C
C   ----------
C
      PARAMETER MT=1600          @ NUMBER OF MINUTES
      PARAMETER MAX=100          @ NUMBER OF ROUTES/PATH
      PARAMETER NODES=4500       @ NUMBER OF NODES
      PARAMETER ARCS=13000       @ NUMBER OF ARCS
      DIMENSION N(NODES),T(NODES),S(NODES),ARC(NODES),SRTE(NODES),NN(MT)
     1          ,TO(ARCS),RTE(ARCS),PATHN(MAX),PATHT(MAX),PATHR(MAX)
      REAL ROUT
C
C   ----------
C
C   VARIABLES AND ARRAYS USED IN THIS PROGRAM
C
C     NODE AND ARC DATA
C
C       N(I) - STOP ASSOCIATED WITH NETWORK NODE I
C       T(I) - TIME ASSOCIATED WITH NETWORK NODE I
C       ARC(I) - LAST POSITION WHERE INFORMATION MAY BE
C                FOUND FOR NODE I IN ARC LIST
C       TO(J) - ENDING NODE OF ARC IN POSITION J OF ARC LIST
C       RTE(J) - ROUTE NUMBER CORRESPONDING TO ARC IN POSITION
C                J OF ARC LIST
C       NN(T) - NODE CORRESPONDING TO THE LAST OCCURRENCE OF
C               TIME T OR EARLIER
C       NODE - NUMBER OF NETWORK NODES
C
C     INPUT VARIABLES FOR TRIP
C
C       ORG - DESIRED ORIGIN STOP OF TRIP
C       DST - DESIRED DESTINATION STOP OF TRIP
C       TIME - TIME BY WHICH TRIP IS TO BE COMPLETED
C
```

```
C     VARIABLES AND ARRAYS USED IN THE ALGORITHM
C
C        S(I) - SUCCESSOR NODE TO NODE I ALONG CURRENT PATH
C        SRTE(I) - ROUTE OUT OF NODE I ALONG THE CURRENT PATH
C
C     ARRAYS USED IN PRINTING THE PATH
C
C        PATHN(K) - STOP IN POSITION K ALONG PATH
C        PATHT(K) - TIME IN POSITION K ALONG PATH
C        PATHR(K) - ROUTE IN POSITION K ALONG PATH
C
C     VARIABLES USED IN TIMING CALCULATIONS
C
C        DIFF - CPU TIME (IN MSECS.) USED IN CALCULATING ONE TRIP
C        RTIME - CUMULATIVE SUM OF DIFF'S
C        RTSQ - CUMULATIVE SUM OF SQUARES OF DIFF'S
C        NRUN - NUMBER OF TRIPS CALCULATED
C        ROUT - USED IN PRINTING AVERAGE AND STANDARD DEVIATION
C               OF TRIP CALCULATION TIMES
C
C  ----------
C
      IMPLICIT INTEGER (A-Z)
C
C     READ THE NETWORK NODE DATA, SORTED ON T.   ASSUME ALL T(I) > 0.
C     THE SCHEDULE TIMES ARE GIVEN UP TO 1600 MINUTES.
C
      I=1
      TT=1
      IN1=9     @ INPUT UNIT FOR NODE DATA
  100 READ(IN1,900,END=103) N(I),T(I)
  900 FORMAT(3I5)
      IF(T(I).EQ.TT) GO TO 102
      TM=T(I)-1
      DO 101 J=TT,TM
      NN(J)=I-1
  101 CONTINUE
      TT=T(I)
  102 I=I+1
      GO TO 100
  103 NODE=I-1
      DO 110 J=TT,1600
      NN(J)=NODE
  110 CONTINUE
C
C     READ THE ARC DATA, SORTED BY ORIGIN NODE.
C
      L=1
      K=1
```

```
      IN2=10     @ INPUT UNIT FOR ARC DATA
  104 READ(IN2,900,END=107) I,TO(L),RTF(L)
      IF(I.EQ.K) GO TO 106
      KM=I-1
      DO 105 J=K,KM
      ARC(J)=L-1
  105 CONTINUE
      K=I
  106 L=L+1
      GO TO 104
  107 DO 108 J=I,NODE
      ARC(J)=L-1
  108 CONTINUE
      WRITE(6,910)
  910 FORMAT(1H1)
      NRUN=0
      RTIME=0
      RTSQ=0
C
C     READ ORIGIN,DESTINATION AND DESIRED ARRIVAL TIME(ASSUMED TO BE
C     BETWEEN 1 AND 1440, INCLUSIVE).
C
      IN3=11     @ INPUT UNIT FOR TRIP DATA
  200 READ(IN3,900,END=500) ORG,DST,TIME
      WRITE(6,911) ORG,DST,TIME
  911 FORMAT(//1H0,'TRIP FROM',I5,' TO',I5,' ARRIVING BY',I5/)
      IF(TIME.LT.1.OR.TIME.GT.1440) GO TO 405
      CALL CPUSUP(STIME)
C
C     INITIALIZE SUCCESSOR ARRAY.
C
      DO 201 I=1,NODE
      S(I)=0
      SRTF(I)=0
  201 CONTINUE
C
C     BEGIN CALCULATION OF ROUTES FROM ORG TO DST.
C
      IF(TIME.LT.160) TIME=TIME+1440
      I=NN(TIME)
      IF(I.EQ.0) GO TO 406
  202 IF(N(I).EQ.DST) GO TO 203
      I=I-1
      IF(I.EQ.0) GO TO 406
      GO TO 202
  203 S(I)=I
  204 I=I-1
      IF(I.EQ.0) GO TO 406
  205 IF(N(I).EQ.DST) GO TO 203
      L=ARC(I-1)+1
      IF(I.EQ.1) L=1
      END=ARC(I)
```

```
  206 IF(END.LT.L) GO TO 204
      J=TO(L)
      IF(S(J).EQ.0) GO TO 207
      IF(RTE(L).EQ.9999.AND.SRTE(J).EQ.9999) GO TO 207
      S(I)=J
      SRTE(I)=RTE(L)
      IF(N(I).EQ.ORG) GO TO 300
      GO TO 204
  207 L=L+1
      GO TO 206
C
C    STORE INFORMATION ABOUT SELECTED ROUTE FROM ORG TO DST.
C
  300 M=I
      K=0
  301 K=K+1
      R=SRTE(M)
      PATHN(K)=N(M)
      PATHT(K)=T(M)
      PATHR(K)=R
  302 M=S(M)
      IF(N(M).EQ.DST) GO TO 303
      IF(SRTE(M).NE.R) GO TO 301
      GO TO 302
  303 K=K+1
      PATHN(K)=DST
      PATHT(K)=T(M)
      CALL CPUSUP(FTIME)
C
C    PRINT ROUTE INFORMATION.
C
      KK=K-1
      DO 304 L=1,KK
      LL=L+1
      IF(PATHN(L).EQ.PATHN(LL)) GO TO 304
      WRITE(6,901) PATHR(L),PATHN(L),PATHT(L),PATHN(LL),PATHT(LL)
  901 FORMAT(5X,'BOARD ROUTE',I5,' AT STOP',I5,' AT TIME',I5,
     1' ARRIVE AT STOP',I5,' AT TIME',I5)
  304 CONTINUE
  305 DIFF=FTIME-STIME
      NRUN=NRUN+1
      RTIME=RTIME+DIFF
      RTSQ=RTSQ+DIFF*DIFF
      WRITE(6,902) DIFF
  902 FORMAT(1H0,'RUN TIME FOR TIMEXA = ',I6,' MILLISECONDS')
      GO TO 200
  405 WRITE(6,905) TIME
  905 FORMAT(     1X,'TIME',I5,2X,'SHOULD BE BETWEEN 1 AND 1440 --- TRY A
     1GAIN')
      GO TO 200
```

```
406 CALL CPUSUP(FTIME)
    WRITE(6,906)
906 FORMAT(     1X,'NO TRIP EXISTS FROM ORIGIN TO DESTINATION ARRIVING
   1WITHIN 160 MINUTES PRIOR TO THE REQUESTED TIME --- TRY ANOTHER ARR
   2IVAL TIME')
    GO TO 305
500 ROUT=FLOAT(RTIME)/FLOAT(NRUN)
    WRITE(6,907) ROUT
907 FORMAT(///1H0,'AVERAGE RUN TIME =',F12.2,' MILLISECONDS')
    ROUT=FLOAT(NRUN)*ROUT*ROUT
    NRUN=NRUN-1
    ROUT=(FLOAT(RTSQ)-ROUT)/FLOAT(NRUN)
    ROUT=SQRT(ROUT)
    WRITE(6,908) ROUT
908 FORMAT(///1H0,'STANDARD DEVIATION OF RUN TIMES =',F12.2,
   1' MILLISECONDS')
    STOP
    END
```

B.9 PROGRAM REMOVE

```
C THIS PROGRAM REMOVES UNNECESSARY NODES FROM A TRANSIT NETWORK, LEAVING
C ONLY THOSE NODES AT WHICH TRANSFERRING IS POSSIBLE AND LIKELY.
C
      PARAMETER    MNODE  = 500       @ MAX NUMBER OF NODES
      PARAMETER    MRTE   = 1000      @ MAX NUMBER OF ROUTES
      PARAMETER    MRSTOP = 10        @ MAX NUMBER OF ROUTES STOPPING AT A
C                                       NODE
      PARAMETER    MSRTE  = 50        @ MAX NUMBER OF STOPES PER ROUTE
      PARAMETER    MSEG   = 10000     @ MAX NUMBER OF ROUTE SEGMENTS PLUS
C                                       ONE EXTRA FOR EACH ROUTE
      IMPLICIT INTEGER (A-Z)
      LOGICAL NIN
      DIMENSION   SRTE(MSEG),REND(MRTE),NIN(MNODE),ORDER(MNODE),
     1            RSTOP(MNODE,MRSTOP),NRTE(MNODE),TEMP(MSRTE),
     2            SORTN(MNODE),TIME(MSRTE),INDEX(MSRTE)
C
C-------------------
C
C ARRAYS USED IN THIS PROGRAM
C
C          SRTE    - STOPS ON EACH ROUTE.  REND IS USED TO INDICATE WHICH
C                    SECTION OF THE ARRAY SRTE REFERS TO A PARTICULAR ROUTE
C          REND    - POSITION IN SRTE OF THE LAST STOP IN EACH ROUTE
C          NIN     - IS TRUE IF NODE HAS NOT BEEN REMOVED (I.E. NODE IS IN)
C                    IS FALSE IF NODE HAS BEEN REMOVED (I.E. NODE IS OUT)
C          ORDER   - USED IN SORTING, GIVES THE ORIGINAL POSITION OF THE
C                    ITH ENTRY IN SORTED ORDER
C          RSTOP   - FOR EACH NODE, LISTS THE ROUTES STOPPING AT THAT NODE
C          NRTE    - NUMBER OF ROUTES STOPPING AT EACH NODE
C          TEMP    - TEMPORARY STORAGE FOR INPUT AND OUTPUT OF NODES ON A
C                    ROUTE.  ALSO USED AS AN INTERMEDIATE ARRAY STORING THE
C                    POSITION IN EACH ROUTE STOPPING AT A NODE, OF THAT
C                    NODE
C          SORTN   - USED IN SORTING THE NODES ON NUMBER OF ROUTES STOPPING
C                    AT EACH.  STORES THE SORTED NUMBER OF ROUTES
C
C-------------------
C
C INITIALIZE ARRAYS
C
      DO 1 I=1,MNODE
      NRTE(I)=0
      NIN(I)=.TRUE.
1     CONTINUE
```

```
C
C------------------
C
C READ ROUTES
C
      K=0
C   K KEEPS TRACK OF THE CURRENT ROUTE READ IN
      N=0
C   N IS THE MAXIMUM NODE NUMBER ENCOUNTERED SO FAR
      NR=0
C   NR IS THE NUMBER OF TOTAL STOPS IN ROUTES
      NEND=0
C NEND IS THE NUMBER OF NODES WHICH BEGIN OR END A ROUTE
2     READ (7,900,END=4) K,NS,NT,(TEMP(I),I=1,NS)
900   FORMAT (20I5)
      DO 201 J=1,NT
      READ (7,900) (TIME(I),I=1,NS)
201   CONTINUE
C
C THE ROUTE INPUT IS READ IN THE FOLLOWING FORMAT -
C FOR EACH ROUTE -
C      CARD 1, COLS. 1-5 CONTAIN THE NUMBER OF STOPS IN THIS ROUTE
C      THE STOPS ON THE ROUTE ARE THEN LISTED IN ORDER, 5COLS. PER STOP.
C      ADDITIONAL CARDS ARE USED IF NEEDED, WITH THE DATA STARTING IN
C      COL. 6, AGAIN 5 COLS. PER STOP.
C STORE THIS ROUTE IN THE ARRAY SRTE, CHECK MAXIMUM NODE NUMBER, ADD
C THIS ROUTE TO THE ROUTES STOPPING AT EACH STOP IN THE ROUTE.
      DO 3 L=1,NS
      I=TEMP(L)
      IF (I.GT.N) N=I
      NR=NR+1
      SRTE(NR)=I
      NRTE(I)=NRTE(I)+1
      NN=NRTE(I)
      RSTOP(I,NN)=K
3     CONTINUE
      REND(K)=NR
      GO TO 2
C
C------------------
C
C SORT NODES ON THE NUMBER OF ROUTES STOPPING AT EACH
C
4      NRT=K
      REWIND 7
      CALL SORTP(NRTE,N,SORTN,ORDER)
C NRT IS THE NUMBER OF ROUTES
```

```
C
C------------------
C
C REMOVE ALL ONE-ROUTE NODES WHICH DO NOT BEGIN OR END A ROUTE
C
      ENTRY=1
C WE PROCESS THE NODES IN THE ORDER THEY APPEAR IN THE ARRAY ORDER.
C ENTRY IS THE CURRENT POSITION IN ORDER.
5     K=ORDER(ENTRY)
C K IS THE CURRENT NODE WHICH IS BEING TESTED FOR REMOVAL
      M=NRTE(K)
      IF (M-1) 10,6,11
6     R=RSTOP(K,1)
C FIND THE STOPS ON ROUTE R IN THE ARRAY SRTE
      BEG=REND(R-1)+1
      IF (R.EQ.1) BEG=1
C IF K IS THE FIRST STOP ON ROUTE R, K CANNOT BE REMOVED
      IF (K.EQ.SRTE(BEG)) GO TO 9
      BEG=BEG+1
      END=REND(R)
C IF K IS THE LAST STOP ON ROUTE R, K CANNOT BE REMOVED
      IF (K.EQ.SRTE(END)) GO TO 9
      END=END-1
      IF (END.GE.BEG) GO TO 7
C ALL ROUTES MUST HAVE AT LEAST 2 STOPS
      WRITE (6,991) K,R,BEG,END
991   FORMAT ('0*** ERROR ***   WHILE REMOVING NODE',I5,' FROM ROUTE',
     1I5,' ROUTE LIST RANGE (',I5,',',I5,') IS TOO SHORT'/)
      GO TO 10
C REMOVE NODE K FROM ROUTE R
7     DO 8 I=BEG,END
      IF (SRTE(I).NE.K) GO TO 8
      SRTE(I)=0
      NIN(K)=.FALSE.
      GO TO 10
8     CONTINUE
C NODE K MUST APPEAR IN ROUTE R
      WRITE (6,992) K,R,BEG,END
992   FORMAT ('0*** ERROR ***   NODE',I5,' NOT FOUND IN ROUTE',I5,
     1' ROUTE LIST RANGE(',I5,',',I5,')'/)
      GO TO 10
9     NEND=NEND+1
C INCREMENT THE CURRENT ENTRY IN THE LIST SORTED BY NUMBER OF ROUTES
C STOPPING AT EACH NODE
10    ENTRY=ENTRY+1
      IF (ENTRY.LE.N) GO TO 5
```

```
C
C-------------------
C
C EXAMINE OTHER NODES FOR POSSIBLE REMOVAL
C
C L IS THE POSITION OF THE ROUTE CURRENTLY BEING EXAMINED IN THE LIST OF
C ROUTES STOPPING AT NODE K
C R IS THE CURRENT ROUTE
11    L=1
12    R=RSTOP(K,L)
C FIND THE POSITIONS OF ROUTE R IN THE LIST SRTE, AND FIND THE POSITION
C WITHIN THAT ROUTE OCCUPIED BY K
      BEG=REND(R-1)+1
      IF (R.EQ.1) BEG=1
      PREV=SRTE(BEG)
C CHECK THAT K IS NEITHER THE FIRST NOR LAST NODE ON ROUTE R
      IF (PREV.EQ.K) GO TO 24
      BEG=BEG+1
      END=REND(R)
      IF (SRTE(END).EQ.K) GO TO 24
      END=END-1
      IF (END.GE.BEG) GO TO 13
C EACH ROUTE HAS AT LEAST 2 NODES
       WRITE (6,991) K,R,BEG,END
C SEARCH FOR K AS AN INTERMEDIATE STOP ON ROUTE R
      GO TO 25
13    DO 17 J=BEG,END
      I=SRTE(J)
      IF (I.EQ.0) GO TO 17
      IF (I.NE.K) GO TO 16
      NEXT=SRTE(J+1)
      IF (NEXT.GT.0) GO TO 15
      JJ=J+2
14    NEXT=SRTE(JJ)
      IF (NEXT.GT.0) GO TO 15
      JJ=JJ+1
      GO TO 14
15    TEMP(L)=J
      GO TO 18
16    PREV=I
17    CONTINUE
C NODE K MUST BE FOUND ON ROUTE R
      WRITE (6,992) K,R
      GO TO 25
18    IF (L.GT.1) GO TO 20
C STORE MINIMUM OF PREV AND NEXT IN P, MAXIMUM IN Q
      IF (PREV.GT.NEXT) GO TO 19
      P=PREV
      Q=NEXT
      GO TO 22
```

```
19    P=NEXT
      Q=PREV
      GO TO 22
C TEST IF PREV AND NEXT  ARE THE SAME AS P AND Q,  THAT IS WHETHER THE
C PREVIOUS AND NEXT NODES ON ROUTE R ARE THE NODES ADJACENT TO K ON
C OTHER ROUTES
20    IF (PREV.GT.NEXT) GO TO 21
      IF (PREV.NE.P) GO TO 25
      IF (NEXT.NE.Q) GO TO 25
      GO TO 22
21    IF (NEXT.NE.P) GO TO 25
      IF (PREV.NE.Q) GO TO 25
C CHECK NEXT ROUTE STOPPING AT K
22    L=L+1
      IF (L.LE.M) GO TO 12
C REMOVE NODE K, ZERO SRTE, AND SET NIN=FALSE
      DO 23 L=1,M
      J=TEMP(L)
      SRTE(J)=0
      NIN(K)=.FALSE.
23    CONTINUE
      GO TO 25
24    NEND=NEND+1
C CHECK NEXT NODE FOR POSSIBLE REMOVAL
25    ENTRY=ENTRY+1
      IF (ENTRY.GT.N) GO TO 26
      K=ORDER(ENTRY)
      M=NRTE(K)
      GO TO 11
C
C-----------------
C
C PRINT OUT NODES WHICH HAVE BEEN DELETED
C
26    WRITE (6,901)
901   FORMAT ('1THE FOLLOWING NODES WERE DELETED'/)
      J=0
      NDEL=0
      DO 27 I=1,N
      IF (NIN(I)) GO TO 27
      J=J+1
      NDEL=NDEL+1
      TEMP(J)=I
      IF (J.LT.25) GO TO 27
```

```
      WRITE (6,992) (TEMP(J),J=1,25)
902   FORMAT (5X,25I5)
      J=0
27    CONTINUE
      IF (J.LT.0) GO TO 28
      WRITE (6,902) (TEMP(I),I=1,J)
      WRITE (6,903) NDEL,NEND
903   FORMAT ('0',I4,' NODES DELETED'/I5,' NODES BEGIN OR END A ROUTE')
C
C-------------------
C
C PRINT OUT REVISED ROUTES WITH DELETED NODES OMITTED
C
28    WRITE (6,904)
904   FORMAT ('1REVISED ROUTES'//' RTE',10X,'STOPS'/)
      END=0
      DO 30 I=1,NRT
      READ (7,900) II,NS,NT
      BEG=END+1
      END=REND(I)
      K=0
      DO 29 J=BEG,END
      IF (SRTE(J).EQ.0) GO TO 29
      K=K+1
      INDEX(K)=J
      TEMP(K)=SRTE(J)
29    CONTINUE
      WRITE (6,905) I,(TEMP(J),J=1,K)
905   FORMAT (I5,2X,25I5/(7X,25I5))
      WRITE (13,900) II,K,NT,(TEMP(J),J=1,K)
      DO 292 IT=1,NT
      READ (7,900) (TIME(J),J=1,NS)
      DO 291 IJ=1,K
      J=INDEX(IJ)
      TIME(IJ)=TIME(J)
291   CONTINUE
      WRITE (13,900) (TIME(J),J=1,K)
292   CONTINUE
30    CONTINUE
      END FILE 13
      STOP
      END
```

NBS-114A (REV. 7-73)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET	1. PUBLICATION OR REPORT NO. NBSIR 78-1426	2. Gov't Accession No.	3. Recipient's Accession No.

4. TITLE AND SUBTITLE Comparison of the Performance of Three Algorithms for Use in an Automated Transit Information System (ATIS)	5. Publication Date March 1978 6. Performing Organization Code
7. AUTHOR(S) J. F. Gilsinn, E. Leyendecker, D. R. Shier	8. Performing Organ. Report No.
9. PERFORMING ORGANIZATION NAME AND ADDRESS NATIONAL BUREAU OF STANDARDS DEPARTMENT OF COMMERCE WASHINGTON, D.C. 20234	10. Project/Task/Work Unit No. 2050402 11. Contract/Grant No.
12. Sponsoring Organization Name and Complete Address *(Street, City, State, ZIP)* National Bureau of Standards A-428, 101 Washington, DC 20234	13. Type of Report & Period Covered 14. Sponsoring Agency Code

15. SUPPLEMENTARY NOTES

16. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here.)*

This paper compares the performance of three algorithms for computing trip itineraries for use in an automated transit information system. One of the approaches (TIMEXD) is based on a time-expanded network. The other two both compute paths in a bipartite route/stop network; one algorithm (LABCOR) is based on the label-correcting approach and the other (LABSET) on the label-setting approach. The transit networks upon which the performance comparison is based are of two types: a grid network with specified, possibly non-uniform, distances between streets, and a spider web type of network. TIMEXD is fastest on all the larger networks, but it requires most computer storage and outputs paths with more transfers. LABCOR is the slowest, but is guaranteed to produce the best routing, since it always outputs an optimal path with fewest transfers. Computation time estimates extrapolated to large transit networks indicate times of 1.5 to 2.5 seconds per itinerary for TIMEXD and LABSET respectively, well within the acceptable range for such networks.

17. KEY WORDS *(six to twelve entries; alphabetical order; capitalize only the first letter of the first key word unless a proper name; separated by semicolons)*

Algorithms; algorithm testing; mass transit; routing; shortest paths; transit; transit information systems; transit routing; transportation; urban transportation.

18. AVAILABILITY	19. SECURITY CLASS (THIS REPORT)	21. NO. OF PAGES
[X] Unlimited [ ] For Official Distribution. Do Not Release to NTIS [ ] Order From Sup. of Doc., U.S. Government Printing Office Washington, D.C. 20402, SD Cat. No. C13 [X] Order From National Technical Information Service (NTIS) Springfield, Virginia 22151	UNCLASSIFIED 20. SECURITY CLASS (THIS PAGE) UNCLASSIFIED	163 22. Price $8.00

USCOMM-DC 29042-P74