MPC REPORT NO. 91-5

TECHNIQUES FOR RURAL TRANSIT SERVICE DESIGN

Dr. Yorgos J. Stephanedes, P.E.
and
Nikos V. Vairamidis

Department of Civil and Mineral Engineering
University of Minnesota
Minneapolis, Minnesota 55455

November 1991

Acknowledgement

This report has been prepared with funds provided by the United States Department of Transportation to the Mountain-Plains Consortium (MPC). The MPC member universities include North Dakota State University, Colorado State University, University of Colorado at Denver, University of Minnesota, University of Wyoming, and Utah State University.

The authors would like to thank the staff of the Transit Section of the Minnesota Department of Transportation for making available the data needed for this study. Thanks is also extended to the staff of the Center for Transportation Studies, Department of Civil and Mineral Engineering, University of Minnesota, for their kind assistance. We also thank Dick Braun whose efforts made this project possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

EXECUTIVE SUMMARY

Transit service is playing an increasingly vital role in maintaining and improving the mobility and economic well-being of rural populations in the North Central region. This has been particularly important as the decline in several small towns has led to a dependency on regional centers or metropolitan areas for services formerly provided locally. While rural transit systems have become more important, higher operating expenses and reduced federal subsidies have made state transit management, design and funding decisions more complex. This project has developed methods and tools to aid these important decisions.

In particular, to facilitate the task of designing and funding transit service by transit managers and funding agencies respectively, this work explicitly includes performance indicators in a simulation method which provides forecasts of transit system efficiency and effectiveness over time. The method is to be used as a quick turn-around design tool at two levels: (a) at the managerial level to provide help in service design and operation, and (b) at the fund allocation level to help in funding decisions.

To be effective, this tool must be "transferable," i.e., it must be applicable across rural areas without the need for substantial data collection. Techniques were developed to improve the potential for transferability of the simulation method, to facilitate its implementation across rural areas of the North Central region. The resulting improvements in transit funding decisions and access of rural populations to employment and other services can, therefore, be accomplished at low cost and with attention to the individual characteristics of each rural area in a cost-effective manner.

A general analysis framework which describes the dynamics of causal factors over time was used in this study. A transit system, at any given point can be characterized by

the classical economic paradigm of the supply-demand relationship, where supply and demand share a common equilibrium point. Over time, however, environmental factors and policy changes can cause dramatic shifts in the equilibrium point for the system. In addition, the time needed for the system to exhibit equilibrium behavior is dependent on the overall system delay. The magnitude of this delay depends on four individual delays, each of which lasts from four months to one year. These delays were identified during case studies of rural transit systems in Minnesota and elsewhere in the U.S.: Vehicle Acquisition Delay, Schedule Change Delay, Subsidy Award Delay, and Ridership Information Delay.

The basic interrelationships among the transit system components were identified using information gathered from case studies and from a detailed review of the literature on rural transit systems. Cause-effect relationships were then quantified and a computer simulation of the system behavior over time was employed. Using the computer simulation, of the long and short term effects of different policies on rural transit productivity and efficiency transit decision makers can perform extensive sensitivity analysis to determine the relative policy efforts prior to implementation. Examples of policies that can be tested include: fuel price increases and operating and design strategies such as fleet size, vehicle utilization and service area changes.

More specifically, the model employed in this work, SOLON2, is broken down into two sectors: Demand and supply. The demand sector is divided into two subsectors according to travel mode chosen: Transit Ridership and Auto. SOLON2 uses previously developed disaggregate specifications for demand estimation. One advantage of disaggregate specifications over aggregate ones is that they use, more efficiently, the variability present in the data. An additional advantage is that they have shown a

transferability over time. Further, a set of differential equations is developed with the disaggregate demand specifications as components. Variables appropriate for analyzing and designing policies of interest to the intended clients are used, rather than descriptive variables. It is thus possible for the model to be used directly for service design. With the inclusion of as many endogenous policy variables as possible, a number of feedback effects relevant to current and future policy-making in transportation can be analyzed.

Feedback effects can be analyzed in both the demand and supply sectors. In particular, the SOLON2 supply sector includes two subsectors - services supplied and resources. The demand sector also includes two parts - transit demand and resources for demand. Transit demand is a function of rider resources (e.g., annual household income and auto ownership) and transit service supplied (e.g., travel time, waiting time and fare). As transit usage grows, users' resources change to reflect this travel choice. If transit usage grows for work trips, auto availability is higher for other trips made by the household.

Transit ridership depends on the system characteristics and on the socio-economic characteristics of the population in the service area, as described by a logit travel-demand model. The relations that govern the transit supply sector can be conveniently altered to represent particular managerial policies. While transit ridership expands or decays, depending on the level of transit service available, it is limited by the system capacity. The load factor value at which the manager is content to keep the system capacity and supply unchanged, the "operating point," plays an important role in the total system behavior.

One additional feedback effect, particularly noticeable in rural areas, is the importance of system awareness and acceptability by the community. Traditional ways of

life, scarcity of convenient transportation alternatives and poor information sources for locating transportation systems are factors that contribute to making the rural resident reluctant to accept new alternatives. These factors are accentuated by advanced age of rural residents, and weak economic conditions in many rural communities. System acceptability increases the longer a transit system operates in the area, and the greater its share is in the work population. Delays may increase in areas where previous attempts to operate a transit system failed.

Data initialization is the first step in applying SOLON2. The data requirements include conventional demographic, socioeconomic, and trip information, readily available to transport planners. Additional data, associated with the dynamic nature of the method, include the information and management time constants, and the initial values of ridership and frequency. Performance guidelines, indicating how transit service should be modified in response to performance changes, are declared by the user. The SOLON2 user can update this policy, if desired, after the performance evaluation results have been reviewed.

Following the simulation, the route performance measures are determined and the demand is estimated. A record of the ridership, frequency, and other performance indicators is kept in the form of tables and plots. These are then available to the user interactively over the time horizon or at any other instant specified by the user. Having inspected the selected printouts and plots of route performance indicators for the policy being evaluated, the user can indicate policy modifications and compare the resulting route performance against the base case.

SOLON2 has been implemented in a way that allows easy access by decision makers with little or no computer experience. As a result, it enables experienced policy

analysts to examine expected performance improvements in greater depth by experimenting with a wide range of plans prior to field application. Further, it can substantially aid personnel training. Finally, recommendations for improvements, made by users, are being incorporated in the current version of the method, so that it continues to be responsive to the changing needs of transit decision makers.

CHAPTER 1

INTRODUCTION

Transit service is playing an increasingly vital role in maintaining and improving the mobility and economic well-being of rural populations in the North Central region. This has been particularly important as the decline in several small towns has led to a dependency on regional centers or metropolitan areas for services formerly provided locally. While rural transit systems have become more important, higher operating expenses and reduced federal subsidies have made state transit management, design and funding decisions more complex. This project has developed methods and tools to aid these important decisions.

In particular, to facilitate the task of designing and funding transit service by transit managers and funding agencies respectively, this work explicitly includes performance indicators in a simulation method which provides forecasts of transit system efficiency and effectiveness over time. The method is to be used as a quick turn-around design tool at two levels: (a) at the managerial level to provide help in service design and operation, and (b) at the fund allocation level to help in funding decisions.

To be effective, this tool must be "transferable," i.e., it must be applicable across rural areas without the need for substantial data collection. Techniques were developed to improve the potential for transferability of the simulation method, to facilitate its implementation across rural areas of the North Central region. The resulting improvements in transit funding decisions and access of rural populations to employment

and other services can, therefore, be accomplished at low cost and with attention to the individual characteristics of each rural area in a cost-effective manner.

This work addresses problems faced by both transit managers and funding agencies. Such problems have been identified through interaction with Mn/DOT decision makers and during a review of rural transit systems in Minnesota and elsewhere in the Upper Midwest. These problems include determining the possible impacts of reduced subsidies on service, ridership and other performance measures; and addressing the need for selective service cutbacks that are due to funding restrictions.

Transit system problems may be divided into two categories depending on the time during the system's life that they appear - problems related to project initiation and those related to project survival.

Initiation Problems

During the start-up period, transit managers apply for initial funding and formulate plans based on socioeconomic and demographic characteristics of the service area. Managers must design routes and schedules, predict ridership, seat miles and revenues based on data such as terrain, service area and population in the service area. These predictions are difficult to make with only the data that are generally available.

During the initial period, subsidies for rural transit projects may be allocated. Applications are expected to present evidence of the ways in which the proposed transit system would meet the area's needs. In the process, the funding agency, such as Mn/DOT, must make a judgement on the relative merit of alternative proposals. When essential data are not readily available, Mn/DOT may use the performance of existing systems to infer the candidate system's future performance. However, unless the whole

spectrum of variables essential in determining the behavior of a rural transit system is examined, such comparisons may produce misleading predictions.

The above problems are further complicated by the tight time schedules within which funding agencies require performance results from operating systems in order to make funding and budget size decisions. Because of resulting time constraints, transit decision makers may use single average values describing system performance to make decisions about the long-term viability of rural transit operations. Such values may then be compared at the national or state level and decisions made about whether a system's performance is "acceptable" or "unacceptable." Potential shortcomings of this procedure are illustrated in Figures 1 and 2. Figure 1 is an example of data from a seemingly unacceptable system. Figure 2 is an example of an apparently acceptable system. Notice that Figures 1 and 2 actually describe the ridership of the same system but Figure 1 contains data from only the first seven months of system operation, while the data in Figure 2 includes 18 months of operation. From the figures the transit system is still expanding and its performance values continue to change with time, i.e., one year has not been enough for them to reach equilibrium. In general, the time needed for the system to exhibit equilibrium behavior is dependent on the overall system delay.

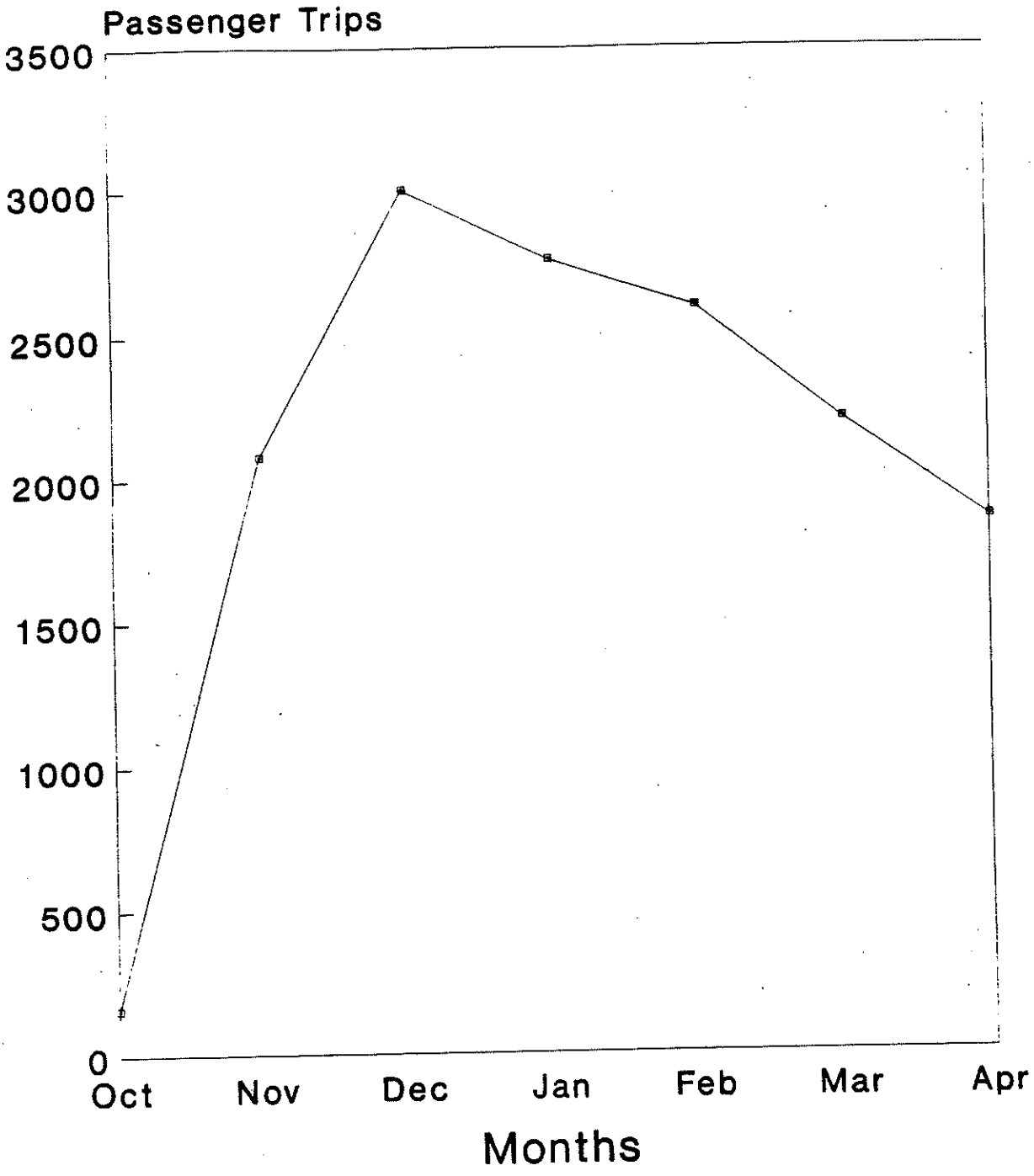


Figure 1
Bus Ridership in System A as of
the Seventh Month of Operation

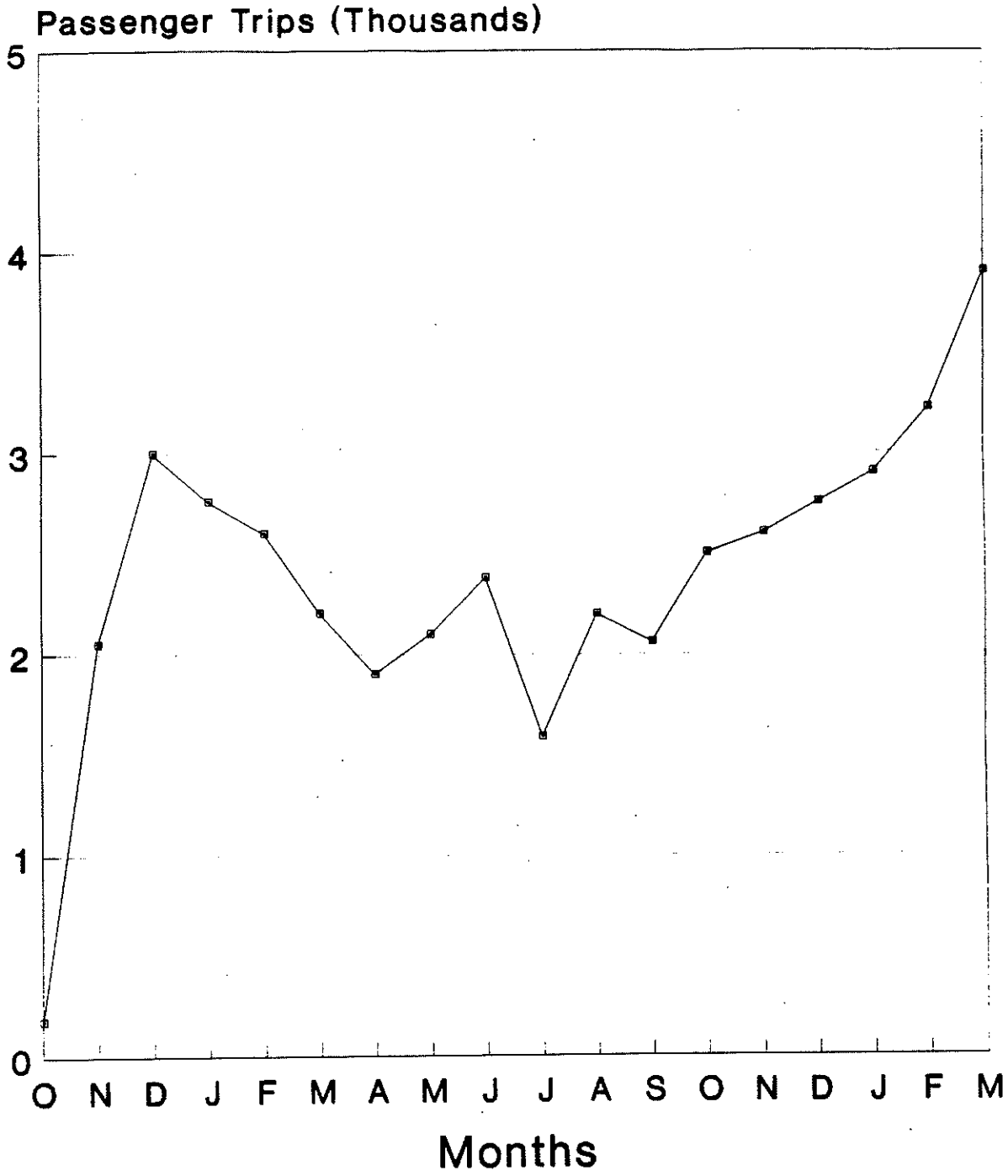


Figure 2
Bus Ridership in System A as of
the 18th Month of System Operation

The magnitude of this overall system delay depends on four individual delays, each of which lasts from four months to one year. These delays were identified and fully documented during case studies of rural transit systems in Minnesota and elsewhere in the U.S. (See references (1) to (11)):

(1) Vehicle Acquisition Delay

(2) Schedule Change Delay

(3) Subsidy Award Delay

(4) Ridership Information Delay

Transition-Stage Problems

At the end of the first or second year of funding, the system's performance is usually reviewed for funding continuation. The benefit-cost standards a local community applies to the expenditure of federal or state subsidies can be somewhat different than for local subsidies. The former are considered to be marginally free, and thus, the accrual of any form of benefit is a net gain to the community. In most cases, this implies that the effectiveness of state subsidies depends heavily on an operator's internal evaluation of the service and/or on external evaluation and monitoring by the allocating agency.

During the transition stage in a life of a rural transit system, performance indicators usually reach values termed acceptable by management and remain relatively unchanged with time. Management may then consider penetrating new market segments or increasing transit use through advertising or by improving service. Any such plans for improvement or expansion depend on adequate and timely funding. Long delays, however, impair the smooth transition from one stage to the other and result in lost time and revenue for the transit system.

CHAPTER 2

GENERAL APPROACH

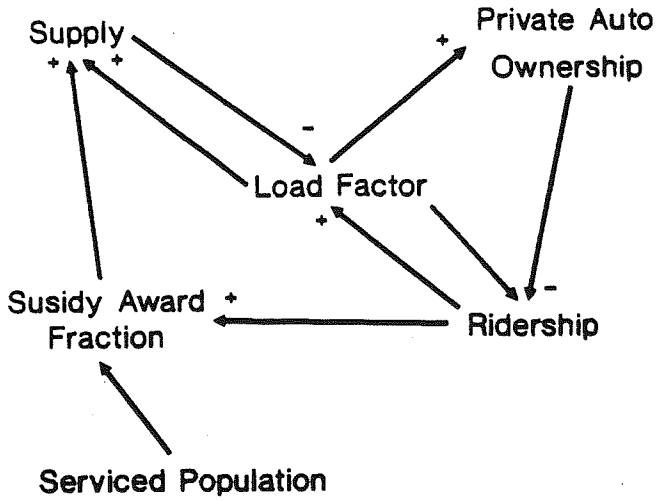
A general analysis framework which describes the dynamics of causal factors over time was used in this study. A transit system, at any given point can be characterized by the classical economic paradigm of the supply-demand relationship, where supply and demand share a common equilibrium point. Over time, however, environmental factors and policy changes can cause dramatic shifts in the equilibrium point for the system.

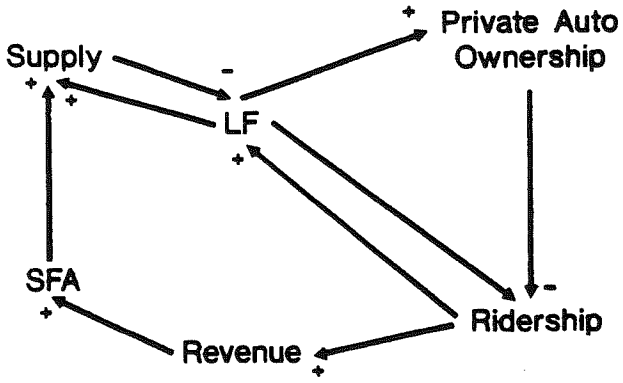
The types of behavior that can characterize a transit system include a downward spiral where reduced ridership (caused by various factors, particularly increased auto ownership) can cause transit operators to reduce level of service (e.g., longer headways, higher fares) which, in turn, results in further ridership reductions. This downward spiral is one type of system behavior now widely recognized by transportation system analysts. There are, however, many other possible modes of behavior under which rural transit systems may alternatively flourish, exist in a marginally productive state or cease operation. High levels of capital assistance can cause over-capitalization in the near term, with inadequate resources set aside by the operations for future capital stock replacement. Fuel price increases could cause a dramatic increase in the viability of rural transit systems, or, for systems operating at low load factors, could force some out of business. (For efficient systems, higher ridership will lead to higher frequency of inefficient service and a larger number of money-losing bus miles, and this will increase net financial losses.)

The basic interrelationships among the transit system components were identified in this work using information gathered from case studies and from a detailed review of the literature on rural transit systems. Cause-effect relationships were then quantified and a computer simulation of the system behavior over time was employed.

Using the computer simulation, of the long and short term effects of different policies on rural transit productivity and efficiency transit decision makers can perform extensive sensitivity analysis to determine the relative policy effects prior to implementation. Examples of policies that can be tested include: fuel price increases and operating and design strategies such as fleet size, vehicle utilization and service area changes. The structure of the simulation allows easy testing of alternative policies, as demonstrated by the example in Figure 3. This figure describes a subsidized system, which is allocated more funds only if the funding agency sees a substantial amount of ridership being attracted. Seat availability and system visibility are two of the factors which influence ridership in the short run. Auto ownership is a long run variable. In this example, ridership is an index of performance for the funding agency and load factor (defined as demand/supply) is an index of performance for the system manager and riders. Note the flexibility of the structure that allows testing of other possible measures, e.g., ridership-to-serviced-population ratio, revenue, and revenue per passenger.

As another example, consider the effects of a possible energy price increase on transit ridership (Figure 4). While it is not clear whether an alternative method would be able to model this exogenous effect, it can be modeled in a straightforward manner by a closed loop dynamic model as long as its occurrence is hypothesized within the model's time horizon.





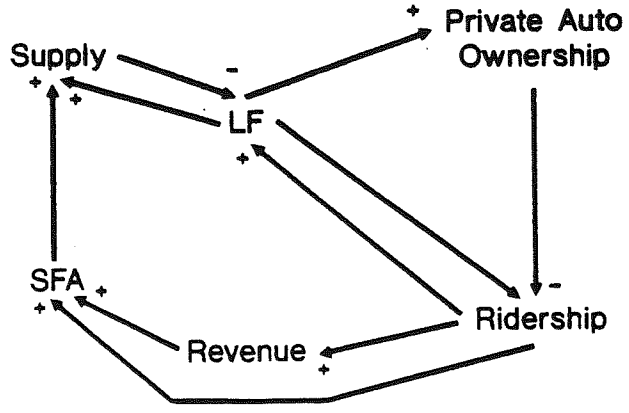


Figure 3
Example of Subsidized Rural System

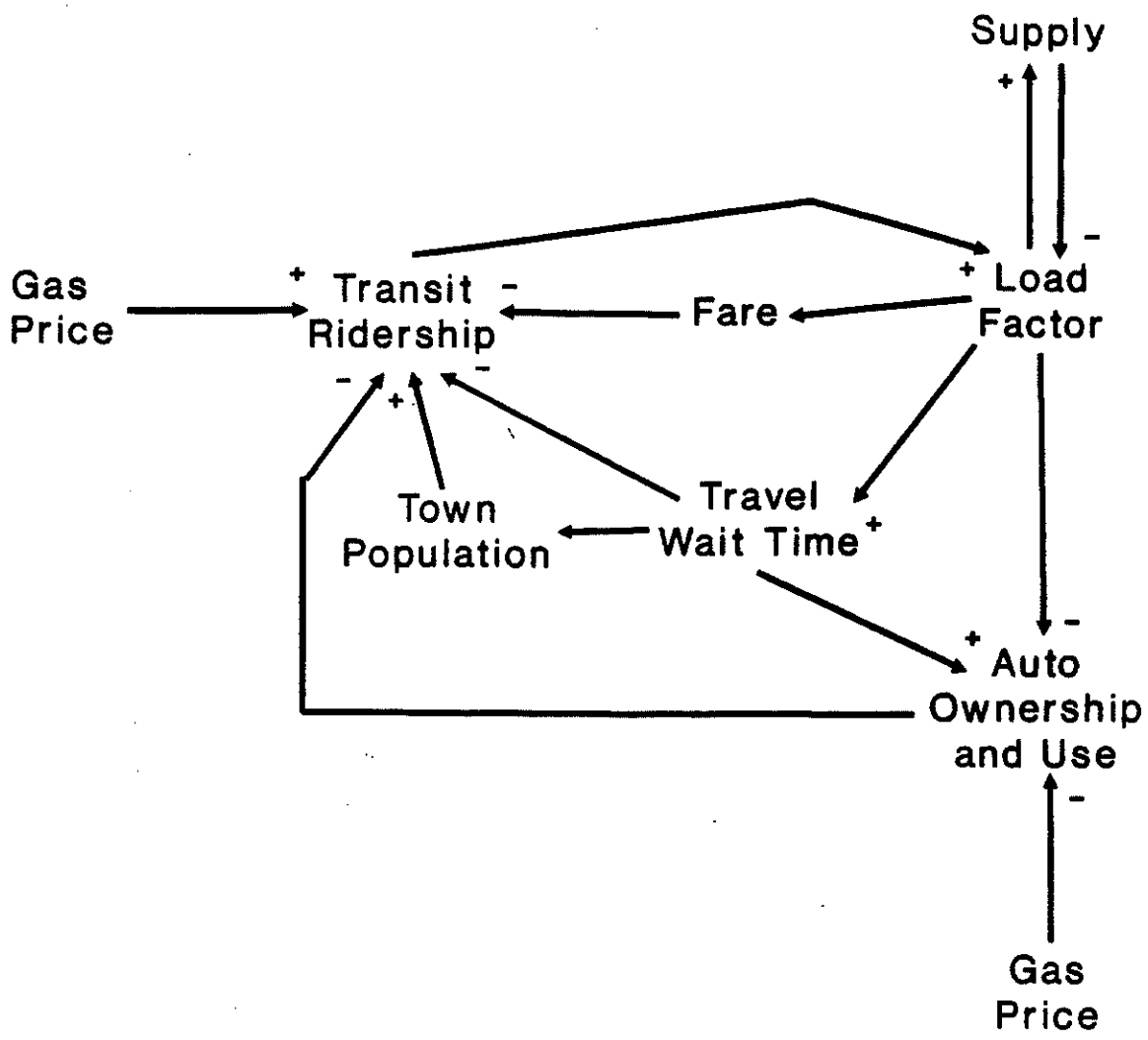


Figure 4
Gas Price Effect on Transit Ridership:
Alternative Modeling Methods

CHAPTER 3

A DYNAMIC MODEL FOR THE DESIGN OF RURAL TRANSIT OPERATIONS

The model employed in this work, SOLON2, is broken down into two sectors:

A. Demand sector

B. Supply sector

The demand sector is divided into two subsectors according to travel mode chosen:

(a) Transit ridership

(b) Auto users

SOLON2 uses previously developed disaggregate specifications for demand estimation. One advantage of disaggregate specifications over aggregate ones is that they use, more efficiently, the variability present in the data. An additional advantage is that they have shown a transferability over time. However, they are usually applied in a static framework. Since this model will be used to analyze transportation services as they change through time, a dynamic structure is needed. To accomplish this, a set of differential equations is developed with the disaggregate demand specifications as components in the overall structure. The approach used here is based on selecting acceptable causal a priori hypotheses. It draws on credible theories of consumer behavior rather than exploiting simple correlations among variable.s Supply, resource and demand sectors are included in a time-dynamic framework.

Variables appropriate for analyzing and designing policies of interest to the intended clients are used, rather than descriptive variables. It is thus possible for the model to be used directly for service design. With the inclusion of as many endogenous

policy variables as possible, a number of feedback effects relevant to current and future policy-making in transportation can be analyzed. As a result, there is not a need for compromising model building toward self-equilibrating recursive progressions. Instead, the causal dynamics are used for continuous forecasting through time.

SOLON2 has potential applications for policy design at two levels: (a) at the managerial level to provide help in system design and operation, and (b) at the fund allocation level to help in funding decisions. It is useful as a quick turn-around policy design tool.

A Feedback Dynamic Representation of Supply-Demand-Resource Interactions

The basic problem for the transit decision maker is the prediction of equilibrium in the transportation system, which is determined by the pattern of flow in the network. The general structure for predicting the flow pattern is the equilibrium between supply and demand. Depending on the variables that are of interest and the assumptions made, this may be a short-term equilibrium, a long-term equilibrium, or something between the two. For example, the interaction between transportation service and ridership is realized over a shorter period of time than is the interaction between these factors and resources (subsidies). A time-dimension, then, has to be taken into account by the decision maker who intends to capture these interactions in a model. The way in which this time-dimension is conceived is a matter of considerable strategic significance. The choice ranges from comparative statics at one extreme, through various types of recrusive progression, to analytical dynamics at the other extreme.

Self-equilibration is not a necessary assumption for analytical dynamics, an approach which focuses attention on the processes of change rather than on the emergent state of the system at a specific future date. Implementation of a dynamic model requires

only specification of its structural parameters and the "initial conditions" of its variables. Thereafter, all processes are endogenous except time, and the time path of any variable can be continuously traced. The state of the system can be evaluated at any point in time. If the system is self-equilibrating, the values of its variables should converge on those indicated by analogous comparative statics; but without self-equilibrating properties, the system may fluctuate cyclically, explode, or degenerate.

The model employed here, SOLON2, is a dynamic representation of the supply-demand-resource interactions in a transportation system. The structure of SOLON2 can be analyzed at several levels of detail. At the most general level, it may be viewed as a simple demand-supply model (see Figure 5). Given the transportation services provided by the supply sector, the demand sector reacts and expresses a need for transportation services. The demand sector reacts to supply changes relatively quickly, but the reaction time of the supply sector to demand changes may be much longer.

At a more detailed level, the supply sector includes two subsectors - service and resources (see Figure 6). As transit demand grows, supply resources also grow, but transit service degrades (ceteris paribus). Service may improve, however, if appropriate managerial action is taken. As more service is supplied by the transportation system, its resources are depleted unless increased subsidies or increases in farebox revenue are received. As resources are depleted, the service supplied decreases and demand for transit also decreases.

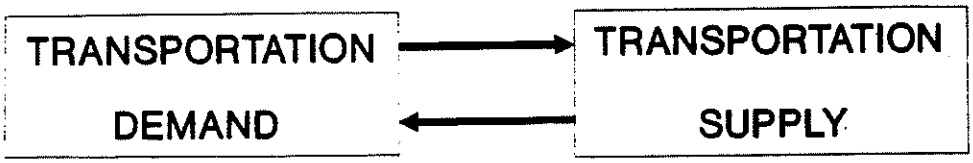


Figure 5
Basic SOLON2 Structure

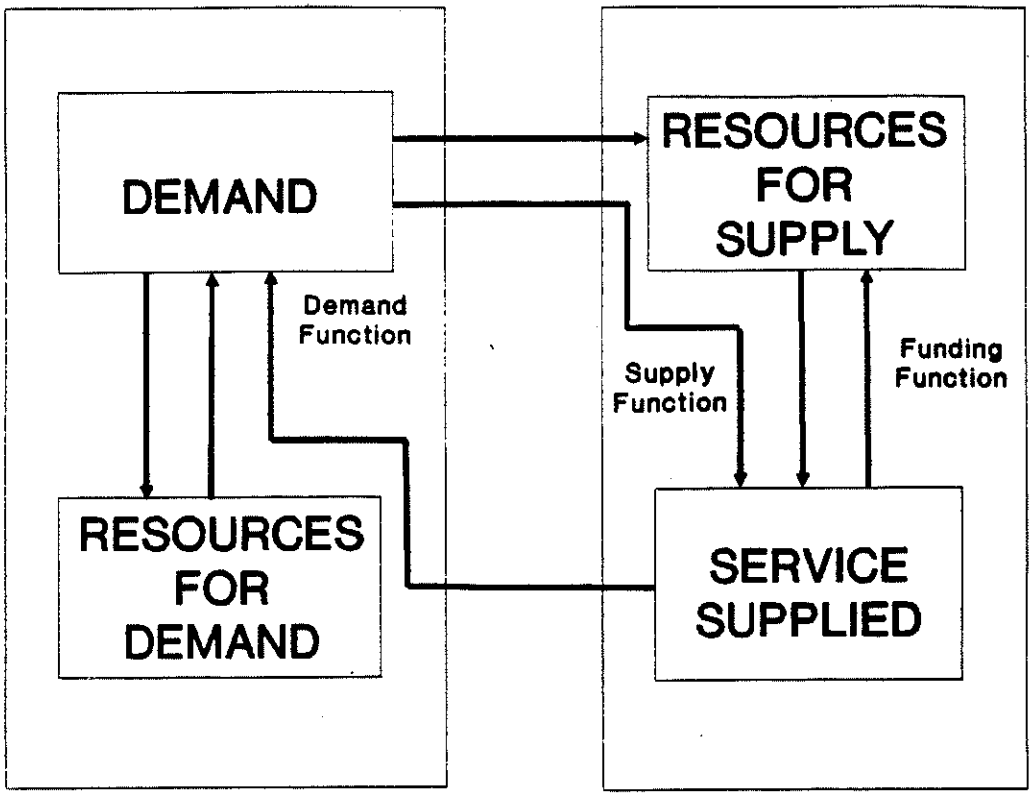


Figure 6
Sector Organization of SOLON2

The demand sector also includes two parts - transit demand and resources for demand. Transit demand is a function of rider resources (e.g., annual household income and auto ownership) and transit service supplied (e.g. travel time, waiting time and fare). As transit usage grows, users' resources change to reflect this travel choice. If transit usage grows for work trips, auto availability is higher for other trips made by the household.

In the next sections, a further breakdown of these sectors illustrates the relationships among their variables and parameters in greater detail.

CHAPTER 4

TRANSIT SECTORS

(A) Demand Sector: Transit Ridership Subsector

The primary assumption of the transit demand sector is that transit service characteristics and transit ridership are fundamentally interrelated. Transit ridership depends on the system characteristics and on the socio-economic characteristics of the population in the service area, as described by a logit travel-demand model.

The multinomial logit formulation itself is described in many references and is expressed as follows:

$$P(i:A_t) = \frac{e^{x_{it}' b}}{\sum_{j \in A_t} e^{x_{it}' b}} \qquad (1)$$

where:

$P(i:A_t)$ = the probability of behavioral unit t selecting alternative i from its choice set A_t

X_{it} = a vector of independent variables for alternative i and behavioral unit t. The coefficients b are estimated using the maximum likelihood method.

The variables included in the work trip logit model are listed in Table 1. The model was tested and shown to be transferable to areas of differing characteristics. Using the logit model, choice probabilities may be computed, given the two alternatives of auto

and transit. The number of trips demanded per week is then found as a function of information delay, according to the following equation:

$$\frac{d}{dt}(Demand) = \frac{Estimated\ Demand - Demand}{Work\ Trip\ Rider\ Information\ Delay} \quad (2)$$

Table 1

Work Mode Choice Model: Definition of Variables

Variable Code	Definition
1. IVT_A	= One way trip in-car travel time (in minutes)
2. IVT_B	= One way trip in-bus travel time (in minutes)
3. OVT_B	= One way trip out-of-bus travel time (includes wait time and walk time, in minutes)
4. EGRESS_C	= Egress time for car users (includes walk time, in minutes)
5. EGRESS_T	= Egress time for bus users (includes walk time, but does not include wait time, in minutes)
6. PARKCOST_C	= Daily parking cost for car users (in dollars)
7. AUTOPH	= Number of cars per household
8. PERSPH	= Number of persons per household
9. INCOME	= Annual household income

where Estimated Demand is directly computed from the choice probabilities and the number of people in the market segment under consideration. Because of the Rider Information Delay, however, actual demand approaches but does not necessarily equal Estimated Demand.

(b) Auto Subsector

The differential equation that computes the number of people using auto is:

$$\frac{d}{dt}(Auto\ Users) = \frac{Estimated\ Auto\ Users - Auto\ Users}{Work\ Trip\ Information\ Delay} \quad (3)$$

(c) Supply Sector

The relations that govern the transit supply sector can be conveniently altered to represent particular managerial policies using this model. In simplified form, the differential equation for transit service frequency, as a function of the policy management follows regarding frequency of service, is:

$$\frac{d}{dt}(Frequency) = \frac{Desired\ Frequency - Frequency}{Frequency\ Change\ Delay} \quad (4)$$

where Desired Frequency is a managerial decision, limited by the number of buses and the travel time and dependent on the Load Factor or some other performance indicator, through a non-linear relationship specified when running the model.

CHAPTER 5

APPLICATION TO A RURAL TRANSIT SERVICE

5.1 Service Characteristics and Transit Demand Growth

Transit ridership is controlled in part by the level of service offered in relation to other modal alternatives. As service improves, ridership increases and significant improvements may cause dramatic changes in ridership. Past a critical point, however, indicated by the logistic form of the demand model, service improvements are no longer as effective. Furthermore, the increase in ridership eventually results in service degradation because crowding on the vehicle tends to slow ridership growth.

Level of service is a general term which encompasses the major service characteristics. Some of these characteristics are used as variables and some as parameters. The choice depends on (1) the importance of the relationship between independent and dependent characteristics, (2) the insights gained by choosing a particular representation, and (3) the relative rate of change of a characteristic during the time period under consideration.

The causal relationships described above are illustrated in Figure 7. By breaking down Figure 7 into is component loops, one can recognize the important causal relationships more easily (see Figures 7.1 - 7.4).

In Figure 7.1, high transit demand causes the transit system manager to offer more service which may attract increased demand. Conversely, low demand causes the manager to decrease service frequency, which results in lower demand. If not controlled, this loop will drive the system toward continuing expansion or decay, although at a slow

pace due to the two delays present. For example, previous work indicates that it takes approximately five months for demand to form in response to some transit service offered (see references (1) to (11)). The control needed is provided by the loop in Figure 7.2, i.e., the transit system capacity.

The loop in Figure 7.2 controls the system expansion or decay. The transit manager changes the transit frequency as a result of a change in some performance indicator, e.g., the Load Factor, where:

$$\textit{Load Factor} = \frac{\textit{Ridership Demand Seat Miles}}{\textit{Capacity Seat Miles}} \qquad (5)$$

It may take six months for a transit system to obtain permission to bring about a radical schedule change.

The loop in Figure 7.3, if acting alone, causes continuous system expansion or decay. AS RPBM increase, transit travel time also increases, and transit demand decreases. This causes a decrease in transit supply offered and a further RPBM increase. This effect would become more important for mixed ridership, shopping and work trips or general and elderly clientele riding on the same bus. The loop presented in Figure 7.4 acts together with the previous loop to bring the system to an equilibrium.

5.2 Managing Capacity and Supply

While transit ridership expands or decays, depending on the level of transit service available, it is limited by the system capacity. It is, therefore, necessary to consider the rules governing management decisions for changing that capacity. These policies are highly dependent on the individual manager, but, in general, the Load Factor is taken into consideration through one or more non-linear relationships. A high load factor

influences management to expand the system capacity. This, in turn, affects the loaf factor and tends to make further expansion unnecessary. The load factor value at which the manager is content to keep the system supply unchanged, the "operating point," plays an important role in the total system behavior. For example, the operating point in Figure 8 is 50 percent. The manager usually sets a Lower Limit under which Frequency is not allowed to fall. The Lower Limit is dependent upon the number of buses which are in running condition. Capacity changes usually take about six months to take effect. This delay has been observed in rural programs across the country, including Minnesota.

Table 2

Transit Service Variables and Parameters

Service Variables	Abbreviation
Transit Frequency	FREQ
Riders per Bus Mile	RPBM
Service Parameters	**Abbreviation**
Transit Out-of-Pocket Travel Cost	TOPTC
Transit Walking Time	TWLKT
Number of Stops per Mile	STOPPM
Stop Time per Rider	STPR

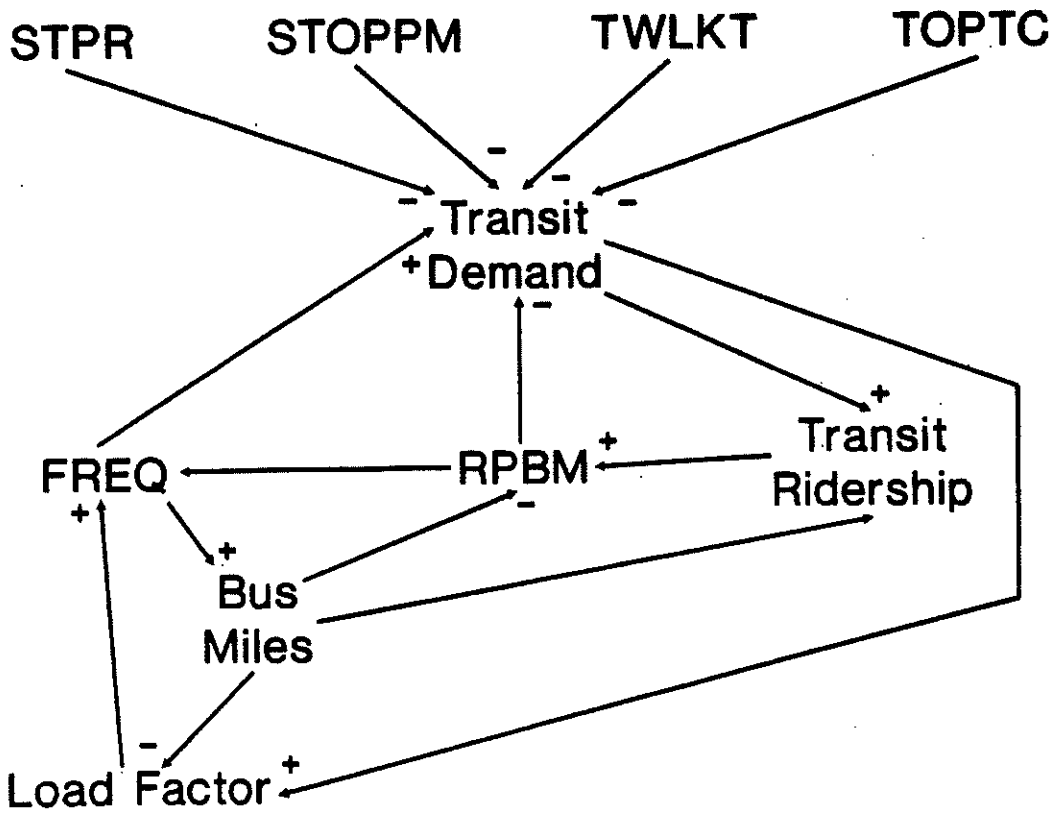


Figure 7
Service Characteristics and Transit Demand Growth

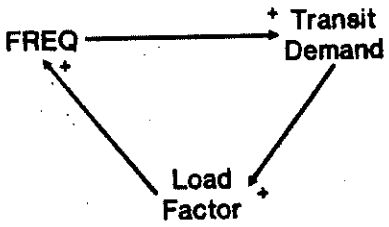


Figure 7.1

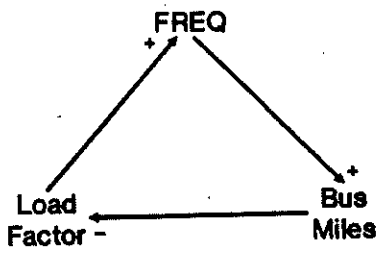


Figure 7.2

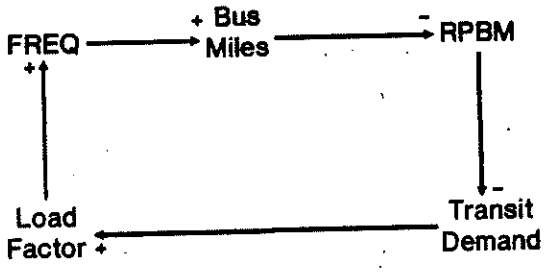


Figure 7.3

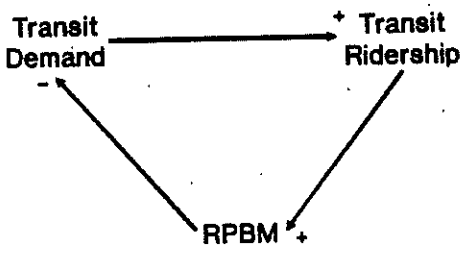


Figure 7.4

Breakdown of Figure 7

5.3 Rural Way of Life and Community Acceptance

One additional feedback effect, particularly noticeable in rural areas, is the importance of system awareness and acceptability by the community. Traditional ways of life, scarcity of convenient transportation alternatives and poor information sources for locating transportation systems are factors that contribute to making the rural resident reluctant to accept new alternatives. These factors are accentuated by advanced age of rural residents, and weak economic conditions in many rural communities. System acceptability increases the longer a transit system operates in the area, and the greater its share is in the work population. Delays may increase in areas where previous attempts to operate a transit system failed.

The acceptability delay is crucial to effective transit service design in rural areas. Failing to take it into account will result in over-optimistic ridership demand estimates. Funding agencies are also aware of this delay when considering the reduction of funding in systems that do not show quick progress.

5.4 Performance Indicators in SOLON2

Previous sections in this chapter and a detailed software listing in the Appendix describe how the dynamic model uses initial input data to determine the behavior of rural transit operations across time. The final section of the model is a synthesis of the variables with values forecast. The final output of the model is presented in terms of performance measures. A list of indicators available by the present version of the software is found in Table 3, following:

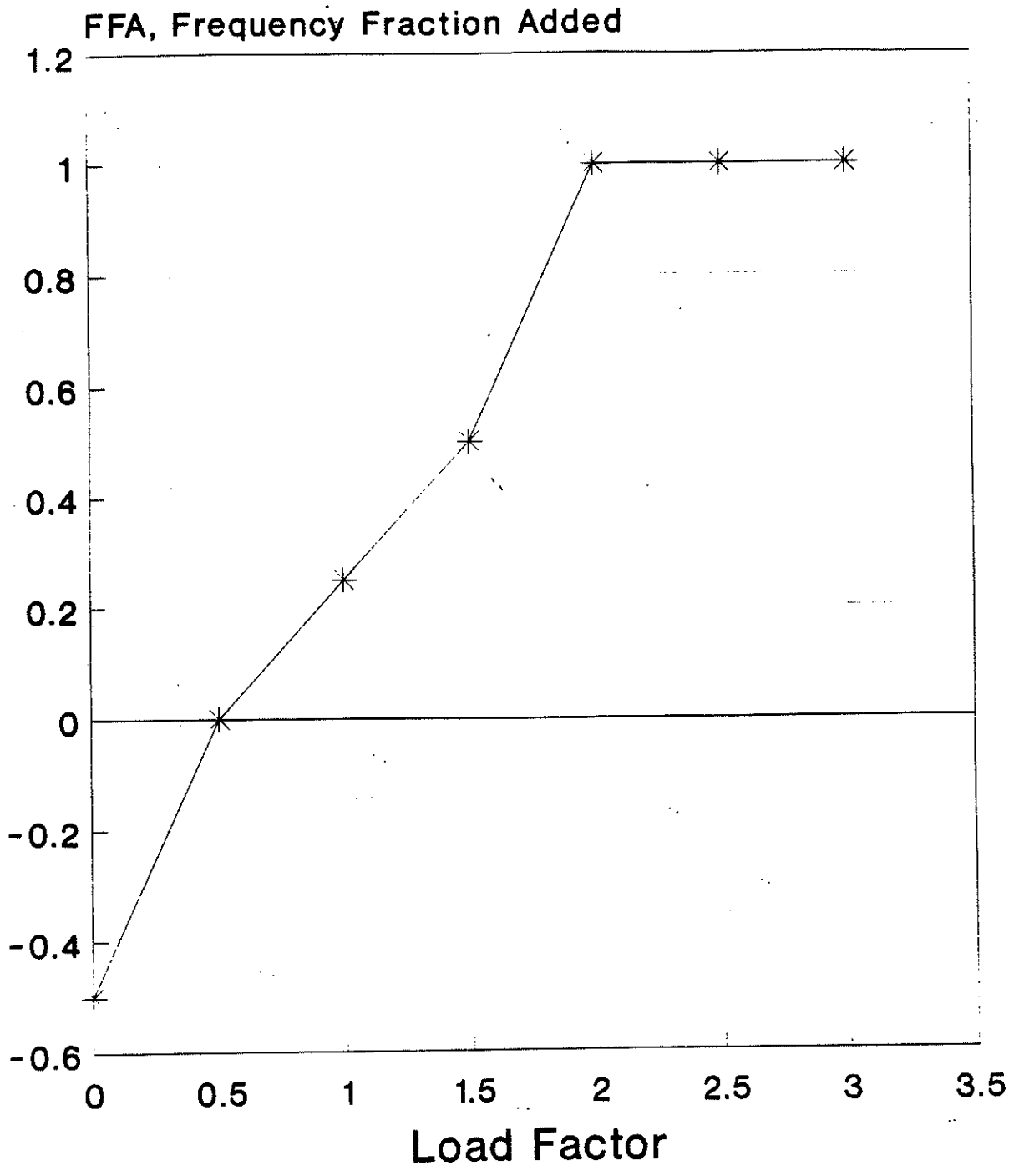


Figure 8
Frequency Change in Response to
Load Factor

TABLE 3

Performance Indicators for Transit - Weekly

Headway

Ridership

Load Factor

Frequency

Cost

Revenue

Profit

Revenue/Cost

CHAPTER 6

APPLICATION: POLICY EVALUATION AND DESIGN

The basic process for evaluating and selecting route performance policies using SOLON2 is now illustrated through application of the dynamic method to peak-period work trips on a radial route along a rural corridor south of a town central business district. This route is characterized by local stops at the two ends of the trip and virtually no stops in the middle portion. Census data and traffic information are routinely available to transit decision makers who are potential users of the method.

Data initialization is the first step in applying the policy decision method. The data requirements include conventional demographic, socioeconomic, and trip information, readily available to transport planners. Most of these data are required by the logit specifications that are built in SOLON2. Additional data, associated with the dynamic nature of the method, include the information and management time constants, and the initial values of ridership and frequency.

Performance guidelines, indicating how transit service should be modified in response to performance changes, must next be declared by the user. In this application, for instance, the transit manager follows a policy that states that there should be no frequency change for round-trip load factors that equal 0.5; below this value, frequency should decrease, whereas above it (i.e., when there are standees), it should increase. The SOLON2 user can update this policy, if desired, at a later stage after the performance evaluation results have been reviewed.

Following initialization, SOLON2 begins operation by computing the initial values of the route performance indicators. Load factor, operating ratio, and net revenue are key indicators in this application because they determine the service frequency modifications. If the load factor (LF) exceeds the maximum value (LF_{max}) allowed by the size of the transit vehicles, then $LF = LF_{max}$ and SOLON2 signals the need for more service. If LF is below LF_{min}, the minimum value allowed by the operating ratio constraints, SOLON2 signals that present service conditions should be reevaluated. Fulfillment of additional performance criteria, critical to service change decisions, may also be similarly determined.

The next step is activated only at regular time intervals, the length of which is controlled by the user. These indicate the points in time when transit management makes service change decisions. If current resources are adequate, frequency change is initiated on the basis of the state of the performance indicators determined previously. Owing to implementation time delays, the frequency change initiated in this step can only be implemented after a period of time. This is necessary as a result of regulations or because of needed adjustments in the rolling stock and the number of available drivers. In extreme cases, when the available transit vehicles are not sufficient for satisfying the needed service improvements, additional capital is desired; in this case vehicles may be ordered or drivers may be hired, or both. In such an event, order and acquisition delays intervene and service change decisions may be implemented more than a year after the process is initiated. At the same time, depreciating vehicles are retired and create the need for additional equipment orders.

Following the change in transit service, the route performance measures are determined and the demand is estimated. A record of the ridership, frequency, and other

performance indicators is kept in the form of tables and plots. These are then available to the user interactively beginning with time zero(i.e.,the time SOLON2 is initialized) and ending with the end of the time horizon or any other instant specified by the user.

Having inspected the selected printouts and plots of route performance indicators for the next base policy, the user can indicate policy modifications and compare the resulting route performance against the base case. To illustrate this option, two such policy modifications have been included in Figure 9. In this example, the performance of the transit operation is evaluated based on the operating ratio. In particular, the performance objectives the attainment of operating ratio values greater than 0.9. Two policies are proposed by management for reaching the desired objective. Policy B requires an initial frequency of service equal to 5 buses per hour, and a fare of $2. Policy C offers an initial service of 4 buses per hour at a $3 fare. The dynamic simulation of the two policies indicates that, in both cases, a period of approximately 20 weeks is necessary for the system to reach equilibrium. This overall system delay results from the time needed for trip makers to be informed about the new service, decide to use it, and for management to react to the increasing demand by scheduling the appropriate bus frequency. Of the two policies, only policy C succeeds in increasing the operating ratio so that it enters the desired region, i.e., becomes greater than 0.9. The overplotting can continue and the user can evaluate additional policies until he or she is confident that one or more of the policies under evaluation are superior and can be selected.

The dynamics of the transition of the operation from the time a policy is instituted until the time the bus system reaches equilibrium can be illustrated with SOLON2. Having provided the necessary data, we can apply SOLON2 to the initial conditions of interest, and let it provide us the equilibrium solution and the time over which it is

achieved. By applying SOLON2 multiple times, we can produce a map of the isochrones of the operation. Based on this map, it is sufficient to locate the initial conditions (frequency of service and ridership) to immediately determine the time it will take for equilibrium to be reached. For instance (see Figure 10), for initial frequency equal to 5 buses/hr and initial ridership of 3000 trips/week, the transit operation will reach equilibrium in approximately 20 weeks. Using the isochrone map a transit manager or funding agency can determine, at a glance, both the potential performance of a transit operation and the speed with which that performance can be achieved given a desired policy.

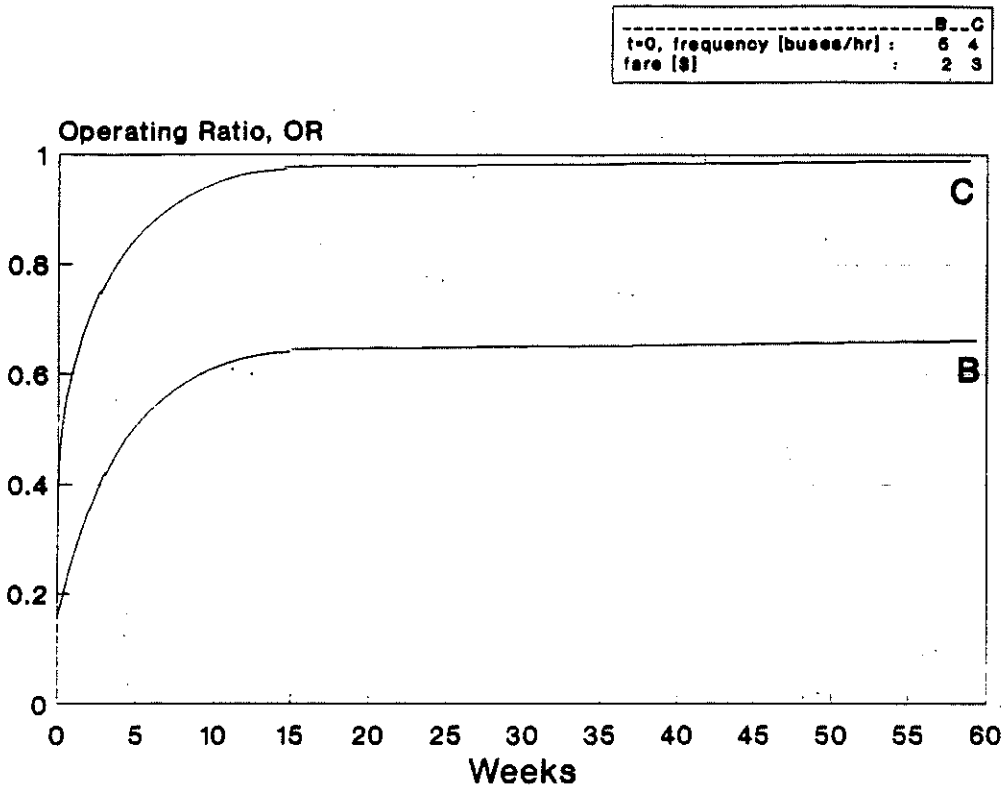


Figure 9
Operating ratio versus time for two
different policies

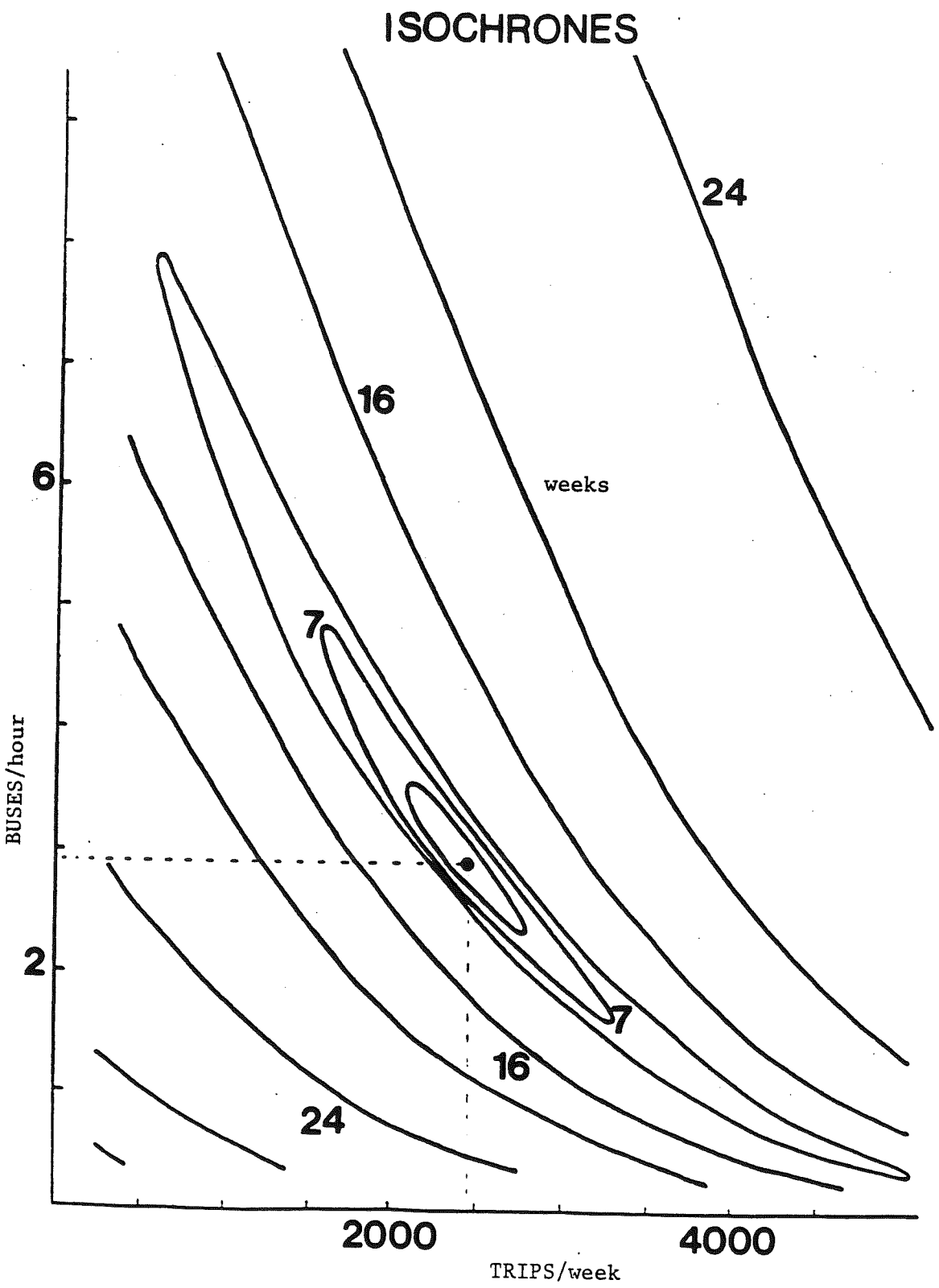


Figure 10
Transit system isochrones

CHAPTER 7

CONCLUSIONS

A dynamic simulation method has been developed that provides forecasts of transit system efficiency and effectiveness over time. The new method explicitly includes performance indicators and can be used as a quick turn-around design tool at two levels: (a) at the managerial level to provide help in service design and operation, and (b) at the fund allocation level to help in funding decisions.

Techniques were developed to improve the potential for transferability of the simulation method, to facilitate its implementation across rural areas of the North Central region without the need for substantial data collection. The resulting improvements in transit funding decisions and access of rural populations to employment and other services can, therefore, be accomplished cost-effectively and with attention to the individual characteristics of each rural area.

The transit framework of the method is based on the classical economic paradigm of the supply-demand relationship, where supply and demand share a common equilibrium point. Over time, however, environmental factors and policy changes can cause dramatic shifts in the equilibrium point for the system. In addition, the time needed for the system to exhibit equilibrium behavior is dependent on the overall system delay. The magnitude of this delay depends on four individual delays, each of which lasts from four months to one year: Vehicle Acquisition Delay, Schedule Change Delay, Subsidy Award Delay, and Ridership Information Delay. The basic interrelationships among the transit system components were identified using information gathered from

case studies and from a detailed review of the literature on rural transit systems. Cause-effect relationships were then quantified and computer simulation of the system behavior over time was employed.

More specifically, the model employed in this work, SOLON2, is broken down into two sectors: Demand and Supply. The demand sector is divided into two subsectors by travel mode chosen: Transit Ridership and Auto. SOLON2 uses previously developed logit disaggregate specifications for travel-demand estimation. One advantage of disaggregate specifications over aggregate ones is that they use, more efficiently, the variability present in the data. An additional advantage is that they have shown a transferability over time. Further, a set of differential equations is developed with the disaggregate demand specifications as components. Variables appropriate for analyzing and designing policies of interest to the intended clients are used, rather than descriptive variables. It is thus possible for the model to be used directly for service design.

With the inclusion of endogenous policy variables in the dynamic model, a number of feedback effects relevant to current and future policy-making in transportation can be analyzed. In particular, transit decision makers can perform extensive sensitivity analysis to determine the relative policy effects prior to implementation. Examples of policies that can be tested include: fuel price increases and operating and design strategies such as fleet size, vehicle utilization and service area changes.

A feedback effect, particularly noticeable in rural areas, is the importance of system awareness and acceptability by the community. Traditional ways of life, scarcity of convenient transportation alternatives and poor information sources for accessing transportation systems are factors that contribute to making the rural resident reluctant to accept new alternatives. These factors are accentuated by advanced age of rural

residents, and weak economic conditions in many rural communities. System acceptability increases the longer a transit system operates in the area, and the greater its share is in the work population. Delays may increase in areas where previous attempts to operate a transit system failed.

SOLON2 data requirements include conventional demographic, socioeconomic, and trip information, readily available to transport planners. Additional data, associated with the dynamic nature of the method, include the information and management time constants, and the initial values of ridership and frequency. The load factor value at which the manager is content to keep the system capacity and supply unchanged, the operating point, also plays an important role in the total system behavior. Performance guidelines, indicating how transit service should be modified in response to performance changes, are declared by the user. The SOLON2 user can update this policy, if desired, after the performance evaluation results have been reviewed.

Following the simulation, the route performance measures are determined and are available interactively over the time horizon or at any instant specified by the user. Having inspected the selected printouts and plots of route performance indicators for the policy being evaluated, the user can indicate policy modifications and compare the resulting route performance against the base case.

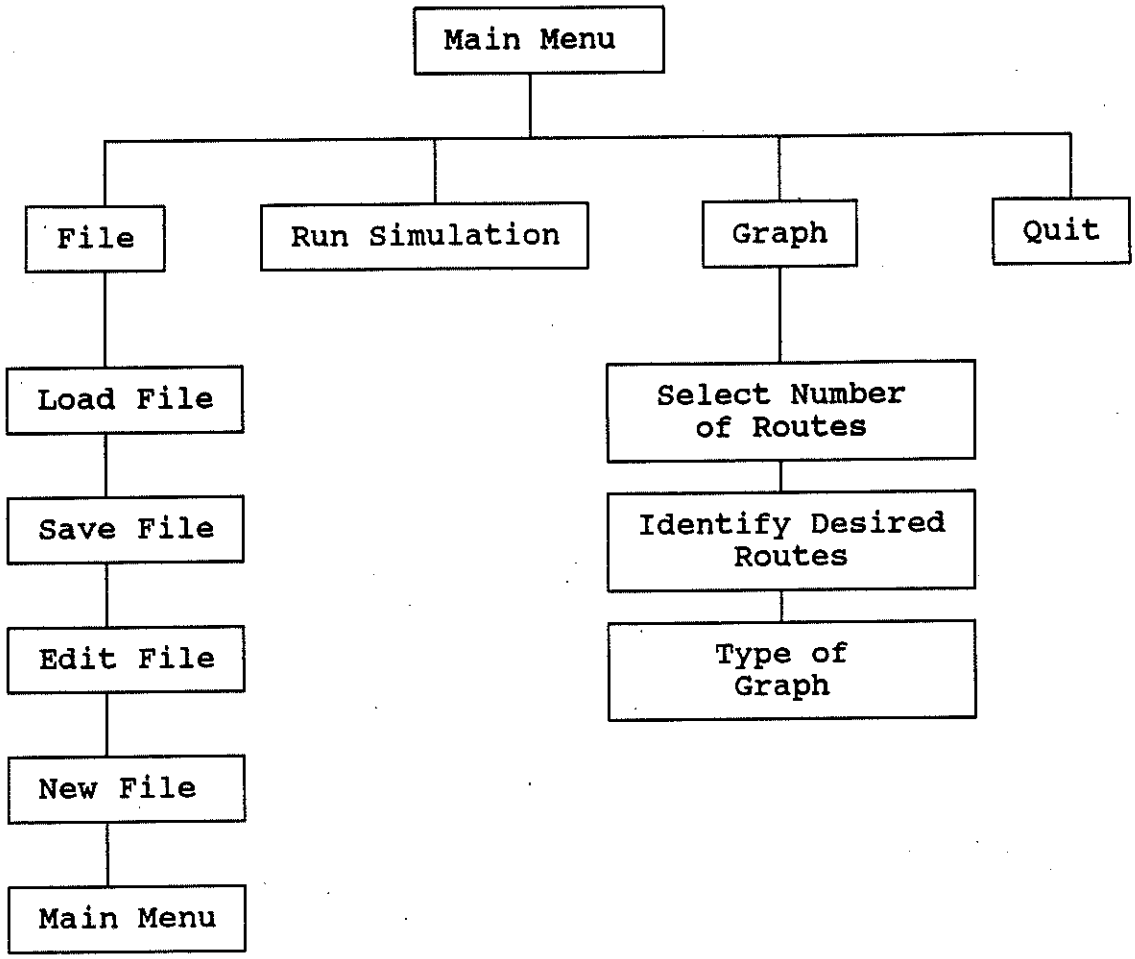
SOLON2 has been implemented in a way that allows easy access by decision makers with little or no computer experience. As a result, it enables experienced policy analysts to examine expected performance improvements in greater depth by experimenting with a wide range of plans prior to field application.

APPENDIX A

STRUCTURE OF SOLON SOFTWARE

SOURCE CODE OF SOLON SOFTWARE

Menu Structure of SOLON2 Software



Glossary

File : Select the File Menu.

Load File : Load a data set from a text file.

Save File : Save the data in a text file.

Edit File : Edit the data in a given text file.

New File : Create a new text file and save the data in this file.

Main Menu : Return to the Main Menu.

Run Simulation : Run the simulation by using the most recent data set.

Graph : Select the Graphics Menu.

Select Number of Routes : The number of routes that will be displayed on the same graph can be selected (max 3 routes).

Identify Desired Routes : Identify the routes that will be displayed (0 - Main Route, 1 ... 12 - Feeder Routes).

Type of Graph : Select the route performance indicator to be displayed over time of simulation (weeks). The available indicators are : Headway, Ridership, Load Factor, Frequency, Cost, Revenue, Profit, Revenue/Cost.

Quit : Quit the program and go to DOS.

Source code of SOLON software

```
/*=============================================================*
 * File : solon2.c                                             *
 *                                                             *
 * This file contains the main structure of the program Solon2. *
 *                                                             *
 * Version :  1.0a                                             *
 * Copywrite (c) University of Minnesota                       *
 * Department of Civil and Mineral Engineering                 *
 * Written by Nikos V. Vairamidis                              *
 *=============================================================*/


/*============* include files *=================================*/

#include "solon2.i"


/*============* main program *=================================*/

void main()
{
  int response;

  set_up();

  lb10:
  response = main_menu();
  switch(response)
  {
    case DATAMENU:
            lb20:
            response = data_menu();
            switch(response)
            {
              case LOADFILE :
                        load_file();
                        goto lb20;
              case SAVEFILE :
                        save_file();
                        goto lb20;
              case EDITFILE :
                        edit_file();
                        goto lb20;
              case NEWFILE  :
                        new_file();
                        goto lb20;
              case MAINMENU :
                        goto lb10;
```

```
                default       :
                              goto lb20;
            }
            goto lb10;
    case SIMULAT :
            simulation();
            goto lb10;
    case GRAPH   :
            graphics();
            goto lb10;
    case QUIT    :
            goto lb30;
    default      :
            goto lb10;
  }

  lb30:
  _setbkcolor(0l);
  _clearscreen(_GCLEARSCREEN);

  remove("results.tmp");
  exit(0);
}


/*============* end of the file *================================*/
```

```
/*=============================================================*
 * File : data.c                                               *
 *                                                             *
 * This file contains all the data management functions.       *
*=============================================================*/


/*============* include files *================================*/

#include "solon2.i"
char naf[16];


/*============* load data file *================================*/

void load_file()
{
  int i, j;
  char ch;
  FILE *inf;

  reset_all();

  cwindow(TEXTWINDOW);

  _settextcolor(3);
  _outtext("Load File");

  lb10:
  swindow(TTEXTWINDOW);
  _settextcolor(7);
  _outtext("Name of file : ");
  read_str(1, 16);

  if ((inf = fopen(naf, "rt")) == NULL)
  {
    _settextposition(3, 1);
    _settextcolor(4);
    _outtext("ERROR : Could not open the file");
    ch = getch();
    _settextcolor(7);
    return;
  }

  fscanf(inf, "%d %d %d %e %e", &simlen,
                    &mandel,
                    &pepdel,
                    &operhrs,
                    &busseats);

  fscanf(inf, "%e %e %e %e %e", &cstst,
                    &cstpml,
```

```
                    &cstphr,
                    &gaspr,
                    &tickt);

  fscanf(inf, "%d %e", &maindata.numzones,
                  &maindata.frequency);

  for (i = 0; i < MAXZONES; i ++)
    fscanf(inf, "%e", &maindata.length[i]);

  fscanf(inf, "%d", &linesdata.numlines);
  for (i = 0; i < MAXLINES; i ++)
  {
    fscanf(inf, "%d %d %d %e",
           &linesdata.linesexist[i],
           &linesdata.linenum[i],
           &linesdata.linezones[i],
           &linesdata.frequency[i]);
  }

  for (i = 0; i < MAXLINES; i ++)
  {
    for (j = 0; j < MAXZONES; j ++)
    {
      fscanf(inf, "%e %e %e %e %e %e %e %e %e %e %e",
             &linesdata.length[i][j],
             &linesdata.ivt_a[i][j],
             &linesdata.ivt_b[i][j],
             &linesdata.ovt_b[i][j],
             &linesdata.egress_c[i][j],
             &linesdata.egress_t[i][j],
             &linesdata.parkcost_c[i][j],
             &linesdata.autoph[i][j],
             &linesdata.persph[i][j],
             &linesdata.income[i][j],
             &linesdata.worktrips[i][j]);
    }
  }

  fclose(inf);
  draw_line();

  return;
}


/*============* save data file *================================*/

void save_file()
{
  int i, j;
  char ch;
```

```
FILE *outf;

cwindow(TEXTWINDOW);

_settextcolor(3);
_outtext("Save File");

lb10:
swindow(TTEXTWINDOW);
_settextcolor(7);
_outtext("Name of file : ");
read_str(1, 16);

if ((outf = fopen(naf, "wt")) == NULL)
{
  _settextposition(3, 1);
  _settextcolor(4);
  _outtext("ERROR : Could not open the file");
  ch = getch();
  _settextcolor(7);
  return;
}

fprintf(outf, " %3d\n %3d   %3d\n %10e   %10e\n", simlen,
                                mandel,
                                pepdel,
                                operhrs,
                                busseats);

fprintf(outf, " %10e   %10e   %10e\n %10e   %10e\n", cstst,
                                cstpml,
                                cstphr,
                                gaspr,
                                tickt);

fprintf(outf, "\n %3d\n %10e\n\n\n", maindata.numzones,
                        maindata.frequency);

for (i = 0; i < MAXZONES; i ++)
  fprintf(outf, " %10e\n", maindata.length[i]);

fprintf(outf, "\n %3d\n", linesdata.numlines);
for (i = 0; i < MAXLINES; i ++)
{
  fprintf(outf, " %3d   %3d   %3d\n %10e\n",
          linesdata.linesexist[i],
          linesdata.linenum[i],
          linesdata.linezones[i],
          linesdata.frequency[i]);
}

for (i = 0; i < MAXLINES; i ++)
```

```
  {
    for (j = 0; j < MAXZONES; j ++)
    {
      fprintf(outf, " %10e   %10e   %10e   %10e   %10e   %10e",
              linesdata.length[i][j],
              linesdata.ivt_a[i][j],
              linesdata.ivt_b[i][j],
              linesdata.ovt_b[i][j],
              linesdata.egress_c[i][j],
              linesdata.egress_t[i][j]);
      fprintf(outf, "   %10e   %10e   %10e   %10e   %10e\n",
              linesdata.parkcost_c[i][j],
              linesdata.autoph[i][j],
              linesdata.persph[i][j],
              linesdata.income[i][j],
              linesdata.worktrips[i][j]);
    }
  }

  fclose(outf);
  return;
}


/*============* edit data file *================================*/

void edit_file()
{
  int i, j;
  char buf[80];

  cwindow(TEXTWINDOW);

  _settextcolor(3);
  _outtext("Edit File");

  cwindow(TTEXTWINDOW);
  _settextcolor(7);
  _outtext("System data");

  cwindow(DATAWINDOW);
  _settextposition(2, 2);
  _outtext("Simulation length (weeks) : ");
  simlen = read_int(2, 29, 500, 10, simlen);

  _settextposition(3, 2);
  _outtext("Manager delay (weeks)     : ");
  mandel = read_int(3, 29, 100, 1, mandel);

  _settextposition(4, 2);
  _outtext("Travelers delay (weeks)   : ");
  pepdel = read_int(4, 29, 100, 1, pepdel);
```

```
_settextposition(5, 2);
_outtext("Operation hours            : ");
operhrs = read_real(5, 29, 24.0, 1.0, operhrs);

_settextposition(6, 2);
_outtext("Seats per bus              : ");
busseats = read_real(6, 29, 200.0, 10.0, busseats);

_settextposition(7, 2);
_outtext("Constant cost / day ($)    : ");
cstst = read_real(7, 29, 100000.0, 0.0, cstst);

_settextposition(8, 2);
_outtext("Cost / mile ($)            : ");
cstpml = read_real(8, 29, 100.0, 0.0, cstpml);

cwindow(DATAWINDOW);
_settextposition(2, 2);
_outtext("Cost / hour ($)            : ");
cstphr = read_real(2, 29, 500.0, 0.0, cstphr);

_settextposition(3, 2);
_outtext("Gas price ($)              : ");
gaspr = read_real(3, 29, 10.0, 0.0, gaspr);

_settextposition(4, 2);
_outtext("Fare price ($)             : ");
tickt = read_real(4, 29, 10.0, 0.0, tickt);

cwindow(TTEXTWINDOW);
_settextcolor(7);
_outtext("Main line");

cwindow(DATAWINDOW);
_settextposition(2, 2);
_outtext("Frequency (bus/h) : ");
maindata.frequency = read_real(2, 21, 60.0, 0.1,
                     maindata.frequency);

for (i = 0; i < maindata.numzones; i ++)
{
  show_zone(0, i + 1, 1);
  cwindow(TTEXTWINDOW);
  _settextcolor(7);
  sprintf(buf, "Zone %2d of main line", i + 1);
  _outtext(buf);

  cwindow(DATAWINDOW);
  _settextposition(2, 2);
  _outtext("Length (mls)       : ");
  maindata.length[i] = read_real(2, 21, 20.0, 0.1,
                       maindata.length[i]);
```

```
  show_zone(0, i + 1, 0);
}

for (i = 0; i < linesdata.numlines; i ++)
{
  if (linesdata.linesexist[i] == 1)
  {
    cwindow(TTEXTWINDOW);
    _settextcolor(7);
    sprintf(buf, "Line %2d", linesdata.linenum[i] + 1);
    _outtext(buf);

    cwindow(DATAWINDOW);
    _settextposition(2, 2);
    _outtext("Frequency (bus/h) : ");
    linesdata.frequency[i] =
          read_real(2, 21, 60.0, 0.1, linesdata.frequency[i]);

    for (j = 0; j < linesdata.linezones[i]; j ++)
    {
   show_zone(i + 1, j + 1, 1);
   cwindow(TTEXTWINDOW);
   _settextcolor(7);
   sprintf(buf, "Zone %d of line %2d",
              j + 1, linesdata.linenum[i] + 1);
   _outtext(buf);

   cwindow(DATAWINDOW);
   _settextposition(2, 2);
   _outtext("Length (mls)          : ");
   linesdata.length[i][j] = read_real(2, 24, 20.0, 0.1,
                            linesdata.length[i][j]);


   _settextposition(3, 2);
   _outtext("IVT(a) (min)          : ");
   linesdata.ivt_a[i][j] =
            read_real(3, 24, 120.0, 0.25, linesdata.ivt_a[i][j]);

   _settextposition(4, 2);
   _outtext("IVT(b) (min)          : ");
   linesdata.ivt_b[i][j] =
            read_real(4, 24, 120.0, 0.25, linesdata.ivt_b[i][j]);

   _settextposition(5, 2);
   _outtext("OVT(b) (min)          : ");
   linesdata.ovt_b[i][j] =
            read_real(5, 24, 60.0, 0.25, linesdata.ovt_b[i][j]);

   _settextposition(6, 2);
   _outtext("Egress Time(c) (min) : ");
   linesdata.egress_c[i][j] =
            read_real(6, 24, 30.0, 0.0,
```

```
                        linesdata.egress_c[i][j]);

      _settextposition(7, 2);
      _outtext("Egress Time(t) (min) : ");
      linesdata.egress_t[i][j] =
               read_real(7, 24, 30.0, 0.0,
                        linesdata.egress_t[i][j]);

      _settextposition(8, 2);
      _outtext("Parking cost(c) ($)  : ");
      linesdata.parkcost_c[i][j] =
               read_real(8, 24, 20.0, 0.0,
                        linesdata.parkcost_c[i][j]);

      cwindow(DATAWINDOW);
      _settextposition(2, 2);
      _outtext("Autos/household      : ");
      linesdata.autoph[i][j] =
          read_real(2, 24, 5.0, 0.0, linesdata.autoph[i][j]);

      _settextposition(3, 2);
      _outtext("Persons/household    : ");
      linesdata.persph[i][j] =
          read_real(3, 24, 10.0, 1.0, linesdata.persph[i][j]);

      _settextposition(4, 2);
      _outtext("Income ($)           : ");
      linesdata.income[i][j] =
          read_real(4, 24, 300000.0, 0.0, linesdata.income[i][j]);

      _settextposition(5, 2);
      _outtext("Work trips           : ");
      linesdata.worktrips[i][j] =
        read_real(5, 24, 30000.0, 0.0, linesdata.worktrips[i][j]);

      show_zone(i + 1, j + 1, 0);
       }
     }
   }

   set_up();
   draw_line();

   return;
}

/*============* new data file *==================================*/

void new_file()
{
   int i, j, response;
```

```
char buf[80];

reset_all();

cwindow(TEXTWINDOW);

_settextcolor(3);
_outtext("New File");

swindow(TTEXTWINDOW);
_settextcolor(7);
_outtext("Number of zones on main line : ");
maindata.numzones = read_int(1, 32, 6, 1, 6);

draw_line();

linesdata.numlines = maindata.numzones * 2;

j = -1;
for (i = 0; i < linesdata.numlines; i ++)
{
  show_line(i + 1, 1);
  swindow(TTEXTWINDOW);
  _settextcolor(7);
  _outtext("Select this line (1: No, 2: Yes) : ");
  response = read_int(1, 36, 2, 1, 2);

  if (response == 2)
  {
    linesdata.linesexist[i] = 1;
    linesdata.linenum[i] = ++j;
  }
  else
    show_line(i + 1, 0);
}

draw_line();

for (i = 0; i < linesdata.numlines; i ++)
{
  if (linesdata.linesexist[i] == 1)
  {
    show_line(i + 1, 1);
    swindow(TTEXTWINDOW);
    _settextcolor(7);
    sprintf(buf, "Number of zones on line %2d : ",
                  linesdata.linenum[i] + 1);
    _outtext(buf);
    linesdata.linezones[i] = read_int(1, 30, 6, 1, 6);
    show_line(i + 1, 0);
  }
}
```

```
draw_line();

cwindow(TTEXTWINDOW);
_settextcolor(7);
_outtext("System data");

cwindow(DATAWINDOW);
_settextposition(2, 2);
_outtext("Simulation length (weeks) : ");
simlen = read_int(2, 29, 500, 10, 100);

_settextposition(3, 2);
_outtext("Manager delay (weeks)     : ");
mandel = read_int(3, 29, 100, 1, 1);

_settextposition(4, 2);
_outtext("Travelers delay (weeks)   : ");
pepdel = read_int(4, 29, 100, 1, 10);

_settextposition(5, 2);
_outtext("Operation hours / week    : ");
operhrs = read_real(5, 29, 24.0, 1.0, 20.0);

_settextposition(6, 2);
_outtext("Seats per bus             : ");
busseats = read_real(6, 29, 200.0, 10.0, 60.0);

_settextposition(7, 2);
_outtext("Constant cost / day ($)   : ");
cstst = read_real(7, 29, 500.0, 0.0, 1000.0);

_settextposition(8, 2);
_outtext("Cost / mile ($)           : ");
cstpml = read_real(8, 29, 500.0, 0.0, 0.5);

cwindow(DATAWINDOW);
_settextposition(2, 2);
_outtext("Cost / hour ($)           : ");
cstphr = read_real(2, 29, 500.0, 0.0, 2.0);

_settextposition(3, 2);
_outtext("Gas price ($)             : ");
gaspr = read_real(3, 29, 10.0, 0.0, 1.3);

_settextposition(4, 2);
_outtext("Fare price ($)            : ");
tickt = read_real(4, 29, 10.0, 0.0, 1.0);

cwindow(TTEXTWINDOW);
_settextcolor(7);
_outtext("Main line");
```

```
cwindow(DATAWINDOW);
_settextposition(2, 2);
_outtext("Frequency (bus/h) : ");
maindata.frequency = read_real(2, 21, 60.0, 0.1, 6.0);

for (i = 0; i < maindata.numzones; i ++)
{
  show_zone(0, i + 1, 1);
  cwindow(TTEXTWINDOW);
  _settextcolor(7);
  sprintf(buf, "Zone %2d of main line", i + 1);
  _outtext(buf);

  cwindow(DATAWINDOW);
  _settextposition(2, 2);
  _outtext("Length (mls)      : ");
  maindata.length[i] = read_real(2, 21, 20.0, 0.1, 5.0);
  show_zone(0, i + 1, 0);
}

for (i = 0; i < linesdata.numlines; i ++)
{
  if (linesdata.linesexist[i] == 1)
  {
    cwindow(TTEXTWINDOW);
    _settextcolor(7);
    sprintf(buf, "Line %2d", linesdata.linenum[i] + 1);
    _outtext(buf);

    cwindow(DATAWINDOW);
    _settextposition(2, 2);
    _outtext("Frequency (bus/h) : ");
    linesdata.frequency[i] = read_real(2, 21, 60.0, 0.1, 6.0);

    for (j = 0; j < linesdata.linezones[i]; j ++)
    {
   show_zone(i + 1, j + 1, 1);
   cwindow(TTEXTWINDOW);
   _settextcolor(7);
   sprintf(buf, "Zone %d of line %2d",
              j + 1, linesdata.linenum[i] + 1);
   _outtext(buf);

   cwindow(DATAWINDOW);
   _settextposition(2, 2);
   _outtext("Length (mls)          : ");
   linesdata.length[i][j] = read_real(2, 24, 20.0, 0.1, 3.0);

   _settextposition(3, 2);
   _outtext("IVT(a) (min)          : ");
   linesdata.ivt_a[i][j] = read_real(3, 24, 120.0, 0.25, 9.0);
```

```
      _settextposition(4, 2);
      _outtext("IVT(b) (min)          : ");
      linesdata.ivt_b[i][j] = read_real(4, 24, 120.0, 0.25, 16.0);

      _settextposition(5, 2);
      _outtext("OVT(b) (min)          : ");
      linesdata.ovt_b[i][j] = read_real(5, 24, 60.0, 0.25, 4.0);

      _settextposition(6, 2);
      _outtext("Egress Time(c) (min) : ");
      linesdata.egress_c[i][j] = read_real(6, 24, 30.0, 0.0, 8.0);

      _settextposition(7, 2);
      _outtext("Egress Time(t) (min) : ");
      linesdata.egress_t[i][j] = read_real(7, 24, 30.0, 0.0, 5.0);

      _settextposition(8, 2);
      _outtext("Parking cost(c) ($)  : ");
      linesdata.parkcost_c[i][j] = read_real(8, 24, 20.0, 0.0,
                                   3.73);

      cwindow(DATAWINDOW);
      _settextposition(2, 2);
      _outtext("Autos/household       : ");
      linesdata.autoph[i][j] = read_real(2, 24, 5.0, 0.0, 1.27);

      _settextposition(3, 2);
      _outtext("Persons/household     : ");
      linesdata.persph[i][j] = read_real(3, 24, 10.0, 1.0, 2.39);

      _settextposition(4, 2);
      _outtext("Income ($)            : ");
      linesdata.income[i][j] = read_real(4, 24, 300000.0, 0.0,
                                         42000.0);

      _settextposition(5, 2);
      _outtext("Work trips            : ");
      linesdata.worktrips[i][j] =
             read_real(5, 24, 30000.0, 0.0, 5000.00);

      show_zone(i + 1, j + 1, 0);
       }
     }
   }

   set_up();
   draw_line();

   return;
}
```

```
/*============* read an integer *================================*/

int read_int(li, co, up, dwn, va)
int li, co, up, dwn, va;
/* li ... line
   co ... column
   up ... max of the input number
   dwn .. min of the input number
   va ... default value */
{
  int in;
  char response, buf[80], stin[80], *stp;

  in = va;

  _settextposition(li, co);
  sprintf(buf, "%-16d", in);
  _outtext(buf);
  response = getch();
  if (response == ENTER)
    return in;

  lb10:
  _settextposition(li, co);
  _outtext("                                ");
  sprintf(stin, "");
  _settextposition(li, co);

  lb20:
  response = getche();
  switch(response)
  {
    case ZERO :
            sprintf(stin, "%s0", stin);
            goto lb20;
    case ONE  :
            sprintf(stin, "%s1", stin);
            goto lb20;
    case TWO  :
            sprintf(stin, "%s2", stin);
            goto lb20;
    case THREE:
            sprintf(stin, "%s3", stin);
            goto lb20;
    case FOUR :
            sprintf(stin, "%s4", stin);
            goto lb20;
    case FIVE :
            sprintf(stin, "%s5", stin);
            goto lb20;
    case SIX  :
            sprintf(stin, "%s6", stin);
```

```
            goto lb20;
    case SEVEN:
            sprintf(stin, "%s7", stin);
            goto lb20;
    case EIGHT:
            sprintf(stin, "%s8", stin);
            goto lb20;
    case NINE :
            sprintf(stin, "%s9", stin);
            goto lb20;
    case ENTER:
            goto lb30;
    case ESC  :
            goto lb10;
    default   :
            goto lb10;
  }

  lb30:
  in = strtol(stin, &stp, 10);
  if ((in > up) || (in < dwn))
    goto lb10;

  return in;
}


/*============* read a real *====================================*/

float read_real(li, co, up, dwn, va)
int li, co;
double up, dwn, va;
/* li ... line
   co ... column
   up ... max of the input number
   dwn .. min of the input number
   va ... default value */
{
  float in;
  char response, stin[80], buf[80], *stp;

  in = va;

  _settextposition(li, co);
  sprintf(buf, "%-16.2f", in);
  _outtext(buf);
  response = getch();
  if (response == ENTER)
    return in;

  lb10:
  _settextposition(li, co);
```

```
_outtext("                                        ");
sprintf(stin, "");
_settextposition(li, co);

lb20:
response = getche();
switch(response)
{
  case ZERO  :
        sprintf(stin, "%s0", stin);
        goto lb20;
  case ONE   :
        sprintf(stin, "%s1", stin);
        goto lb20;
  case TWO   :
        sprintf(stin, "%s2", stin);
        goto lb20;
  case THREE :
        sprintf(stin, "%s3", stin);
        goto lb20;
  case FOUR  :
        sprintf(stin, "%s4", stin);
        goto lb20;
  case FIVE  :
        sprintf(stin, "%s5", stin);
        goto lb20;
  case SIX   :
        sprintf(stin, "%s6", stin);
        goto lb20;
  case SEVEN :
        sprintf(stin, "%s7", stin);
        goto lb20;
  case EIGHT :
        sprintf(stin, "%s8", stin);
        goto lb20;
  case NINE  :
        sprintf(stin, "%s9", stin);
        goto lb20;
  case PERIOD:
        sprintf(stin, "%s.", stin);
        goto lb20;
  case ENTER :
        goto lb30;
  case ESC   :
        goto lb10;
  default    :
        goto lb10;
}

lb30:
in = strtod(stin, &stp);
if ((in > up) || (in < dwn))
```

```
    goto lb10;

  return in;
}

/*============* read a string *==================================*/

void read_str(li, co)
int li, co;
/* li ... line
   co ... column */
{
  char buf[80],
       response;

  sprintf(naf, "noname.dat");

  _settextposition(li, co);
  sprintf(buf, "%-16s", naf);
  _outtext(buf);
  response = getch();
  if (response == ENTER)
    return;

  lb10:
  _settextposition(li, co);
  _outtext("                                    ");
  sprintf(naf, "");
  _settextposition(li, co);

  lb20:
  response = getche();
  switch(response)
  {
    case ENTER:
            break;
    case ESC  :
            goto lb10;
    default   :
            sprintf(naf, "%s%c", naf, response);
            goto lb20;
  }

  return;
}

/*============* reset all the variables *=======================*/

void reset_all()
{
```

```
  int i, j;

  simlen = mandel = pepdel = 0;
  operhrs = busseats = 0.0;
  cstst = cstpml = cstphr = gaspr = tickt = 0.0;

  maindata.numzones = 0;
  maindata.frequency = 0.0;
  for (i = 0; i < MAXZONES; i ++)
    maindata.length[i] = 0.0;

  linesdata.numlines = 0;
  for (i = 0; i < MAXLINES; i ++)
  {
    linesdata.linesexist[i] = linesdata.linenum[i]
                    = linesdata.linezones[i] = 0;
    linesdata.frequency[i] = 0.0;
  }

  for (i = 0; i < MAXLINES; i ++)
  {
    for (j = 0; j < MAXZONES; j ++)
    {
      linesdata.length[i][j] = 0.0;
      linesdata.ivt_a[i][j] = 0.0;
      linesdata.ivt_b[i][j] = 0.0;
      linesdata.ovt_b[i][j] = 0.0;
      linesdata.egress_c[i][j] = 0.0;
      linesdata.egress_t[i][j] = 0.0;
      linesdata.parkcost_c[i][j] =0.0;
      linesdata.autoph[i][j] =0.0;
      linesdata.persph[i][j] = 0.0;
      linesdata.income[i][j] = 0.0;
      linesdata.worktrips[i][j] = 0.0;
    }
  }

  return;
}

/*============* end of the file *================================*/
```

```
/*================================================================*
 * File : simulate.c                                              *
 *                                                                *
 * This file contains the simulation functions.                   *
 *================================================================*/


/*============* include files *=================================*/

#include "solon2.i"


/*============* variable declaration *=========================*/

float mainrid, prmainrid, mainfreq, mainlgth;
linereal linrid, prlinrid, lintr, linfreq, linlgth;


/*============* run the simulation *============================*/

void simulation()
{
  int i, j, w;
  float ua, ub, emu, ut, uc,
     pat, pbt, pt,
     pa, pb, pc,
     lf;
  char ch;
  FILE *tmpf;

  mainfreq = maindata.frequency;
  for (i = 0; i < MAXLINES; i ++)
    linfreq[i] = linesdata.frequency[i];

  cwindow(TEXTWINDOW);

  _settextcolor(3);
  _outtext("Running Simulation");

  swindow(TTEXTWINDOW);
  _settextcolor(7);
  _outtext("Wait ... ");

  if ((tmpf = fopen("results.tmp", "wt")) == NULL)
  {
     _settextposition(3, 1);
    _settextcolor(4);
    _outtext("ERROR : Could not open temporary file");
    ch = getch();
     _settextcolor(7);
    return;
   }
```

```
  mainlgth = 0.0;
  for (i = 0; i < MAXZONES; i ++)
    mainlgth += maindata.length[i];

  for (i = 0; i < MAXZONES; i ++)
  {
    linlgth[i] = 0.0;
    for (j = 0; j < MAXLINES; j ++)
      linlgth[i] += linesdata.length[i][j];
  }

  _settextposition(3, 10);
  _outtext("                                                              ");

  for (w = 0; w < simlen; w ++)
  {
    _settextposition(3, 10 + w * 50 / simlen);
    _outtext("▓");

    mainrid = 0.0;

    for (i = 0; i < MAXLINES; i ++)
    {
      linrid[i] = lintr[i] = 0.0;

      if (linesdata.linesexist[i] == 1)
      {
     for (j = 0; j < linesdata.linezones[i]; j ++)
      {
        ua = - 3.23 - 0.073 * linesdata.ivt_a[i][j]
             +    1.74    *    (linesdata.autoph[i][j]    /
linesdata.persph[i][j]);
        ub = - 0.073 * linesdata.ivt_b[i][j]
             - 0.105 * (linesdata.ovt_b[i][j] + wait(i, j));

        emu = log(exp(ua) + exp(ub));

        ut = 0.485 * emu - 0.0599 * linesdata.egress_t[i][j]
             - 0.0144 * (gaspr / linesdata.income[i][j] * 1000);
        uc = - 1.626 - 0.0599 * linesdata.egress_c[i][j]
             - 0.180 * (linesdata.parkcost_c[i][j] /
               linesdata.income[i][j]
               * 1000.0) - 0.0144 * (gaspr / linesdata.income[i][j]
               * 1000);

        pat = exp(ua) / (exp(ua) + exp(ub));
        pbt = 1.0 - pat;
        pt = exp(ut) / (exp(ut) + exp(uc));

        pa = pat * pt;
        pb = pbt * pt;
        pc = 1.0 - pt;
```

```
    lintr[i] += pa * linesdata.worktrips[i][j];
    linrid[i] += pb * linesdata.worktrips[i][j];
  }
   }

   mainrid += lintr[i] + linrid[i];
}

if (w != 0)
  riders_calc();
else
{
  prmainrid = mainrid;
  for (i = 0; i < MAXLINES; i ++)
  {
 if (linesdata.linesexist[i] == 1)
   prlinrid[i] = linrid[i];
  }
}

lf = prmainrid / (operhrs * mainfreq) / busseats;

fprintf(tmpf,
"\n %3d\n %10e   %10e   %10e   %10e   %10e   %10e   %10e
    %10e\n",
w,
60.0 / mainfreq,
prmainrid,
(lf < 1.0) ? lf : 1.0,
mainfreq,
mainfreq * operhrs * (cstst + cstphr + mainlgth * cstphr),
prmainrid * tickt,
prmainrid * tickt
- mainfreq * operhrs * (cstst + cstphr + mainlgth * cstphr),
prmainrid * tickt
/ (mainfreq * operhrs * (cstst + cstphr + mainlgth * cstphr)));

for (i = 0; i < MAXLINES; i ++)
{
  if (linesdata.linesexist[i] == 1)
  {
 lf = prlinrid[i] / (operhrs * linfreq[i]) / busseats;

 fprintf(tmpf,
 " %10e   %10e   %10e   %10e   %10e   %10e   %10e   %10e\n",
 60.0 / linfreq[i],
 prlinrid[i],
 (lf < 1.0) ? lf : 1.0,
 linfreq[i],
 linfreq[i] * operhrs * (cstst + cstphr + linlgth[i] * cstphr),
 prlinrid[i] * tickt,
 prlinrid[i] * tickt
```

```
      - linfreq[i] * operhrs * (cstst + cstphr + linlgth[i] *
      cstphr),
      prlinrid[i] * tickt
      / (linfreq[i] * operhrs * (cstst + cstphr + linlgth[i] *
        cstphr)));
       }
     }

     frequences_calc();
   }

   fclose(tmpf);
   return;
}

/*============* calculate the waiting time *====================*/

float wait(i, j)
int i, j;
{
   float ti;

   ti = 60.0 / linfreq[i] / 6.0
      + 60.0 / mainfreq / 3.0;

   return ti;
}

/*============* calculate the ridership *=======================*/

void riders_calc()
{
   int i;

   prmainrid += (mainrid - prmainrid) / pepdel;

   for (i = 0; i < MAXLINES; i ++)
   {
     if (linesdata.linesexist[i] == 1)
       prlinrid[i] += (linrid[i] - prlinrid[i]) / pepdel;
   }
}

/*============* calculate the frequences *======================*/

void frequences_calc()
{
   int i;
   float u, lf;
```

```
  lf = prmainrid / operhrs / mainfreq / busseats;

  if (lf < 1.0)
    u = 2.0 * lf - 1.0;
  else
    u = 1.0;

  mainfreq += u / mandel / 4.0 * mainfreq;
  mainfreq = (mainfreq >= 0.1
          ? mainfreq
          : 0.1);

  for (i = 0; i < MAXLINES; i ++)
  {
    if (linesdata.linesexist[i] == 1)
    {
      lf = prlinrid[i] / operhrs / linfreq[i] / busseats;

      if (lf < 1.0)
     u = 4.0 * lf - 1.0;
      else
     u = 1.0;

      linfreq[i] += u / mandel * linfreq[i];
      linfreq[i] = (linfreq[i] >= 0.1
             ? linfreq[i]
             : 0.1);
    }
  }
}


/*============* end of the file *================================*/
```

```
/*=================================================================*
 * File : graphics.c                                               *
 *                                                                 *
 * This file contains the graphic functions.                       *
 *=================================================================*/

/*============* include files *=================================*/

#include "solon2.i"

/*============* variable declaration *=========================*/

int linu[3];
char tit[8][14] = {"Headway",
                   "Ridership",
                   "Load Factor",
                   "Frequency",
                   "Cost",
                   "Revenue",
                   "Profit",
                   "Revenue/Cost"};

/*============* main graphics function *=======================*/

void graphics()
{
  int i,
      nuli, inca;
  char ch;

  cwindow(TEXTWINDOW);
  _settextcolor(3);
  _outtext("Graphics Menu");

  swindow(TTEXTWINDOW);
  _settextcolor(7);
  _outtext("Number of lines : ");
  nuli = read_int(1, 19, 3, 1, 1);
  _settextposition(1, 25);
  _outtext("Select line number (0 for main line) : ");
  for (i = 1; i <= nuli; i ++)
    linu[i - 1] = read_int(1, 64 + 3 * (i - 1), linesdata.numlines,
                           0, 0);
  _outtext("\n1. Headway     3. Load factor   5. Cost
           7. Profit\n");
  _outtext("2. Ridership   4. Frequency     6. Revenue
           8. Revenue/Cost\n");
  _outtext("Select indicator : ");
  inca = read_int(4, 20, 8, 1, 1);
```

```
  do_graphics(nuli, inca);

  return;
}


/*============* do the graphics *================================*/

void do_graphics(nuli, inca)
int nuli, inca;
{
  int i, j, k, g, w;
  float data, val[8],
     maxx, minx,
     maxy, miny,
     sx, sy,
     xx, yy;
  char ch, buf[80];
  FILE *tmpf;

  cwindow(TTEXTWINDOW);

  _outtext("Reading the data ...");

  if ((tmpf = fopen("results.tmp", "rt")) == NULL)
  {
    _settextposition(3, 1);
    _settextcolor(4);
    _outtext("ERROR : Could not open temporary file");
    ch = getch();
    _settextcolor(7);
    return;
  }

  miny = 1.0e30;
  maxy = -1.0e30;

  for (k = 1; k <= nuli; k ++)
  {
    tmpf = fopen("results.tmp", "rt");
    for (w = 0; w < simlen; w ++)
    {
      fscanf(tmpf, "%d", &g);
      for (i = 0; i <= linesdata.numlines; i ++)
      {
     if ((i == 0) || ((i != 0) && (linesdata.linesexist[i - 1]
          != 0)))
      {
        for (j = 0; j < 8; j ++)
          fscanf(tmpf, "%e", &val[j]);
        if (((i == linu[k - 1]) && (i == 0)) ||
           (linesdata.linenum[i - 1] + 1 == linu[k - 1]))
```

```
        data = val[inca - 1];
    }
     }
     if(data > maxy)
    maxy = data;
     if(data < miny)
    miny = data;
   }

   fclose(tmpf);
}


if (miny == maxy)
{
   miny -= 1.0;
   maxy += 1.0;
}

sy = -270.0 / (maxy - miny);

minx = 0;
maxx = simlen;
sx = 550.0 / maxx;

_setvideomode(_ERESCOLOR);

_setlogorg(70, 300);

_moveto(0, 0);
_lineto(0, -270);
_lineto(550, -270);
_lineto(550, 0);
_lineto(0 ,0);

for (k = 0; k < nuli; k ++)
{
   _settextcolor(k + 2);
   _settextposition(3, 25 + 12 * k);
   if (linu[k] == 0)
      _outtext("Main line");
   else
   {
      sprintf(buf, "Line %2d", linu[k]);
      _outtext(buf);
   }
}

_settextcolor(7);
_settextposition(42, 33);
_outtext("Week of Simulation");
```

```
_settextposition(3, 5);
_outtext(tit[inca - 1]);

for (i = 0; i <= 10; i ++)
{
  _moveto(i * 55, 0);
  _lineto(i * 55, 5);
}

_settextposition(40, 10);
_outtext("0");

_settextposition(40, 42);
sprintf(buf, "%3d", simlen / 2);
_outtext(buf);

_settextposition(40, 77);
sprintf(buf, "%3d", simlen);
_outtext(buf);

for (i = 0; i <= 10; i ++)
{
  _moveto(0, i * - 27);
  _lineto(-5, i * - 27);
}
_settextposition(38, 1);
sprintf(buf, "%8.1e", miny);
_outtext(buf);

_settextposition(21, 1);
sprintf(buf, "%8.1e", (maxy + miny) / 2.0);
_outtext(buf);

_settextposition(5, 1);
sprintf(buf, "%8.1e", maxy);
_outtext(buf);

for (k = 1; k <= nuli; k ++)
{
  _moveto(0, 0);
  _setcolor(k + 1);
  tmpf = fopen("results.tmp", "rt");
  for (w = 0; w < simlen; w ++)
  {
    fscanf(tmpf, "%d", &g);
    for (i = 0; i <= linesdata.numlines; i ++)
    {
   if ((i == 0) || ((i != 0) && (linesdata.linesexist[i - 1]
        != 0)))
   {
     for (j = 0; j < 8; j ++)
       fscanf(tmpf, "%e", &val[j]);
```

```
        if (((i == linu[k - 1]) && (i == 0)) ||
           (linesdata.linenum[i - 1] + 1 == linu[k - 1]))
          data = val[inca - 1];
      }
       }
       xx = (w - minx) * sx;
       yy = (data - miny) * sy;
       _lineto(xx, yy);
     }

     fclose(tmpf);
   }

   ch = getch(); ch = getch();

   _setvideomode(_DEFAULTMODE);

   set_up();
   draw_line();

   return;
}

/*============* end of the file *================================*/
```

```
/*=============================================================*
 * File : menus.c                                              *
 *                                                             *
 * This file contains all the menus functions.                 *
*=============================================================*/


/*============* include files *=================================*/

#include "solon2.i"


/*============* set up function *===============================*/

void set_up()
{
  int i;

  _clearscreen(_GCLEARSCREEN);

  _setbkcolor(11);
  _settextwindow(1, 1, 35, 80);
  _clearscreen(_GWINDOW);

  _setbkcolor(01);
  _settextwindow(36, 1, 43, 80);
  _clearscreen(_GWINDOW);

  _settextcolor(7);
  _settextposition(1,1);
  for (i = 1; i <= 80; i ++)
    _outtext("");

  return;
}


/*============* select windows graph or text *==================*/

void swindow(window)
int window;
{
  switch(window)
  {
    case GRAPHWINDOW:
                _setbkcolor(11);
                _settextwindow(1, 5, 35, 75);
                break;
    case TEXTWINDOW :
                _setbkcolor(01);
                _settextwindow(38, 5, 43, 75);
                break;
```

```
    case TTEXTWINDOW:
                _setbkcolor(01);
                _settextwindow(40, 5, 43, 75);
                break;
    case DATAWINDOW :
                _setbkcolor(11);
                _settextwindow(34, 41, 42, 78);
  }

  return;
}


void cwindow(window)
int window;
{
  switch(window)
  {
    case GRAPHWINDOW:
                _setbkcolor(11);
                _settextwindow(1, 5, 35, 75);
                _clearscreen(_GWINDOW);
                break;
    case TEXTWINDOW :
                _setbkcolor(01);
                _settextwindow(38, 5, 43, 75);
                _clearscreen(_GWINDOW);
                break;
    case TTEXTWINDOW:
                _setbkcolor(01);
                _settextwindow(40, 5, 43, 75);
                _clearscreen(_GWINDOW);
                break;
    case DATAWINDOW :
                _setbkcolor(71);
                _settextwindow(33, 40, 43, 79);
                _clearscreen(_GWINDOW);

                _setbkcolor(11);
                _settextwindow(34, 41, 42, 78);
                _clearscreen(_GWINDOW);
  }

  return;
}


/*============* main menu function *============================*/

int main_menu()
{
  int response;
```

```
  cwindow(TEXTWINDOW);

  _settextcolor(3);
  _outtext("Main Menu\n\n");

  _settextcolor(7);
  _outtext("1.   Data Management          4.   Quit\n");
  _outtext("2.   Run Simulation\n");
  _outtext("3.   Graphics");

  lb10:
  response = getch();

  switch(response)
  {
    case  ONE  :
          return DATAMENU;
    case  TWO  :
          return SIMULAT;
    case  THREE:
          return GRAPH;
    case  FOUR :
          return QUIT;
    default    :
          goto lb10;
  }
}


/*============* data menu function *============================*/

int data_menu()
{
  int response;

  cwindow(TEXTWINDOW);

  _settextcolor(3);
  _outtext("Data Management\n\n");

  _settextcolor(7);
  _outtext("1.   Load File                4.   New File\n");
  _outtext("2.   Save File                5.   Main Menu\n");
  _outtext("3.   Edit File");

  lb10:
  response = getch();

  switch(response)
  {
    case  ONE  :
          return LOADFILE;
```

```
    case  TWO   :
          return SAVEFILE;
    case  THREE:
          return EDITFILE;
    case  FOUR  :
          return NEWFILE;
    case  FIVE  :
          return MAINMENU;
    default     :
          goto lb10;
  }
}

/*============* end of the file *================================*/
```

```
/*================================================================*
 * File : draw.c                                                  *
 *                                                                *
 * This file contains all the draw functions.                     *
*================================================================*/


/*============* include files *=================================*/

#include "solon2.i"


/*============* variable declaration *==========================*/

step[6] = {61, 29, 19, 15, 11, 9};
step2[6] = {31, 15, 10, 8, 6, 5};


/*============* draw transit lines *============================*/

void draw_line()
{
  int i, j, li;
  struct rccoord pos;
  char buf[80];

  if (maindata.numzones == 0)
    return;

  cwindow(GRAPHWINDOW);

  _settextcolor(7);
  _setbkcolor(11);
  _settextposition(17, 2);
  _outtext("ML");
  _settextposition(18, 3);
  _outtext("");
  for (i = 0; i < maindata.numzones; i ++)
  {
    for (j = 0; j < step[maindata.numzones - 1]; j ++)
      _outtext("=");
    _outtext("+");
  }
  pos = _gettextposition();
  _settextposition(pos.row, pos.col - 1);
  _outtext("");
  _settextposition(pos.row - 1, pos.col - 1);
  _outtext("ML");

  if (linesdata.numlines == 0)
    return;
```

```
for (i = 0; i < maindata.numzones; i ++)
{
  j = 0;
  if (linesdata.linesexist[i] == 1)
    j = 10;
  if (linesdata.linesexist[maindata.numzones + i] == 1)
    j += 1;
  _settextposition(18, 3 + i * (step[maindata.numzones - 1] + 1)
                + step2[maindata.numzones - 1]);
  switch(j)
  {
    case 10:
         _outtext("╨");
         break;
    case  1:
         _outtext("╥");
         break;
    case 11:
         _outtext("╫");
  }
}

for (i = 0; i < linesdata.numlines; i ++)
{
  if ((linesdata.linesexist[i] == 1) && (linesdata.linezones[i]
       != 0))
  {
    if (i < maindata.numzones)
    {
   for (j = 0; j < linesdata.linezones[i]; j ++)
   {
     _settextposition(17 - 2 * j,
                3 + i * (step[maindata.numzones - 1] + 1)
                  + step2[maindata.numzones - 1]);
     _outtext("║");
     _settextposition(17 - 2 * j - 1,
                3 + i * (step[maindata.numzones - 1] + 1)
                  + step2[maindata.numzones - 1]);
     _outtext("╫");
   }
   pos = _gettextposition();
   _settextposition(pos.row, pos.col - 1);
   _outtext("■");
   _settextposition(pos.row - 1, pos.col - 1);
   sprintf(buf, "L%d", linesdata.linenum[i] + 1);
   _outtext(buf);
    }
    else
    {
   li = i - maindata.numzones;
   for (j = 0; j < linesdata.linezones[i]; j ++)
   {
```

```
        _settextposition(19 + 2 * j,
                    3 + li * (step[maindata.numzones - 1] + 1)
                      + step2[maindata.numzones - 1]);
        _outtext("‖");
        _settextposition(19 + 2 * j + 1,
                    3 + li * (step[maindata.numzones - 1] + 1)
                      + step2[maindata.numzones - 1]);
        _outtext("╫");
      }
      pos = _gettextposition();
      _settextposition(pos.row, pos.col - 1);
      _outtext("■");
      _settextposition(pos.row + 1, pos.col - 1);
      sprintf(buf, "L%d", linesdata.linenum[i] + 1);
      _outtext(buf);
       }
     }
   }

   return;
}


/*======* show zones on main line and on the other lines *=======*/

void show_zone(li, zo, on)
int li, zo, on;
/* li ... # of line (if li==0 then is the main line)
   zo ... # of zone on the line
   on ... on==1 to show a zone or on==0 not to show a zone */
{
   int i;

   if (li == 0)
   {
     if (maindata.numzones == 0)
       return;

     swindow(GRAPHWINDOW);

     _settextcolor(7);
     if (on == 1)
       _setbkcolor(41);
     else
       _setbkcolor(11);

     _settextposition(18, 4 + (zo - 1) *
                      (step[maindata.numzones - 1] + 1));
     for (i = 0; i < step[maindata.numzones - 1]; i ++)
       _outtext("=");
     return;
   }
```

```
  if ((linesdata.linesexist[li - 1] == 0) ||
      (linesdata.linezones[li - 1] == 0))
    return;

  swindow(GRAPHWINDOW);

  _settextcolor(7);
  if (on == 1)
    _setbkcolor(4l);
  else
    _setbkcolor(1l);

  if (li <= maindata.numzones)
    _settextposition(17 - 2 * (zo - 1),
                3 + (li - 1) * (step[maindata.numzones - 1] + 1)
                  + step2[maindata.numzones - 1]);
  else
  {
    li -= maindata.numzones;
    _settextposition(19 + 2 * (zo - 1),
                3 + (li - 1) * (step[maindata.numzones - 1] + 1)
                  + step2[maindata.numzones - 1]);
  }
  _outtext("|");

  return;
}


/*============* show lines *=====================================*/

void show_line(li, on)
int li, on;
/* li ... # of line
   on ... on==1 to show a line or on==0 not to show a line */
{
  int i;

  if (maindata.numzones == 0)
    return;

  swindow(GRAPHWINDOW);
  _settextcolor(7);
  _setbkcolor(1l);

  if (li <= maindata.numzones)
  {
    _settextposition(17, 3 + (li - 1) *
                     (step[maindata.numzones - 1] + 1)
                   + step2[maindata.numzones - 1]);
    if (on == 1)
      _outtext("^^");
```

```
    else
      _outtext(" ");
  }
  else
  {
    li -= maindata.numzones;
    _settextposition(19, 3 + (li - 1) *
                     (step[maindata.numzones - 1] + 1)
                    + step2[maindata.numzones - 1]);
    if (on == 1)
      _outtext("^_");
    else
      _outtext(" ");
  }

  return;
}

/*============* end of the file *================================*/
```

```
/*============================================================*
 * File : solon2.i                                            *
 *                                                            *
 * This file contains all the constants, variables used in the *
 * solon2 program and the function declarations.              *
*============================================================*/


/*============* include files *=================================*/

#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <conio.h>
#include <float.h>
#include <graph.h>
#include <math.h>


/*============* constants *=====================================*/

#define MAXZONES 6              /* maximum number of zones for
                                   every line */
#define MAXLINES 12             /* maximum number of bus lines */

#define GRAPHWINDOW 1           /* graph window */
#define TTEXTWINDOW 2           /* text window title */
#define TEXTWINDOW 3            /* text window */
#define DATAWINDOW 4            /* data window */

#define ZERO   48               /* number 0 */
#define ONE    49               /* number 1 */
#define TWO    50               /* number 2 */
#define THREE  51               /* number 3 */
#define FOUR   52               /* number 4 */
#define FIVE   53               /* number 5 */
#define SIX    54               /* number 6 */
#define SEVEN  55               /* number 7 */
#define EIGHT  56               /* number 8 */
#define NINE   57               /* number 9 */
#define PERIOD 46               /* period */

#define ESC 27                  /* esc key */
#define ENTER  13               /* enter key*/

#define MAINMENU 1              /* main menu */

#define DATAMENU 2              /* data menu */
#define LOADFILE 21             /* load a file */
#define SAVEFILE 22             /* save a file */
#define EDITFILE 23             /* edit a file */
#define NEWFILE 24              /* new data file */
```

```
#define SIMULAT 3                    /* run simulation */

#define GRAPH 4                      /* create graphics */

#define QUIT 5                       /* quit from program */


/*============* types of data *================================*/

typedef int zoneint[MAXZONES];          /* array of integers */
typedef float zonereal[MAXZONES];       /* array of reals */

typedef zoneint intmatrix[MAXLINES];    /* matrix of integers */
typedef zonereal realmatrix[MAXLINES];  /* matrix of reals */

typedef int lineint[MAXLINES];          /* array of integers */
typedef float linereal[MAXLINES];       /* array of reals */

int simlen,                             /* simulation length */
    mandel,                             /* manager decision
                                           delay */
    pepdel;                            /* people reaction delay */
float operhrs,                          /* hours of operation */
      busseats,                         /* seats per bus */
      cststs,                           /* constant cost for
                                           system */
      cstpml,                           /* cost per mile */
      cstphr,                           /* cost per operation
                                           hour */
      gaspr,                            /* gas cost for auto
                                           trip */
      tickt;                           /* fare price of the bus */

typedef struct
{
  int numzones;                         /* number of zones */
  float frequency;                      /* frequency of the main
                                           line */
  zonereal length;                      /* length of the zones */
} MAIN_DATA;                            /* data for main line */

typedef struct
{
  int numlines;                         /* number of lines */
  lineint linesexist,                   /* what lines exist */
          linenum,                      /* number of line */
          linezones;                   /* how many zones each line
                                           has */
  linereal frequency;                   /* frequency of each
                                           line */
  realmatrix length,                    /* length of the zones */
             ivt_a,                    /* in veh time for auto */
```

```
                ivt_b,                      /* in veh time for bus */
                ovt_b,                     /* out veh time for bus */
                egress_c,                   /* egress time for car */
                egress_t,                   /* egress time for
                                               transit */
                parkcost_c,                /* parking cost for car */
                autoph,                     /* autos per household */
                persph,                     /* autos per household */
                income,                     /* income */
                worktrips;                  /* work trips for each
                                               zone */
} LINES_DATA;                               /* data structure of data
*/


/*============* variables *=====================================*/

MAIN_DATA maindata;                      /* structure for main line
                                            data */

LINES_DATA linesdata;                   /* structure for lines data */


/*============* functions *=====================================*/

void main();

void set_up();
void swindow();
void cwindow();

int read_int();
float read_real();
void read_str();

void reset_all();

int main_menu();
int data_menu();

void load_file();
void save_file();
void edit_file();
void new_file();

void draw_line();
void show_zone();
void show_line();

void simulation();
```

```
float wait();
void riders_calc();
void frequences_calc();

void graphics();
void do_graphics();

/*============* end of the file *================================*/
```

APPENDIX B

REFERENCES

REFERENCES

1. Stephanedes, Y.J. (1978). A feedback dynamic model for transportation systems. 2nd International Symposium on Large Engineering Systems. University of Waterloo and IEEE, Waterloo, Ontario, Canada.

2. Stephanedes, Y.J. and T.J. Adler (1979). Forecasting experiments for rural transit policy makers. Transportation Research Record 718, 42-44.

3. Stephanedes, Y.J. (1979). Transportation policy modeling using dynamic systems methodologies. Systems, Man, and Cybernetics International Conference, 431-435, IEEE, Denver, October.

4. Stephanedes, Y.J. (1981) Modeling dynamic operating and financial strategies in transportation. Systems Dynamics and Analysis of Change, 19-32.

5. Stephanedes, Y.J. and T.C. Bountis (1982). Nonlinear dynamics in transportation demand and supply. 10th IMACS World Congress on System Simulation and Scientific Computation 1, 17-18, Montreal, Canada.

6. Stephanedes, Y.J. and D.M. Eagle (1982). "Analyzing the impacts of transportation policies on rural mobility and economic development." Fin. rep. DOT-RC-92019, 277 pp., Civil and Mineral Engineering, University of Minnesota.

7. Stephanedes, Y.J., P. Michalopoulos, D. Gabriel and H. Hanna (1982). "Development and implementation of dynamic methodologies for evaluating energy conservation strategies." Fin. rep. MN-11-0004, 184 pp., Civil and Mineral Engineering, U. of Minn.

8. Stephanedes, Y.J. (1983) Dynamic methodologies for assessing transport policy impacts. World Conference on Transport Research 2, 1008-1018, Hamburg, Federal Republic of Germany, April.

9. Stephanedes, Y.J. (1985). SOLON: Microcomputer interactive method for evaluating and improving transit route performance. Transportation Research Record 1013, 1-9.

10. Stephanedes, Y.J., P. Michalopoulos, and T. Bountis (1985). Dynamic transit scheduling under efficiency constraints. Transportation Research 19B:2, 95-111.

11. Grebner, T.C. (1990). Estimation of dynamic transfer volumes in transit networks. M.S. thesis, 118 pp., Civil and Min. Engineering, U. of Minn.