DEVELOPMENT AND TESTING OF DYNAMIC TRAFFIC ASSIGNMENT AND SIMULATION PROCEDURES FOR ATIS/ATMS APPLICATIONS

Technical Report DTFH6 1-90-R-00074-FG

Prepared by

Center for Transportation Research
The University of Texas at Austin
Austin, Texas 787 12- 1075

Prepared for

U.S. Department of Transportation
Federal Highway Administration
Office of Safety and Traffic Operations
Research and Development
6300 Georgetown Pike
McLean, Virginia 22106

June 1993
Revised June 1994

Foreword

The ability to model vehicular flows in traffic networks under real-time information, and to provide system users with route guidance information, constitute essential methodological components required to support the successful operation of Advanced Traveler Information Systems and Advanced Traffic Management Systems. This report describes the methodologies and procedures developed through a contract to the University of Texas at Austin, in collaboration with the University of Maryland, to address these essential needs. Specifically, a simulation-assignment methodology has been developed to describe user's path choices in the network in response to real-time information, and the resulting flow patterns that propagate through the network, yielding information about overall quality of service and effectiveness, as well as localized information pointing to problem spots and opportunities for improvement. This methodology is intended for use off-line for evaluation purposes, or on-line for prediction purpose in support of advanced traffic management functions. In additional, algorithmic procedures have been developed to determine the best paths to which users should be directed so as to optimize overall system performance. Powerful extensions to incorporate multiple user classes are also described as well as strategies for real-time operational implementation. Taken collectively, and individually, the procedures described in this report constitute a significant advancement in the state of the art of traffic modelling and dynamic network analysis, and an important step towards the realization of intelligent vehicle-highway systems.

Notice

Technical Report Documentation Page

1. Report No. DTFH61-90-R--00074-FG	2. Government Accession No.	3. Recipent's Catalog No.
4. Title and Subtitle Development and Testing Of Dynamic Traffic Assignment And Simulation Procedures For ATIS/ATMS Applications		5. Report Date June 1993; Revised June 1994
		6. Performing Organization Code
		8. Performing Organization Report No. DTFH6 l-90-R--00074-FG
7. Author(s) Hani S. Mahmassani, Ta-Yin Hu, Srinivas Peeta & Athanasios Ziliaskopoulos		10. Work Unit No. (TRAIS)
9. Performing Organization Name and Address Center for Transportation Research The University of Texas at Austin Austin, Texas 78712-1075		11. Contract or Grant No. DTFH61-90-R--00074-FG
12. Sponsoring Agency Name and Address U.S. DOT Federal Highway Administration Office of Safety and Traffic Operations Research and Development 6300 Georgetown Pike McLean, VA 22106		13. Type of Report and Period Covered: FINAL
		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

This report describes the dynamic assignment and traffic simulation models developed to support the functional operating core of ATIS (Advanced Traveler Information Systems) and ATMS (Advanced Traffic Management Systems). Two principal functions are addressed by the assignment-simulation models in this context. The first consists of determining, in real-time, the network paths to which the drivers should be directed in going toward their destination, so as to achieve system-level objectives. The second is the prediction or description of the time-varying link flow patterns that result from the path choices made by motorists in response to supplied route guidance or other forms of information. The procedures developed to address the first capability incorporate the second capability as a sub-problem. The methodologies developed in this report represent a significant advance in the state-of-the-art of network assignment and traffic simulation, and form the basis of the new generation of modelling approaches and advanced methodologies needed to support emerging IVHS technologies.

17. Key Words: ATIS/ATMS dynamic traffic assignment, net-work modelling, route guidance	18. Distribution Statement No restrictions. This document is available to the public through the National Technical Information Service, Springfield, Virginia 22161		
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages	22. Price

Form DOT F 1700.7 (8-72)

Preface

This report describes the procedures developed under Tasks B and C of project DTFH61-90-R-00074, “Traffic Modelling to Support Advanced Driver Information Systems (ADIS)“ The objectives of these tasks are to develop dynamic assignment and traffic simulation models that can be used to support the functional operating core of ATIS (Advanced Traveler Information Systems) and ATMS (Advanced Traffic Management Systems).

Two principal functions are addressed by the assignment-simulation models in an ATIS/ATMS context. The first consists of determining, in real-time, the network paths to which the drivers should be directed in going toward their destination, so as to achieve system-level objectives. The second is the prediction or description of the time-varying link flows patterns that result from the path choices made by motorists in response to supplied route guidance, traffic control actions, or other forms of information.

The procedures developed to address the first capability incorporate the second capability as a sub-problem. The dynamic traffic assignment algorithm developed to address the first capability is an interactive procedure, within which a traffic simulation capability is required.

This report first describes DYNASMART, a simulation-assignment framework that meets all functional requirements for ATIS/ATMS applications. It satisfies the second capability indicated above. In addition to describing its conceptual and mathematical aspects, implementation issues are extensively discussed, and results of application examples are presented.

Next, the algorithmic procedures for the dynamic assignment capability described above are developed and documented, including special requirements for multiple user classes. A rolling horizon framework for on-line implementation is developed. Computational tests are also presented.

The report also describes the extensive path processing capabilities developed in conjunction with the above procedures.

The methodologies developed in this report represent a significant advance in the state-of-the-art of network assignment and traffic simulation, and form the basis of the new generation of modeling approaches and advanced methodologies needed to support emerging IVHS technologies.

METRIC/ENGLISH CONVERSION FACTORS

ENGLISH TO METRIC

LENGTH (APPROXIMATE)

1 inch (in) = 2.5 centimeters (cm)
1 foot (ft) = 30 centimeters (cm)
1 yard (yd □ 0.9 meter (m)
1 mile (mi) = 1.6 kilometers (km)

AREA (APPROXIMATE)

1 square inch (sq in, in^2 = 6.5 square centimeters (cm^2)
1 square foot (sq ft, ft^2 = 0.09 square meter (m^2)
1 square yard (sq yd, yd^2) = 0.8 square meter (m^2)
1 square mile (sq mi, mi^2) = 2.6 square kilometers (km^2)
1 acre = 0.4 hectares (he) = 4,000 square meters (m^2)

MASS - WEIGHT (APPROXIMATE)

1 ounce (oz) = 28 grams (gr)
1 pound (lb) = .45 kilogram (kg)
1 short ton = 2,000 pounds (Lb) = 0.9 tonne (t)

VOLUME (APPROXIMATE)

1 teaspoon (tsp) = 5 milliliters (ml)
1 tablespoon (tbsp □ 15 milliliters (ml)
1 fluid ounce (fl oz) = 30 milliliters (ml)
1 cup (c) = 0.24 liter (l)
1 pint (pt) = 0.47 liter (l)
1 quart (qt) = 0.96 liter (l)
1 gallon (gal) = 3.8 liters (l)
1 cubic foot (cu ft, ft^3) = 0.03 cubic meter (m^3)
1 cubic yard (cu yd, yd^3) = 0.76 cubic meter (m^3)

TEMPERATURE (EXACT)

[(x-32)(5/9)] ^{o}F □ y ^{o}C

METRIC TO ENGLISH

LENGTH (APPROXIMATE)

1 millimeter (mm) = 0.04 inch (in)
1 centimeter (cm) = 0.4 inch (in)
1 meter (m) = 3.3 feet (ft)
1 meter (m) = 1.1 yards (yd)
1 kilometer (km) = 0.6 mile (mi)

AREA (APPROXIMATE)

1 square centimeter (cm^2) = 0.16 square inch (sq in, in^2)
1 square meter (m^2) = 1.2 square yards (sq yd, yd^2)
1 square kilometer (km^2) = 0.4 square mile (sq mi, mi^2)
1 hectare (he) = 10,000 square meters (m^2) = 2.5 acres

MASS - WEIGHT (APPROXIMATE)

1 gram (gr) = 0.036 ounce (oz)
1 kilogram (kg) = 2.2 pounds (lb)
1 tonne (t) = 1,000 kilograms (kg) = 1.1 short tons

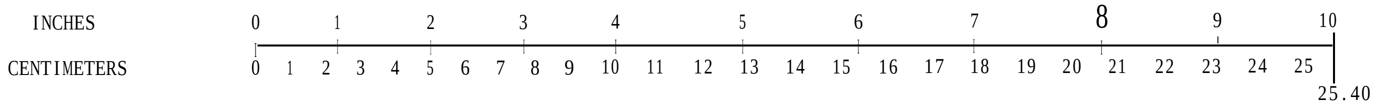
VOLUME (APPROXIMATE)

1 milliliters (ml) □ 0.03 fluid ounce (fl oz)
1 liter (1) = 2.1 pints (pt)
1 liter (l) = 1.06 quarts (qt)
1 liter (l) = 0.26 gallon (gal)
1 cubic meter (m^3) = 36 cubic feet (cu ft, ft^3)
1 cubic meter (m^3) = 1.3 cubic yards (cu yd, yd^3)

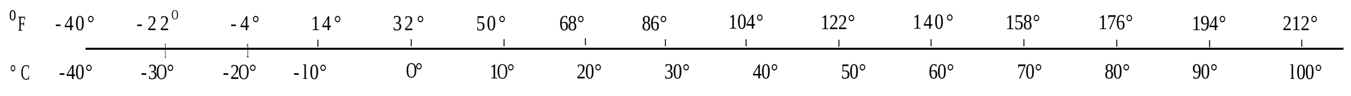
TEMPERATURE (EXACT)

[(9/5) y + 32] ^{o}C □ x ^{o}F

QUICK INCH-CENTIMETER LENGTH CONVERSION



QUICK FAHRENHEIT-CELSIUS TEMPERATURE CONVERSION



°F -40° -22° -4° 14° 32° 50° 68° 86° 104° 122° 140° 158° 176° 194° 212°
°C -40° -30° -20° -10° 0° 10° 20° 30° 40° 50° 60° 70° 80° 90° 100°

For more exact and or other conversion factors, see NBS Miscellaneous Publication 286, Units of Weights and Measures. Price $2.50. SD Catalog No. C13 10286.

TABLE OF CONTENTS

List of Tables

List of Figures

CHAPTER 1
INTRODUCTION

MOTIVATION

Applications of advanced technologies in telecommunications, information technology, microprocessors and automation to intelligent vehicle-highway systems provide new opportunities to improve the performance of traffic networks under both recurrent and non-recurrent congestion. For instance, Advanced Traveler Information Systems (ATIS) and Advanced Traffic Management Systems (ATMS) will provide drivers the capability to communicate with the network control center on a real-time basis.

However, the sophistication in technological and hardware capabilities needs to be matched by more powerful methodological and algorithmic constructs than presently available, especially for real-time control in large-scale traffic systems. This report describes the procedures developed to address the critical need for dynamic route assignment and associated real-time network traffic simulation capabilities.

PROBLEM DEFINITION

The problem addressed in this study consists of the specification and development of dynamic network assignment capabilities and associated traffic performance simulation capabilities that will be necessary to achieve the potential of in-vehicle route guidance Advanced Traveler Information Systems (ATIS) in conjunction with Advanced Traffic Management Systems (ATMS) for improving the productivity and efficiency of traffic networks under recurrent and non-recurrent congestion. The dynamic assignment capabilities required must serve the following principal functions:

1. Allow a central controller, with partial or complete information about time-dependent origin destination (O-D) trip desires as well as current link status conditions (loadings, prevailing link travel times, capacity reducing incidents), to route all trips from their current position (including initial origins and intermediate locations) to their respective destinations so as to achieve system-wide objectives, subject to certain constraints. In other words, the controller seeks to direct users to routes that somehow “optimize” the overall performance of the system, subject to reasonableness and fairness constraints for individual users. This information would form the basis of route guidance instructions to be provided to suitably equipped vehicles on a real-time basis. This capability then reflects a *normative* perspective, and gives rise to a *system optimal* dynamic assignment problem.

In solving this problem, the controller needs to consider the presence of multiple classes of users in terms of access to information, types of available information, and behavior in response to this information, This capability would be used primarily on-line for the above purpose, or off-line to determine initial assignments and routing schemes for the routine and historically known trip patterns, which would subsequently be updated on-line.

2. Allow the controller or analyst to determine, for known O-D trip desires, the time varying link flow patterns that result from the path choice decisions made by motorists, in response to real-time information supplied by the ATIS controller. This on board information might consist of specific route guidance instructions, or of prevailing and/or predicted link trip times, subjected to varying degrees of on-board and/or central processing. *This descriptive* assignment capability is needed off-line to evaluate alternative traffic control schemes, information supply strategies, and/or normative routing and assignment approaches, as well as on-line in connection with a model system to determine what information to provide to motorists. This capability is also needed as a support function for Advanced Traffic Management Systems (ATMS).

Both types of assignment capabilities require the network traffic simulation capability to determine the principal figures of merit that describe the performance of the system, particularly the link trip times, for a given dynamic assignment pattern (i.e., time-dependent link flow patterns), for both on-line and off-line use.

In addition to the conceptual and algorithmic aspects of the above models, the computational issues associated with their implementation for real-time operation constitute an integral element of the problem. In particular, the development of algorithmic procedures must consider the issue of computational efficiency in novel computing architectures with varying forms and degrees of parallelism.,

OBJECTIVES AND STRUCTURE OF THE REPORT

This report describes the conceptual, mathematical and algorithmic aspects of the procedures developed to provide the dynamic assignment and associated simulation functions described in the previous section. In addition, it describes the implementation of these procedures into computer code, and gives illustrative results of computational applications to test networks.

The review of existing procedures for dynamic assignment and traffic simulation, and their limitations vis-a-vis ATIS/ATMS applications have been presented in a separate project report (Mahmassani, et al., 1992), and will not be repeated here.

The DYNASMART simulation-assignment framework, which addresses the second (descriptive) capability stated in the previous section, is described in Chapter 2. The functional requirements for ATIS/ATMS applications, as set forth by the project Statement of Work prepared by FHWA, are first reviewed, as these have guided the development of the various capabilities of the modelling framework. Chapter 3 focuses on implementation and computational issues, and reports the results of several numerical experiments with DYNASMART.

Chapters 4 and 5 address the dynamic assignment capability stated first in the previous section. Chapter 4 develops the conceptual and mathematical formulations of the problem, and describes the algorithmic procedures designed for the solution of both System Optimal (SO) and User Equilibrium (UE) versions of the problem, for a single class of users. Results of computational experiments are also discussed, highlighting the potential benefits of SO assignment.

Chapter 5 extends the procedures described in Chapter 4 to the more general and realistic case of multiple user classes. In this case, only a fraction of users receive SO information, while others may follow UE principles or other behavioral rules. A rolling horizon framework for the on-line real-time application of these procedures is also described.

Chapters 6 and 7 focus on the path processing procedures developed as essential components of both the DYNASMART simulation-assignment framework and the normative dynamic assignment models. Several types of path processing needs are encountered in these problems, including the computation of k-shortest paths and the computation of time-dependent shortest and least-cost paths. In addition, special implementation issues associated with turning movements and multiple user classes are also addressed. Because of the computational intentiveness of path calculations in the context of the assignment and simulation procedures, particular attention is directed at optimizing their computational performance.

Finally, concluding comments are presented in Chapter 8.

CHAPTER 2
DYNASMART

INTRODUCTION

This chapter documents the development of DYNASMART, a network assignment-simulation modelling framework designed to assign time-varying traffic demands and model the corresponding traffic patterns to evaluate overall network performance of ATIS and/or **ATMS.** In its present form, DYNASMART is primarily a descriptive analysis tool for the evaluation of information supply strategies, traffic control measures and route assignment rules at the network level. However, it is evolving towards a model that may be executed on-line in quasi real-time to support the functions of the system controller in the ATIS/ATMS. The model is designed to meet functional requirements set forth by FHWA for ATMS/ATIS applications, including sensitivity to a wide range of traffic control measures for both intersections and freeways, capability to model traffic disruptions due to incidents and other occurrences, representation of several user classes corresponding to different vehicle performance characteristics, different information availability status and different behavioral rules. DYNASMART is based on the assignment-simulation model developed by Mahmassani and Jayakrishnan (1990, 1991) at the University of Texas at Austin. The structure, capabilities and principal modelling features of DYNASMART are examined in this chapter. Implementation issues, numerical results and computational tests are discussed in Chapter 3.

Functional Requirements

Traffic simulation models for surface street networks and for freeways have historically developed independently; therefore, these models by themselves are not appropriate to evaluate a traffic system or to predict traffic patterns for an ATIS/ATMS. A traffic simulation model for use in connection with ATIS/ATMS should be able to simulate both freeways and arterial streets as an integrated network. The control strategies for different sub-networks can thus be coordinated and operated efficiently. In addition, there are several functional requirements set forth by FHWA for ATIS/ATMS applications that need to be incorporated in the simulation-assignment model. These are summarized as follows:

A. Functional Requirements for Surface Streets

1. Realistic representation of changes in traffic signal control.

2. Capable of modelling various types of link geometric configurations.
3. Precise simulation of traffic related effects.
4. Concise but precise measurement of the system effectiveness on both a link specific and network wide bases.
5. Capable of simulating trip generation and attraction centers, and bus operations as well as related facilities.

B . Functional Requirements for Freeway Systems

1. Realistic representation of traffic characteristics and geometric configurations.
2. Detailed simulation of ramp flow characteristics and traffic control strategies.
3. Concise output statistics for measurement of the effectiveness on both a link specific and network-wide bases.

C. Special Functional Requirements for Use in ATIS systems

1. Able to simulate traffic flows at the individual vehicle/driver level.
2. Able to model the route choice behavior of drivers with and without the access to ATIS systems.
3. Capable of accepting data from both the surveillance and historical traffic information at a user specified time period.
4. Responsive to dynamic OD information reported by the ATIS system.
5. Able to track the route and location of each driver who accepts the route advice from the control center.

Features of DYNASMART

DYNASMART has been conceived and developed as an integrated simulation-assignment model which meets and in many respects exceeds the requirements for ATIS and ATMS applications. The flexible framework of DYNASMART allows users to add independent modules for future developments. This modularity and flexibility are essential in a rapidly evolving area such as M-IS, where emerging knowledge on aspects such as user behavior and response to traffic information, as well as new traffic control schemes must be incorporated in the dynamic assignment-simulation framework. Although DYNASMART is still a descriptive model, it can be incorporated within other algorithmic frameworks for optimization purposes, The special features in DYNASMART to date are summarized as follows:

1. Simulate traffic flow at the individual or packet level (according to macroscopic traffic relations).

2. Model the path selection decisions of individual travelers, both en-route and at the trip origins.
3. Model multiple user classes corresponding to different vehicle performance characteristics, information availability, and different behavioral rules.
4. Track the route and location of vehicles, individually or in packets.
5. Model different control strategies for both freeway systems and surface streets.
6. Model traffic disruptions due to incidents and other occurrences.
7. Simulate different signal control strategies
8. Provide the model users with three output levels (system, link and vehicle)
9. Provide an experimental graphic subsystem in Xwindow.

DYNASMART MODEL STRUCTURE

Several approaches have been applied to evaluate traffic systems under ATIS/ATMS, including analytical methods, assignment-based models and simulation-based models. Because of the complexity of the problem and the issues involved, a model for evaluating system performance for ATIS/ATMS with adequate realism needs to combine the concepts and features of simulation and assignment methodologies. In light of the limitations of existing traffic simulation models, as well as those of network assignment models, for ATIS/ATMS applications, four possible development strategies were identified:

1. Interface existing traffic simulation models and network assignment models.
2. Add network path processing and route choice capabilities to an existing traffic simulation package.
3. Add dynamic traffic flow simulation capability to an existing (static, by necessity) network assignment package.
4. Configure a simulation-assignment model structure to best fit the functional requirements of the ATIS/ATMS context.

DYNASMART is based on the fourth strategy. Its overall structure is shown in Figure 1. The approach adopted in DYNASMART integrates traffic flow models, path processing methodologies, behavioral rules and information supply strategies into a single simulation-assignment framework. The input data include time-dependent OD matrices and network data. At the core of the framework, and essential to its flexibility and efficiency, is an integrated network representation system that supports extremely efficient path processing routines, which provide essential information to both user behavior models as well as information supply strategies.

Given the network representation, link characteristics as well as control parameters, the simulation component will take a time-dependent loading pattern and process the movement of vehicles on links, as well as the transfers between links according to specified control parameters. These transfers require instructions that direct vehicles approaching the downstream node of a link to the desired outgoing link. The user behavior component is the source of these instructions, as it determines individual path decisions of users in the network. Alternatively, path decisions may be pre-assigned for some or all users according to a particular assignment scheme, as is the case when DYNASMART is used as a simulator in the context of algorithmic procedures (such as those described in Chapters 4 and 5). The components of DYNASMART are described hereafter.

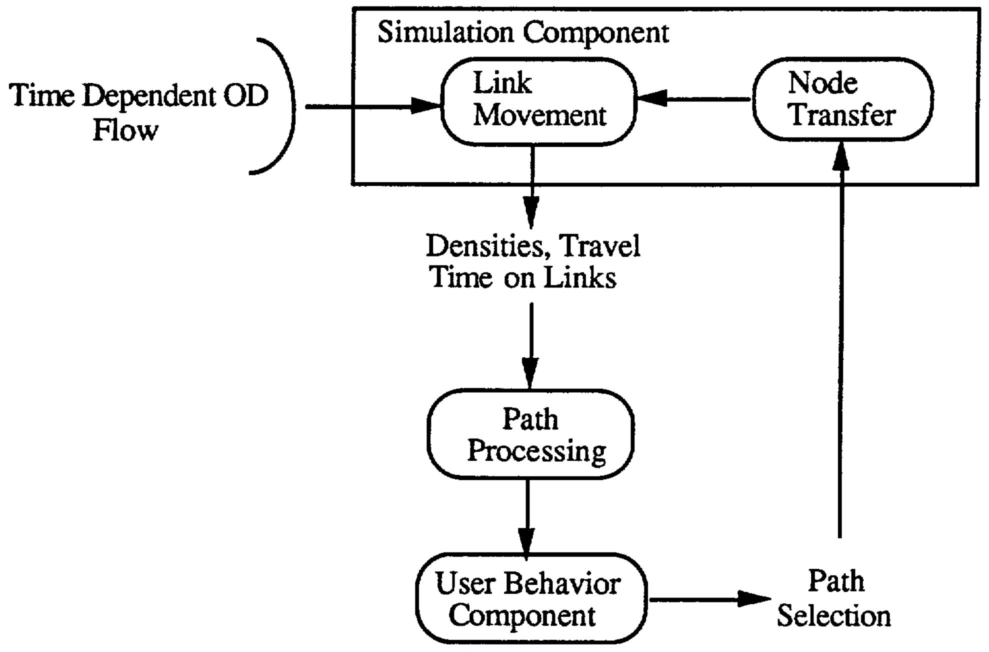


Figure 1. DYNASMART Model Structure

Simulation Component

In DYNASMART, the macroparticle, macroscopic simulation concept is applied in the simulation of mixed traffic. The simulation model is an extension of the macroparticle simulation model (MPSM) (Chang et al., 1985), initially developed as a special-purpose code for experimental studies of commuter behavior dynamics in congested traffic corridors.

Macroscopic simulation models use the traffic stream relationships to describe traffic interactions. In general, this approach is based on a continuum representation of traffic, described in terms of the continuity equation:

$$\frac{\partial q}{\partial x} + \frac{\partial k}{\partial t} = g(x,t)$$

where,

q = flow (vehicles/ hour),

k = concentration (vehicles/mile), and

g = net generation at source/ sink.

The above equation is usually coupled with a speed-density relationship. In addition, macroscopic simulation models typically calculate link flows using the identity $q = k\ v$ (where v is the average speed). The continuity equation, expressed in finite difference form, is solved numerically using discrete time steps. However, for links of finite lengths, moving vehicles according to the $q = k\ v$ identity may lead to physically unrealistic speeds, as discussed in Chang et al. (1985).

For this reason, DYNASMART moves vehicles in discrete bunches or macroparticles, at the prevailing local speeds determined from the speed-density relations. The macroparticle concept is adapted from plasma physics (Leboeuf et al., 1979) which exhibits similar properties in this regard. In previous work, 5 to 20 vehicles were used as a macroparticle (Chang et al.,1985; Mahmassani and Jayakrishnan,1988). In its current implementation, DYNASMART uses a macroparticle of one vehicle, meaning that it effectively track the movement and location of individual vehicles. However, it does not keep track of the microscopic details of individual traffic maneuvers, such as car following.

DYNASMART uses established macroscopic traffic flow models and relationships to model the flow of vehicles through a network. However, whereas macroscopic simulation models do not keep track of individual vehicles, DYNASMART moves vehicles individually or in packets, thereby keeping a record of the locations and itineraries of the individual particles. Multiple user classes with different vehicle performance characteristics are modelled as different packet sizes in DYNASMART. The traffic simulation consists of two primary modules: link movement and node transfer, as described hereafter.

Link Movement

The link movement module consists of a process for moving vehicles on links during every simulation time step or scanning time interval in the simulation. Note that the network's links are subdivided into smaller sections or segments for traffic simulation purposes. The vehicle concentration prevailing in a section over a simulation time step is determined from the solution of the finite difference form of the above continuity equation, given the concentration as well as inflows and outflows over the previous time-step. Using the current concentration, the corresponding section's speeds are calculated according to a speed-density relationship, e.g.,

$$V_i^t = (V_f - V_0) (1 - K_i^t / K_0)^{\alpha} + V_0$$

where,

V_i^t , K_i^t = mean speed and concentration in section i during the t-th time step,

V_f, V_0 = mean free speed and the minimum speed, respectively,

K_0 = jam concentration, and

α = a parameter used to capture the sensitivity of speed to the concentration.

Node Transfer

The node transfer module performs the link to link or section to section transfer of vehicles at nodes. For interrupted link flow, the node transfer allocates appropriately the right of way according to the control strategy at this intersection. It determines the number of vehicles that are traversing each intersection in the network at each simulation time step as well as the number of vehicles entering and exiting the network. The output of the node transfer includes the number of vehicles that remain in queue and the number added to and subtracted from each link section for each simulation time step. A wide range of traffic control measures for both intersections and freeways are reflected in the outflow and inflow capacity constrains of the node transfer module.

User Behavior Component

It is assumed that for different alternative designs of an information supply system, the basic information ultimately available to the drivers will include travel times on alternate routes. The best route available may also be brought to the driver's attention. However, drivers may not be required to follow the route suggested to them. Thus behavioral rules governing travelers' route-choice decisions need to be incorporated, with the flexibility to also model the special case in which drivers actually follow the guidance suggested. Experimental evidence presented by Mahmassani and Stephan (1988) suggests that commuter route choice behavior exhibits a boundedly-rational character. This means that drivers look for gains only outside a threshold, within which the results are satisfying and sufficing for them. This can be translated into the following route switching model:

$$\delta_j(k) = \begin{cases} 1 \text{ . if } TTC_j(k) - TTB_j(k) > \max(\eta_j \cdot TTC_j(k) , \tau_j) \\ 0 \text{ . otherwise} \end{cases}$$

where, for driver j,

δ_j (k) : 1, indicates a route switch; 0, no switch at node k,

$TTC_j(k)$: Trip time from node k to destination on current path,

$TTB_j(k)$: Trip time along the best path,

η_j : Relative indifference threshold, and

τ_j : Minimum improvement needed for a switch.

The threshold level may reflect perceptual factors, preferential indifference, or persistence and aversion to switch. The quantity η_j governs users' responses to the supplied information and their propensity to switch. The value is treated as a random number; when generated, a user is assigned randomly and independently a value for η_j . For convenience, η_j is assumed to follow a triangular distribution, with given mean η and range of $\eta/2$. The minimum improvement τ_j is currently taken to be identical across tripmakers according to user defined values. Nevertheless, all these parameters should be calibrated from field experiments.

Alternatively, DYNASMART can model route choice at a node according to a probabilistic discrete choice function, e.g. of the logit form. As behavioral research results in improved user response models, these can be incorporated within the DYNASMART framework with relative ease because of its modularity and flexibility afforded by the path processing capabilities.

Path Processing

The path processing component of DYNASMART determines the route-level attributes (e.g. travel time), for use in the user behavior component, given the link-level attributes obtained from the simulator. For this purpose, a multiple user class K-shortest path algorithm with movement penalties is interfaced with the simulation model to calculate K different paths for every origin-destination pair. However, in order to improve the model's computational performance, the K-shortest paths are not re-calculated every simulation time step, but only at pre-specified intervals. In the interim, the travel times on the set of K current paths are updated using the prevailing link travel times at each simulation time step, or every few steps to further reduce computational requirements. The K-shortest path algorithms and computational experiments are further described in Chapters 6 and 7. There are two important ways that this path information is used:

1. Initial Routes

At the beginning of trips, non-equipped drivers need to be assigned to specific paths or initial routes. While there is no universally agreed upon process for assigning initial routes, some researchers have suggested user equilibrium or stochastic user equilibrium assignment for these initial routes. In DYNASMART, initial routes are modelled in an explicit way, allocating drivers to the K-shortest paths according to a pre-specified rule. Of course, when DYNASMART is used as a simulator in conjunction with an algorithmic search procedure, initial paths may be determined by the search. In practice, such assignments for some vehicles may also be available from historical information based on actual measurements.

2. Current Path Information

Current path information forms the basis of driver path choice decisions at every node according to the user behavior component module. In its present version, only current trip times are available to drivers. The current path information is used in equipped vehicles as well as in Variable Message Signs (VMS) route control module. The latter is explained further in the VMS sections. A time-dependent K-shortest path routine has also been developed and could be incorporated within DYNASMART to simulate anticipatory information supply strategies. Such "anticipatory" strategies are now provided with the system optimal, user equilibrium or multiple user class assignment algorithms discussed in Chapters 4 and 5. Additional anticipatory strategies with predicted time-dependent trip times can also be easily implemented if a data fusion and prediction function is provided (in a separate module).

TRAFFIC SIMULATION IN DYNASMART

DYNASMART uses macroscopic traffic models to quantify interactions among vehicles and calculate movements of vehicles along links. However, there are features that need to be included in order to capture traffic complexities and provide essential capabilities for **ATIS/ATMS** applications. This section addresses these features in the modeling **process.**

Table 1. Traffic Control Strategies in DYNASMART

	Surface Street	Freeway System
I.	**Control Types**	
	a. No Control	a. Ramp metering
	b. Yield Control	b. Changeable message signs
	c. Stop signs	
	d. Signal Control (green,red,amber time, cycle time, offsets, phases)	
	Pretimed	
	Pretimed Coordinated	
	Multidial pretimed	
	Actuated (full)	
II.	**Geometric Configurations**	
	a. Saturation Flow Rate	a. Number of lanes
	b. Number of Lanes	b. Capacity
	c. Number of Approaches	c. HOV lanes
III.	**Measure of Effectiveness**	
	a. Average Speed	a. Average Speed
	b. Average Travel Time	b. Average Density
	c. Average Delay	c. Average Ramp Queue Length

Traffic Control Elements

DYNASMART provides the ability to explicitly model an array of control elements, listed in Table 1. The major element for surface streets is signal control, which includes pretimed control and actuated control. Ramp metering and variable message signs (VMS) are the major controls for the freeway system. The geometric configurations and measures of effectiveness which are to be included are also listed in Table 1. The following sections address these elements in detail.

Capacity Control

The node transfer is designed to simulate the input and output flows of vehicles on each approach at intersections operating under a number of control strategies. It calculates the number of vehicles that traverse each intersection in the network during each simulation time step as well as the number of vehicles entering and exiting the network. Several concepts regarding the modelling of vehicle flows in the node transfer are discussed hereafter, in particular: outflow and inflow capacity constraints, equivalent green time for unsignalized intersections, and signalized control.

Outflow Capacity Constraints

The outflow constraints limit the maximum number of vehicles allowed to leave each approach lane at an intersection. These constraints are described in the following equation which states that the total number of vehicles that enter an intersection (from a given approach) depends on the number of vehicles waiting in the queue at the end of the current simulation interval (time step), AT, and the capacity of this approach. The definition of capacity follows the 1985 HCM, and consists of the maximum number of vehicles that can be served under prevailing traffic signal operation.

$$VI_i = \text{Min}\ \{VQ_i; VS_i\}$$

where,

VI_i : maximum number of vehicles that can enter the intersection during A T,

VQ_i : number of vehicles in queue on link i at the end of A T,

VS_i : maximum number of vehicles can enter the intersection during A T, i.e. Gi Si

Gi : remaining effective green time during A T,

S_i : saturation flow rate, and

A T : the simulation interval.

Inflow Capacity Constraints

The inflow constraints determine the maximum number of vehicles allowed to enter a link. These constraints bound the total number of vehicles from all approaches that can be accepted by the receiving link; they include the maximum number of vehicles from all upstream links wishing to enter link j, the available physical space constraint on the outbound link and the section capacity constraint of link j.

$$VO_j = \text{Min} \{ \sum_{k \in U} VI_{kj} , VE_j , C_j \Delta T \}$$

where,

VO_j : number of vehicles that can enter link j

U : set of inbound links into link j (i.e. in the backward star of j)

VI_{kj} : number of vehicles wish that to move from k to j

VE_j : the available space on link j

C_j : approach capacity of link j

Signal Control

Signal control can be separated into pretimed signal control, pretimed coordinated control, multidial pretimed signal control, and actuated signal control. All such signal controls are modelled explicitly in DYNASMART. Detailed input data preparation is described in a separate Technical Report (Mahmassani et al. , 1993), and summarized in Appendix A.

Equivalent Green Time for Unsignalized Intersections

DYNASMART uses the equivalent green time concept to allocate the right of way based on the incoming volume at unsignalized intersections. This can be applied to no control, stop sign and yield sign control.

$$GE_i = (CVQ_i / \Sigma\, CVQ_k) \Delta T$$

where,

GE_i : equivalent green time for i-th phase

CVQ_i : critical vehicle volume in queue of i-th phase

ΔT : simulation time step

Greater detail in modelling unsignalized intersections is not warranted for ATIS/ATMS applications because such intersections tend to be relatively uncongested and to serve mostly local traffic needs.

Pretimed Signal and Pretimed Coordinated Control

Input data in this module includes phase number, offset, green time, red time and amber time for every phase. For pretimed signal control, green times are set for every phase according to this data. Since DYNASMART is not intended as an optimizer of signal system control, the model user has to input offsets obtained exogenously from other models to coordinate arterial streets or the network as a whole. Alternatively, optimization modules could be developed for ATMS applications.

Actuated Signal Control

Instead of detecting individual vehicles, DYNASMART uses an appropriate macroscopic method that determines equivalent green times that are updated to reflect prevailing approach volumes. Two alternative methods are provided in the current version of DYNASMART to represent actuated signal control.

In the first method, green splits are apportioned according to Webster's rule for the measured arrival flow rate. This approach attempts to capture the essential features of actuated signal control : "max out" and "gap out". Max out occurs when the green time for a given phase reaches a preset maximum green time; it is modelled explicitly here. Gap out occurs in the field when a preset time elapses with no detector (generally specified to avoid excessively long delays at conflicting approaches) actuations for the phase in progress, resulting in discontinuation of the green for that phase. In the simulation, because detector actuations are not directly simulated, gap out is only approximately emulated under the first method as the provided green time is intended to serve only vehicles present on a particular approach. The input data set in DYNASMART include maximum green time, minimum green time, default cycle length and the other signal data, such as phase number. The equation used in calculating green time under the first method for an actuated signal control emulation is given below. The concept is to allocate green time depending on incoming volume. If the required green time is larger than the maximum green time or smaller than the minimum green time, the maximum or minimum green time is assigned, respectively.

$G_i = (CV_i / \Sigma\, CV_j)\,(\,C - \text{lost time}\,)$

subject to

$\text{Min Green} \le G_i \le \text{Max Green}$

CVi : critical volume for phase i

C : default cycle length

If CVi is less than the maximum number of allowable vehicles, the green time will be reduced accordingly. These calculations are performed at the end of the current cycle. Cycle length will change every cycle. This modeling will be fairly accurate in allocating green time as congestion increases in the network. It will be somewhat less accurate under light traffic conditions; however the dynamic assignment capability in ATMS is of primary concern during congested periods.

In the second method, the green time for a given phase is determined based on the number of vehicles that would have reached the intersection at the end of the current simulation interval. This green is subsequently extended as appropriate each simulation interval until "max out" is reached, or terminated if no longer needed, thereby emulating "gap out". This second method does not require a default cycle length, and may skip a phase altogether if no vehicle demand exists and no minimum green is specified.

Real-Time Signal Control

DYNASMART provides an independent module for real time signal control which gives an interface to update signal parameters during the simulation. These parameters can be controlled by user specified rules or prepared exogenously in advance. The module is intended to assist in testing different real time control strategies.

Communication Interface between Simulation and Path Processing

In DYNASMART, the path processing component utilizes the travel time information generated from the simulation. The travel time information for links is separated into two parts : travel time for vehicle movement and queueing time. Traffic on each link segment is modelled as consisting of two parts (as shown in Figure 2): those vehicles in the upstream, moving part, and those in the downstream, queueing part.

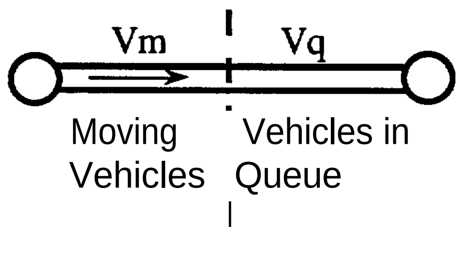


Figure 2. Conceptual Portions on a Link Segment

1. Average Travel Time for Moving Part

Average travel time on link segments for each time step is calculated directly from the traffic stream model used in the simulation. The density and speed is obtained for every simulation interval, then travel time is calculated from available length and associated speed.

2. Average Queue Delay

The average queue delay is considered to be the time for clearing the queue at the queue service rate experienced in the recent past over a certain period (say 3 minutes). The queue discharge calculation considers the average outflow rate and the congestion of downstream links.

Queue Delay = V_q / AS_i

where,

Vq : number of vehicles in queue (vehicles),

AS_i : average flow rate (vehicles/seconds) = $\frac{\sum_{k=t}^{t+T} f_k}{T}$,

f_k : flow rate at k-intervals, and

T : period over which the average queue service rate is calculated.

INCIDENT MODELLING

Incidents are modelled in DYNASMART to reflect accidents, lane closures or other occurrences. Basically, incidents are modelled completely based on external data, and can be specified to occur at any time during the simulation on any link or segment. All incidents cause the reduction of lane capacity. If a whole link or segment is closed, all vehicles (equipped as well as non-equipped) otherwise using the link are diverted to other paths. Some features of incidents modelling in DYNASMART are summarized as follows:

1. Incidents are specified as reductions of link capacity for a specified time period.
2. All calculations are based on user-specified input information about the incident specifics (location, start time, end time, severity).
3. Complex incidents can be modelled as a series of consecutive incidents.
4. Non-equipped vehicles will be diverted for a street closure only when they reach the upstream node of the blocked link.

FREEWAY CONTROL

Freeway management techniques can be categorized as capacity management and demand management. Capacity management, such as ramp control and variable speed control, tries to maximize throughput and maintains a certain level of service. Demand management, on the other hand, attempts to reduce the number of vehicles at the peak period. In DYNASMART, two important elements of freeway management are implemented, namely, entrance ramp control and HOV lanes. In addition, variable message signs (including speed control for mainline regulation) may be modelled, though these are not limited to freeway links.

Ramp Control

Ramp control is the most widely used freeway control measures. Its purpose is to limit the number of entering vehicles in order to maintain a satisfactory level of service within capacity limit. Ramp control includes entrance ramp control as well as exit ramp control. Since exit ramp control is seldom used, it is not explicitly modeled in DYNASMART. However, it could be simulated through other built-in modules, such as lane closure and VMS. According to the Traffic Control Systems Handbook, (FHWA, 1985) there are five types of entrance ramp control: closure, ramp metering, traffic-responsive metering, gap-acceptance merge control and integrated ramp control. The first three methods, explicitly modelled in DYNASMART, are explained as follows:

1. Closure

For ramp closure, drivers need to select alternate routes to their destination. Since equipped vehicles receive current traffic information, they can respond to ramp closure before they reach the ramp. On the other hand, non-equipped vehicles do not have this advantage, so they will choose another route after they reach the closed ramp. However, the VMS can be applied on arterial streets as early warning, so non-equipped vehicles can be diverted prior to their arrival. The choice of alternate route for non-equipped vehicles also depends on driver behavior, and requires an observational basis to develop appropriate path selection rules. DYNASMART provides a flexible way to divert the non-equipped vehicles which allows users to define a k-th best path number or to randomly choose a path from path files.

2. Ramp-Metering

Basically, DYNASMART controls vehicle flow under in-flow and out-flow constraints. In ramp metering, a fixed ramp rate or a dynamic ramp rate that determines the maximum number of entering vehicles can be determined in conjunction with the capacity calculations during a specified time period.

3. Traffic-Responsive Metering

Traffic-responsive metering is directly controlled by the mainline and ramp traffic conditions during the metering period. Occupancy control and demand control are two widely used methods for traffic-responsive metering. ALINEA (Papageorgious et al., 1991), a local feedback control law for on-ramp metering, is implemented in DYNASMART. A typical feedback law is given as follows:

$\text{Rate}(T+l) = \text{Rate}(T) + K_r\ (K_o - OCC)$

K_r : rate adjustment parameter (default value 0.32)

K_o : nominal (target) occupancy (default value 0.2)

OCC : detector occupancy

Rate(t) : max: 35 - 25 vehs/min-lane; min: 5 vehs/min-lane

The given default values of K_r and K_o are from numerical results by Joseph (1993), and are intended for illustrative purposes only.

High-Occupancy Vehicle Priority Control

Priority for high-occupancy vehicles is to provide preferential treatment through HOV lanes for buses and carpools. The purpose of HOV lanes is to encourage carpools or buses in order to reduce overall vehicle demand. Methods of priority control include separated facilities, reserved lanes, and priority access control. In DYNASMART, HOV lanes are part of a traffic network represented by links and nodes. In order to preclude non-high-occupancy vehicles from using the HOV lanes, the travel times on these links are set to infinity for non-HOV vehicles for the path calculation.

LEFT TURN MOVEMENT

Left-turn movements are a critical delay-causing factor in urban networks. However, it is very difficult to model the left turn movement in a macroscopic simulation model. In this section, the left-turn issue is discussed and the modelling process used in

DYNASMART is introduced.

For left-turn movements without a turning phase, the analytical approach is to calculate the blocked time by opposing vehicles flow at the onset of green and then use gap acceptance models to calculate the actual number of vehicles which can pass the intersection during the residual green interval.

Left-turn capacity is determined by several factors : opposing volume, number of lanes of the opposing approach and green time for this phase.

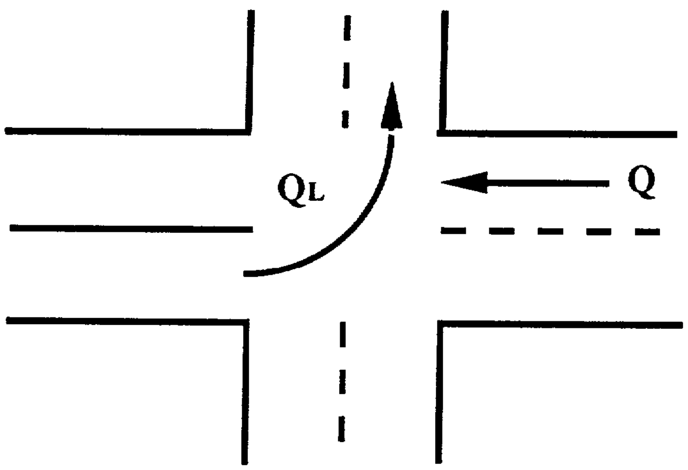


Figure 3. Left Turn Movement

The blocked time from the onset of green that left turning vehicles cannot use is calculated as follows :

Blocked Time:

$T_b = (Q/N) (L1+ L2 + R) / (S - Q/N)$

where,

T_b : blocked time, time blocked by opposing traffic; clear time for queue,

Q : total opposing flow (vehs/hr),

N : # of opposing lanes,

L1 : lost time for opposing traffic,

L2 : lost time for start-up,

R : red time, and

S : saturation flow for opposing traffic.

Then, usable time for left turn vehicles can be calculated as:

$$Tu=G+(Ta-L1)-L2-Tb$$

where,

Tu : usable time of cycle for left-turn (second)

G : Green time

Ta : amber (yellow) time

The maximum number of possible left turn vehicles is equal to:

$n = (Tu/ h) +1$, where h : minimum turning headway (= 2.5 seconds)

Thus, a gap acceptance model (Drew, 1968) can be used to calculate the left turn capacity as follows:

$$QL=(Tu/C)QLT$$

$$QLT=\frac{Qe^{-(Q/3600)*Tc}}{1-e^{-(Q/3600)h}}$$

where,

QL : left turn capacity,

QLT : left turn saturation flow, veh/hr,

Tc : critical gap, seconds, and

h : turning headway, seconds (= 2.5 seconds).

The modeling process for the left turn is complex and not easy to combine with any macroscopic simulation. Therefore, a heuristic modeling process is used to capture effects of left turns in DYNASMART. The process is summarized as follows:

1. Count left-turn vehicles.
2. Calculate maximum flow rate for left-turns;
   This rate can be calculated under different situations:
   a) Protected left turn phase: saturation flow rate.
   b) Permissive phase: from gap acceptance models or established tables.
3. Calculate an average number of left-turn vehicles and also reduce the saturation flow rate for straight and right-turn approaches.

4. Follow outflow- inflow constraints to transfer vehicles from link to link.
5. Calculate the left turn delay for the K-shortest path calculation.

Left-turn capacity estimation determines the number of left-turn vehicles which can enter the intersection without delay due to opposing volume. Different approaches have been used in determining the left-turn capacity. For example, a gap acceptance model is applied in TRANSYT 7F for permissive movement (Wallace et al., 1991). A review of left-turn capacity issues can be found in Lin, et al. (1984). DYNASMART adopted the left-turn capacity values from Lin et al. (1984) that are derived on simulations using TEXAS (Lee et al., 1983) model. The left-capacity is determined by several factors, such as opposing flow, number of opposing lanes, and signal timing). The saturation flow rate for other movements is adjusted according to 1985 HCM. The left-factor in the adjustment is based on four variables, namely, exclusive or shared lanes, type of phasing, proportion of left-turn vehicles, and opposing volume. The left-turn capacity and adjusted saturation flow rate are used in inflow-outflow capacity constraints.

MULTIPLE USER CLASSES

DYNASMART allows for different classes of users with different information availability, and/or behavioral responses and/or traffic performance characteristics. Vehicle classes can differ by vehicle type, network restrictions, and information availability. Since a variety of attributes are generated for vehicles, vehicles are not identical even within the same class. Currently, seven different classes are modelled in DYNASMART for illustration purposes, and more classes can be included. The seven classes are:
1. non-equipped passenger car,
2. non-equipped truck,
3. non-equipped high occupancy passenger car,
4. equipped passenger car,
5. equipped truck,
6. equipped high occupancy passenger car, and
7. bus.

All the equipped vehicles follow the rules stipulated in the user decisions component. In the current version, the default is the boundedly rational behavior rule discussed earlier, with a relative indifference band and a minimum threshold value. Different vehicle sizes are modelled as packets of different passenger car units, specified by the user. The packet size is used in calculating concentration, available capacity, inflow and outflow constraints.

With this ability, DYNASMART can model virtually any network restrictions, such as turning prohibitions, and special facilities, such as bridges. The HOV concept was described in a previous section. Bus operation is discussed in a later section.

VARIABLE MESSAGE SIGNS (VMS)

One way to provide dynamic route information to drivers is by means of Variable Message Signs (VMS) where visual word, number, or symbolic display can be electronically or mechanically varied according to current traffic conditions. VMS displays can address a considerably wide range of traffic management functions; however, drivers are not usually required to follow all messages from VMS. Since the response of drivers to different VMS is still in need of further study, the use of the VMS module in DYNASMART should be accompanied by a reasonable assumption on driver behavior. The VMS module in DYNASMART includes three parts: speed advisory, route advisory and route warning messages.

Speed Advisory

Speed advisory is mainly used for mainline control of freeway systems; experiments with speed advisory changes have been undertaken in several European countries. Through field experiments, it has been reported that reasonable speed limitation during rush hours increases capacity (Papageorgiou, 1983). In DYNASMART, speed advisory applies at VMS locations when the density exceeds a pre specified value. Then, all the vehicles are assigned the advised speed.

Route Advisory

Route advisory may provide an alternative path for vehicles in order to avoid a congested section. In DYNASMART, the user needs to define a k-th number of paths (or a fixed path) to be displayed, and all the vehicles will follow the new path to their destinations. Of course, a more comprehensive set of response rules will need to be specified in the user behavior component as results of related targeted research become available.

Route Warning

This form of real-time information provides instructions for drivers to divert in advance of a congested section. In the current implementation, the warning message is generated when the concentration of downstream link reaches the maximum concentration, and a given fraction of the vehicles are diverted to other randomly generated routes. The intent is to retain flexibility to incorporate more complete instructions as ongoing research into ATMS strategies produces testable concepts.

BUS OPERATIONS

In DYNASMART, buses are treated as packets with predefined paths; each packet includes two passenger car units. Simulation of bus operations largely depends on related input information namely:

BUS ID : an identifier for bus

Start Time : the start time of the bus

Average stop (dwell) time

Number of nodes in the route

The sequence of nodes

The activities on links

0: no stop

1: stop at the near side

2: stop at the midblock

3: midblock curb stop (or bus bay)

During the simulation, each bus is treated as a packet of two passenger car units. In the link movement, buses are mixed with other vehicles when calculating the prevailing average speeds and concentration. In the node transfer, capacity with 2 pcu's is used for transferring a bus from link to link. Loading and unloading of buses will cause the short term blockage of traffic, and this situation is modeled in DYNASMART according to the locations of bus stops. If the location of the stop is near an intersection, one lane of outflow capacity will be dropped. If the location of a bus stop is in the middle of a block, the short term blockage will be simulated as a short term incident. The blockage time is defined as the average dwell time (user needs to include the average additional time loss due to starting.) According to the 1985 HCM, where the buses stop in a lane that is not used by moving traffic (a curb parking lane, or a bus bay), the time loss to other vehicles is approximately 3 to 4 seconds per bus. The blockage time of midblock curb stop is set as 4 seconds, but can of course be readily changed to reflect actual conditions.

The above modelling of bus operations is not limited to buses, but may also be applied to any other type vehicle with fixed route and schedule.

OTHER CONSIDERATIONS

Driver Compliance Factors

Driver compliance factors are modeled as part of the user decisions component. Several possible rules can be postulated for this behavioral process, and will eventually be developed based on empirical experimental evidence. The ability of DYNASMART to explicitly model multiple user classes on the basis of behavioral provides the necessary flexibility to accomodate a wide range of possible compliance rules.

Output Information

For different analyses, three levels of output can be obtained from DYNASMART:

1. Overall System Performance (the statistics are also reported for different user classes)
    - average overall travel time
    - average travel (moving) time
    - average entry queue time
    - average stop time
    - average travel distance
    - congestion index
    - simulation summary report
2. Selective Information

    -Link

    average speed

    average density

    average end queue

    total number of vehicles passed by

    - Vehicle

    behavior attributes

    travel time

    travel distance

    travelled path
3. Detailed Information

    vehicle trajectories

    signal timing

    path information

    concentration profiles

Detailed input information and sample output information are included in Appendix A.

Graphic Display Systems

An experimental graphic display system has been developed to assist in the display of DYNASMART output information. The graphic system creates a network graph with detailed characteristics, and displays static and dynamic information from DYNASMART on the graph. The graphic system provides the following features:

1. adding and editing traffic road network elements,
2. displaying dynamic simulation results from DYNASMART,
3. displaying the path data, and
4. displaying vehicle trajectories in the traffic road network.

This program requires an X-window Vl 1 R4 server and a C compiler. It was originally developed on a Sun Sparc workstation; however, it is portable to most workstations. With pull down menus and easy-to-use dialog boxes, the system is a convenient tool to view the complicated simulation results.

There are four windows in this system as shown in Figure 4. The *menu bar* window consists of all the available choices: **File, Edit, Show, Sim,** and **Options.** ***The network*** *display window* displays the whole network, with circles as nodes and lines as links. The *dialog window* displays the information request from the computer or shows some instructions for users. The *data and information window* displays the static and dynamic traffic simulation data.

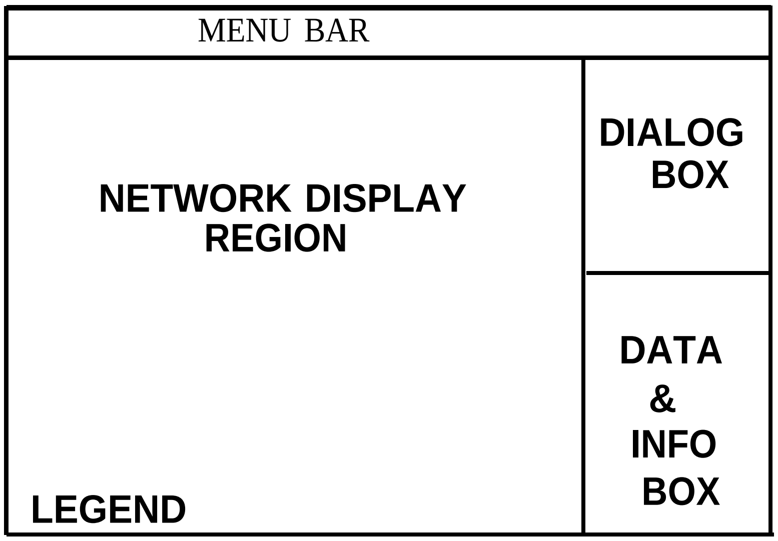


Figure 4. Configuration of the Graphic System

CHAPTER 3
COMPUTATIONAL ISSUES AND NUMERICAL EXPERIMENTS

INTRODUCTION

Numerical experiments are conducted to illustrate different functional features of DYNASMART. These experiments, performed on a hypothetical test network as well as the Austin core network, demonstrate various aspects of DYNASMART. Since DYNASMART is primarily a descriptive analysis tool, detailed input data sets need to be prepared for proper execution. Data sets are described in a separate Technical Report (Mahmassani et al., 1993) and illustrated in Appendix A. Due to the critical role of execution time in real-time control environments, techniques for minimizing execution time are explored in DYNASMART. Some optimization techniques and features of the CRAY FORTRAN (CFT77) language are reviewed.

This chapter contains four sections. The first section is concerned with computational issues, particularly code optimization techniques for FORTRAN program codes. This section may be skipped by the reader interested only in the substantive traffic aspects of DYNASMART with no loss of continuity. The next section presents numerical experiments on DYNASMART, to explain and illustrate various aspects of the program. The third section discusses computational performance of DYNASMART. Extensions and applications are briefly discussed in the last section.

COMPUTATIONAL ISSUES

In this section, some FORTRAN program optimization methods are reviewed and discussed. DYNASMART is written in CRAY FORTRAN. To efficiently utilize the FORTRAN optimization techniques, some important aspects of CF77 and CRAY YMP are listed as follows:

1. Compiler CF77

The CF77 compiling system compiles FORTRAN that conforms to the American National Standards Institute (ANSI) standard, often called FORTRAN 77. The CF77 supports extensions to this standard to offer broader capabilities and to take advantage of the features of CRAY. The CF77 compiling system, composed of FPP, FMP and the CFT77, provides different level of parallel abilities, such as autotasking, and microtasking. Some features of FORTRAN 90 and FORTRAN D are also included in the CF77.

2. CRAY YMP
- 6 ns clock cycle
- 8CPU
- 64 megawords (512 megabytes) of high speed memory
- 512 megawords(4 gigabytes) of SSD (Solid State Storage Disk)
- 55 GB CRAY high Speed disk storage
- UNICOS 6.1 operating system

Types of Optimization

The purpose of optimization is to produce computer code which can be executed in less time. This can be achieved by different ways, such as algorithm optimization and code optimization. Algorithm optimization is to select or develop an algorithm that offers the best possibility of optimum execution within the hardware and software constraints. Code optimization consists of making modifications to an existing code to improve execution time. Optimizing execution of a particular program depends on several factors: utilities available under the compiler and operating system, and the type of code (for example, computation-intensive versus I/O-intensive code).

Parallel Programming on CRAY YMP

CRAY YMP has different levels of parallel processing capabilities in both hardware and software. The evolution of CRAY parallel processing software has followed three implementation levels: macrotasking, microtasking and autotasking. At the macrotasking level, programmers need to modify their code to exploit parallelism by the insertion of library calls provided by CRAY. Microtasking, which inherits the power of macrotasking uses compiler directives instead of library function calls. The most recent implementation, autotasking, combines the best aspects of microtasking with automatic compiling process. In addition, autotasking can exploit parallelism at the Do loop level without extending to subroutines boundaries. DYNASMART fully utilizes autotasking and microtasking within program codes. According to CRAY (SG-3074, p.4, 1990), the goals of autotasking can be summarized as:

1. Detect parallelism automatically and exploit the parallelism without user intervention.
2. Define a syntax by which parallelism is expressed.
3. Define the scope of variables when transforming a program to exploit parallelism.
4. Provide a simple command line interface to autotasking.

FORTRAN Optimization Guidelines

Even with powerful capability of CRAY, FORTRAN codes need to be carefully written to exploit parallelism at the compiler and machine levels. Numerous techniques have been proposed to speed up FORTRAN program codes with CRAY vector and parallel abilities. However, the critical problem is to identify and remove data and control dependencies in program codes. Various such techniques have been applied in the development and prototyping of DYNASMART, and are briefly reviewed in this section. The intent of this discussion is primarily illustrative, to indicate the kinds of computational considerations involved in the development process.

Data dependence is present when the input of a particular statement depends in some fashion on the output of another. Forms of such dependence include flow dependence, antidependence, and output dependence (Aho et al., 1986). Detecting scalar dependencies among statements is relatively straightforward: it involves taking the intersection of the corresponding IN and OUT sets (of variables read and written by the given statements, respectively). The same strategy also works for arrays, but gives coarse dependent results. For more accurate information, subscript analysis of array variables needs to be performed. Testing for dependencies then involves checking whether two subscript expressions could take on identical values during the execution of the program. Some techniques to overcome data dependencies are explained later.

In the presence of complex flow control, focusing on data dependence is not sufficient to transform programs because of the possibility of control dependence. Such dependence exists between two statements when the execution of one can prevent the execution of the other.

Generally speaking, FORTRAN statements can be classified into four groups (Allen et al., 1983):

1. Action Statements -- statements that cause some change in the state of the computation or produce some important side effect. Examples : assignment, read, write, call.
2. Branch Statements -- statements that make an explicit transfer of control to another location in the program. Examples : go to.
3. Iterative Statements -- statements that cause another statement or a block of statements to be iterated. Example : Do loop.
4. Placeholder statements -- statements that take no action but can be used as placeholders for the computation. Example : continue statement.

Control dependence takes place in branch statements, which transfer control to another statement. For the purpose of analysis, branches are categorized into three types:

1. Exit Branch: a branch that terminates one or more loops, as in

```
     DO 100 I = 1,100
     lF(S1) GOT0 200
100        CONTINUE
----
200  CONTINUE
```

2. Forward Branch: a branch whose target occurs after the branch but at the same loop nesting level.

```
     DO 100 I =l,l0
     lF(S1) GOT0 100
     s2
100        CONTINUE
```

3. Backward Branch: a branch to a statement occurring lexically before the branch but at the same nesting level, as in

```
10         Sl
     S2
     IF(S3) GOT0 10
```

In accordance with this classification, IF conversion uses two different transformations to eliminate branches within the program:

1. Branch Relocation

Branch relocation moves branches out of loops until the branch and its target are nested in the same number of DO loops. This procedure converts each exit branch into either a forward branch or a backward branch.

2. Branch Removal

Branch removal eliminates forward branches by computing guard expressions (a Boolean expression which represents the conditions under which the statement is executed) for action statements under their control and conditioning execution on these expressions.

Further techniques can be found in Allen et al. (1983).

Techniques and algorithms for code improvement on the basis of data flow information can be found in Aho et al. (1986). These techniques include global common sub-expression elimination, copy propagation, code motion, and elimination of induction variables. Some practical examples are described in Brawer (1989). A common example of a data dependency is a situation in which a variable (scalar or array element) assigned in one iteration of a loop is read in another iteration, e.g:

```
      do  i=l,n
      sum = sum +a(i)
      end do
```

Data dependencies complicate the parallel execution of programs as they normally require that the statements of the loop be executed in a particular sequential order. Some techniques and examples to eliminate data dependencies are described hereafter (these may be skipped by the reader with no loss of continuity).

Induction Variable

```
c     assign x(4*i)=a(i)                  c  -- rewrite
      integer m,k,i                       m=O
      real x( 1 000),a( 1000)             k=4

      k=4                                 do i =l, 1000
      m=
      do i = l,1000                       m=i*k
            m=m+k                         x(m)=a(i)
            x(m)=a(i)                     enddo
      enddo
```

In general, loops with data dependencies will have to be transformed into a form in which the dependencies do not exist. In the above case, the transformed version substitutes a multiplication for an addition.

Forward Dependency

```
c     forward data dependency
      integer i
      real x(1001)

      do i=1,1000
      x(i)=x(i+l)                 -- x(i) is assigned its value after it is
      end do            -- read

c     incorrect parallel code          c     parallel code
      id = process-fork(nproc)               id = process-fork(nproc)
      do i=l +id, 1000,nproc           do i=l+id,l000,nproc
         x(i)=x(i+l)                         xold(i) = x(i+l)
      end do                                     end do
      call barrier()                                     call barrier( )
                                                 do i = l+id,1000,nproc
                                                         x(i)=xold(i)
                                                 end do
```

In this example, a new array is used to avoid data dependency.

Backward Dependency

Backward dependency is a less tractable kind of data dependency, and requires more advanced techniques to affect data flow. There is no simple way to parallelize the following loop except reformulate the problem.

```
c   example of backward dependency
    integer i
    real a,b,x( 1000)

    do i=2,100
            x(i)=a*x(i)+b*x(i-1)
    end do
```

Break Out of Loop

Some programs manipulate the elements of an array one-by-one so long as some condition holds, then terminate the manipulation when the condition is no longer true.

```
c     example for break out of loop
      integer a( 1000),n,predicate,i

      do i=l,n
         if(predicate(a(i)) .eq. 1) then
                 call  transform(a(i))
         else
                 go to 2
         endif
      end do
2     continue
```

Splittable Loops

Certain kinds of data dependencies can be removed by creating two or more loops from a single loop.

```
c     example of splittable loop
      int i,n,k
      real a( lOOO),b( 1000),c( 1000),f(2000),d
```

```
do i=l,n
        a(i) = b(i) + c(i)*d + f(i+k)
        c(i) = a(i-1)
end do
```

If the statement c(i)=a(i- 1) is changed to c(i)=y(i), there is no longer data dependency. Sometimes this kind of dependency can be avoided by splitting into two loops without data dependency.

```
c     second example                   c       rewrite
      do i=l,n                                 do i=l,n
         a(i)=b(i) + c(i) *d + f(i+k)          x(i) = c(i)
         c(i)=a(i+l)                                   c(i) = a(i+l)
      end do                                   end do
                                               do i =l,n
                                                       a(i)= . . .
```

NUMERICAL EXPERIMENTS

The numerical experiments described in this section are intended primarily to illustrate some basic aspects of DYNASMART. As a general simulation-assignment model, DYNASMART can simulate a variety of scenarios according to specified variables. The series of experiments described hereafter also provide insights into network performance under real-time information for different behavioral assumptions, as well as into the computational characteristics of the program.

Description of the Test Network

Figure 5 depicts the network used in the first set of numerical tests. It consists of a freeway surrounded by a street network. The network consists of 50 nodes and 168 links. All streets are two-directional (represented by two directional links in the graph) and have two lanes in each direction except the entrance and exit ramps that connect the street network to the freeway; these are directed arcs with one lane as shown in the figure. There are 11 on-ramps and off-ramps that connect the freeway with arteries. All other information is given in Table 2.

Demand Levels and Behavioral Experiments

Similar experiments were performed for different scenarios and results are reported in Mahmassani et al. (1992) and Mahmassani and Jayakrishnan (1991). These experiments are primarily included for illustration purposes. The simulation experiments were conducted in two parts. The first part examines system performance under different demand levels. The second part investigates sensitivity of the system's performance, under the information strategy, with respect to two principal factors: (1) the fraction of users with access to information, and (2) the mean relative indifference band, which captures the propensity of users to switch in response to information. Two fundamental assumptions in these experiments arc as follows:

1. All non-equipped vehicles are assumed to have current information before their trips (through radio or TV) , and thus will be assigned to the current best path. After this assignment, they are not allowed to change their routes during the trip.
2. The signal control parameters are not changed during the simulation. The green time is allocated according to arrival flow rates and queue lengths based on preliminary tests.

Demand Levels

The base case loading pattern is set to follow a typical peak-period pattern with rapid build-up and subsequent decrease of the loading vehicles. The vehicles are generated over a 35-minute period and statistics are accumulated for vehicles generated after the first five minutes (initialization period). The average number of generated vehicles is about 367 vehicles per minute, the average trip time is about 3.4 minutes and the average trip distance is 1.32 miles. The number of vehicles and loading pattern are shown in Table 3. The total number of vehicles is about 11,234 which is not expected to cause significant congestion in the test network. The total demand is subsequently multiplied by a factor to examine the variation of trip time with increasing demand. All the parameters are fixed at this stage.

Fraction of Equipped Vehicles

To examine the effect of this fundamental parameter in the large-scale deployment of any in-vehicle information system, DYNASMART allows users to define different fractions of vehicles in different classes. Output information will be generated by vehicle class. In these experiments, three levels are considered 0.0, 0.50, and 1.00. Information availability status is assigned randomly and independently to each vehicle as it is generated, according to the specified fraction.

Mean Relative Indifference Band

The quantity nj in the boundedly rational behavioral rule governs users' responses to the supplied information and their propensity to switch. As noted earlier, we treat it as a random variable; when generated, a user is assigned randomly and independently a value

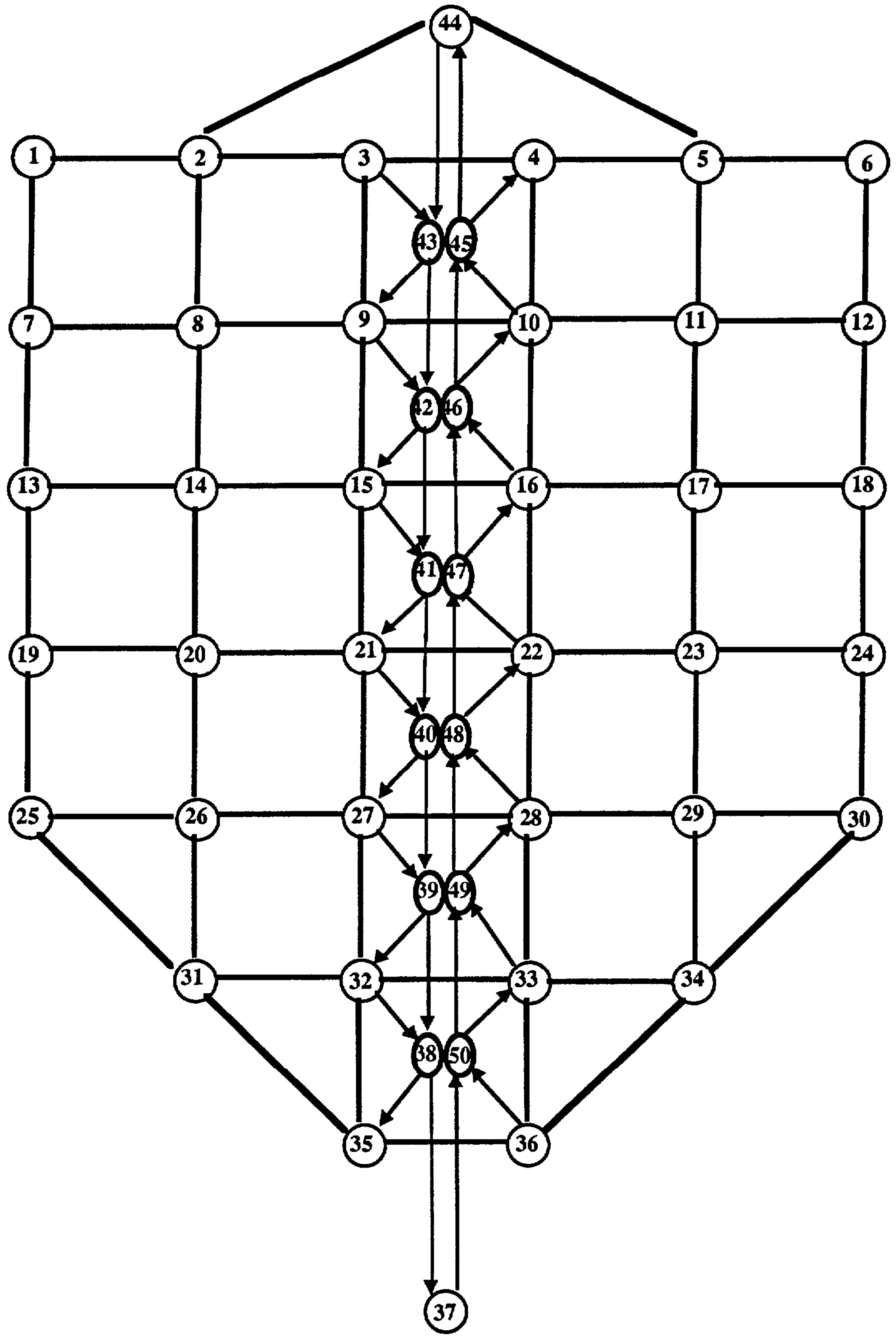


Figure 5. The Test Network Structure

Table 2. Characteristics of the Test Network

SimulationInformation
- start up time : 5 minutes
- period of interest : 5 - 30 minutes
- Max simulation time : 150 minutes

Network :
- Overall Data
  - number of nodes : 50
  - number of links : 168
  - destinations : 10 (2,5,13,18,25,30,35,36,37, and 44)
  - demand zones : 10 (1 to 10, each zone covers 3-4 nodes)
  - ramp control : 11 on ramps
- Link
  - Jam density = 160 vehicles/mile
  - Maximum density = 260 vehicles/mile
  - arterial street
  - length : l/4 mils
  - number of lanes : 2
  - velocity : 30 miles/hr
- freeway
  - length : l/4 miles
  - number of lanes : 2
  - velocity : 55 mph
- on-ramp and off-ramp
  - length : l/4 mils
  - number of lanes : 1
  - velocity : 30 miles/hr
- HOV links
  - length : l/4 miles
  - number of lanes : 1
  - velocity : 55 mph

Signal Data
- No-control : 16
- Retimed control : 26
  - 2-phases operation
  - green time : 25 seconds
  - amber time : 5 seconds
- Actuated Signal Control : 8
  - 2 phases operation
  - min green time : 10 seconds
  - max green time : 25 seconds
  - amber time : 5 seconds

Table 3. The Loading Pattern for the Base Case

Time Interval	# of Vehicles	Cumulative #
5	1205	1205
10	2077	3282
15	2805	6087
20	2142	8229
25	2107	10336
30	855	11191
35	43	11234

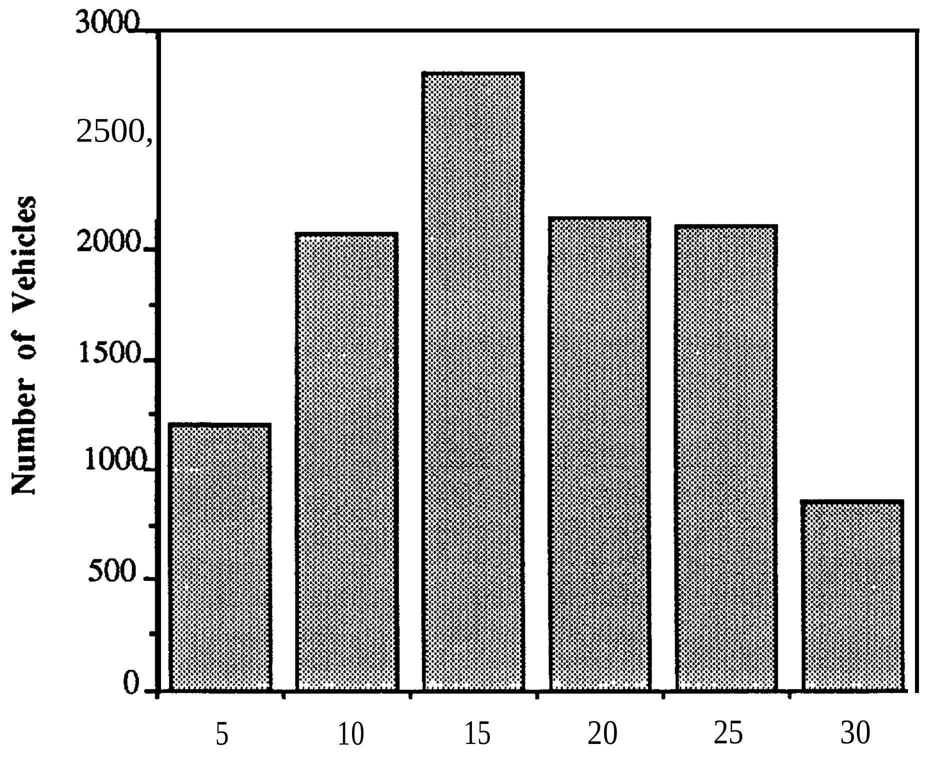


Figure 6. Vehicle Generation Pattern

for nj. For convenience, nj is assumed to follow a triangular distribution, with mean n and range of n/2. The minimum improvement tj in the switching rule expression is taken to be identical across users. In these experiments, the n is set to 0.2 and t is equal to one minute, unless noted otherwise.

Results

Effect of Demand Levels

The purpose of these experiments is to observe the variation in system performance when the demand is increased. None of the vehicles are assumed to be equipped with real-time information systems (0% market penetration). The variation of average trip time (ATT), average stopped time (AST), and average trip distance(ATD) are reported in Table 4. The variation of the systemwide average trip time and average stopped time is shown in Figure 7. As expected, the average trip time increases when the demand is increased. The average trip time for demand factor 2.2 is 15.0 minutes which is four times more than the base case (3.51 minutes). In Figure 7, we also observe that the ATT and AST are highly positively correlated, and that the AST also increases rapidly with increasing congestion. The variation of ATD is shown in Figure 8. The increase in trip distance is relatively small in magnitude. The ATD for the 2.2 demand level case is 1.44 miles, that is 9 % more than the base case (1.32 miles).

Effect of In-Vehicle Information and Behavioral Scenarios

The experiments illustrate the behavioral modelling capabilities of DYNASMART. The results of myopic switching and 0.2 indifference band are reported in Tables 5 and 6, respectively for a 50% market penetration level (equipped vehicles). Similar results were reported in Mahmassani et al (1992) and Mahmassani and Jayakrishnan (1991). In the myopic case (relative indifference band = 0.0 and minimum bound = O.O), equipped vehicles always switch to an alternate path if it offers an improvement in estimated travel time, no matter how small its magnitude. In Table 5, the difference between equipped and non-equipped vehicles becomes larger for higher demand levels. For example, the relative benefit to equipped vehicles is -0.6% for the base case and 9.4% for demand factor 2.0 case, as shown in Figure 9.

Table 4. System Performance Statistics for Different Demand Factors

Demand Factor	1	1.2	1.4	1.6	1.8	2	2.2
Vehicles	11234	13589	15565	18012	20064	22347	24789
Non-tagged	1248	1528	1741	2104	2326	2654	2932
Tagged-Vehicles	9986	12061	13824	15908	17738	19693	21857
ATT (in minutes)	3.51	4.12	5.34	7.16	9.88	12.18	15.03
AST (in minutes)	0.9	1.4	2.39	3.83	5.99	7.62	9.87
ADT (in miles)	1.324	1.333	1.356	1.379	1.397	1.427	1.44

* Tagged vehicles are those generated between minute 5 and minute 35 over the simulation and for which statistics are accumulated.

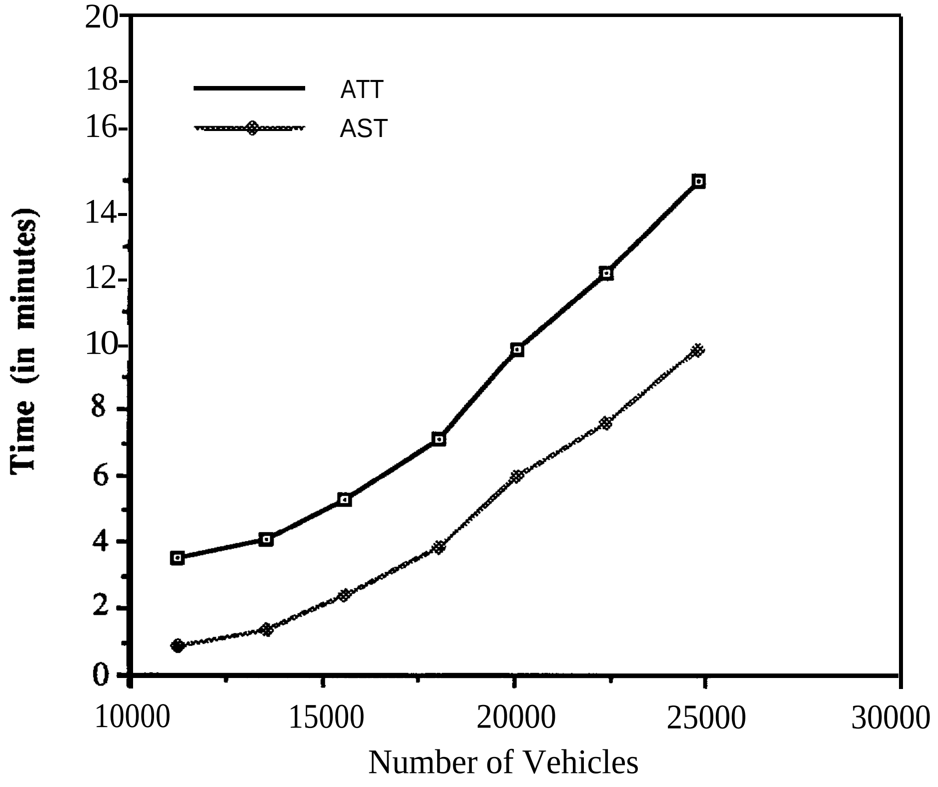


Figure 7. Variation of Average Trip Time (ATT) and Average Stopped Time (AST) for Different Demand Loads

If all equipped vehicles with the same destination switch to the same route, the route is not likely to remain a superior one to use. The variation of ATT and AST compared with the original case (0% market penetration, or no information) is shown in Figures 10 and 11 respectively. In Figure 10, ATT is the curve for the case without real time information (no equipped vehicles in the network), and ATT-1 is the curve for the case with 50% market penetration. We observe that the gap between ATT and ATT-1 becomes larger when the demand factor reaches the 1.6 level, suggesting that the real-time information is more effective in this network when traffic conditions are congested. The comparison of the average stopped times is shown in Figure 11, and exhibits similar patterns.

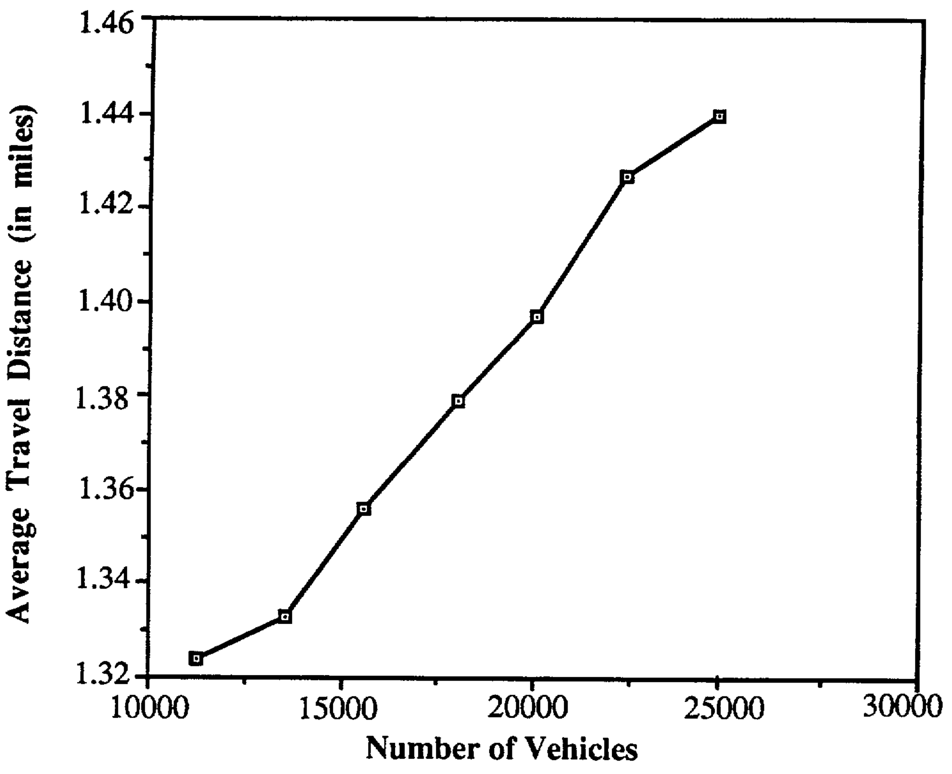


Figure 8. Variation of Average Travel Distance (ATD)

Two important parameters in the behavioral rule considered here determine how drivers respond to real-time information. For η of 0.2 and τ of one minute, the benefit of equipped vehicles over non-equipped vehicles is shown in Figure 12 (which is similar to Figure 9 for myopic switching). When demand in the network is low, the benefit is negative, indicating that switching under this rule does not provide any improvement. However, the benefit is significant as the demand factor reaches or exceeds 1.6, a reasonable congestion level in the test network.

Table 5. System Performance Statistics for the Myopic Case

Demand Factor	1	1.2	1.4	1.6	1.8	2	2.2
ATT (in minutes)	3.37	4.06	4.82	6.75	8.12	10.57	13.9
non-equipped	3.36	4.09	483	6.95	8.31	11.09	14.33
equipped	3.38	4.02	4.81	6.54	7.93	10.05	13.48
AST (in minutes)	0.79	1.34	1.96	3.48	4.6	6.2	8.47
non-equipped	0.78	1.37	1.97	3.64	4.77	6.63	8.83
equipped	0.8	1.32	1.94	3.32	4.44	5.77	8.1
ADT (in miles)	1.321	1.335	1.343	1.374	1.386	1.413	1.45
non-equipped	1.32	1.341	1.34	1.373	1.389	1.412	1.446
equipped	1.323	1.329	1.346	1.374	1.383	1.414	1.453

Table 6. System Performance Statistics for the 0.2-band Case

Demand Factor	1	1.2	1.4	1.6	1.8	2	2.2
ATT (in minutes)	3.37	4.05	4.91	6.8	8.62	11.81	14.33
non-equipped	3.35	4.09	5	7.13	8.96	12.41	14.96
equipped	3.39	4	4.82	6.47	8.28	11.22	13.7
AST (in minutes)	0.78	1.33	2.02	3.55	4.91	7.62	9.21
non-equipped	0.78	1.37	2.1	3.85	5.22	8.14	9.83
equipped	0.78	1.28	1.95	3.26	4.61	7.09	8.59
ADT (in miles)	1.318	1.333	1.346	1.372	1.418	1.396	1.454
non-equipped	1.311	1.334	1.353	1.373	1.414	1.394	1.444
equipped	1.326	1.332	1.339	1.372	1.421	1.397	1.463

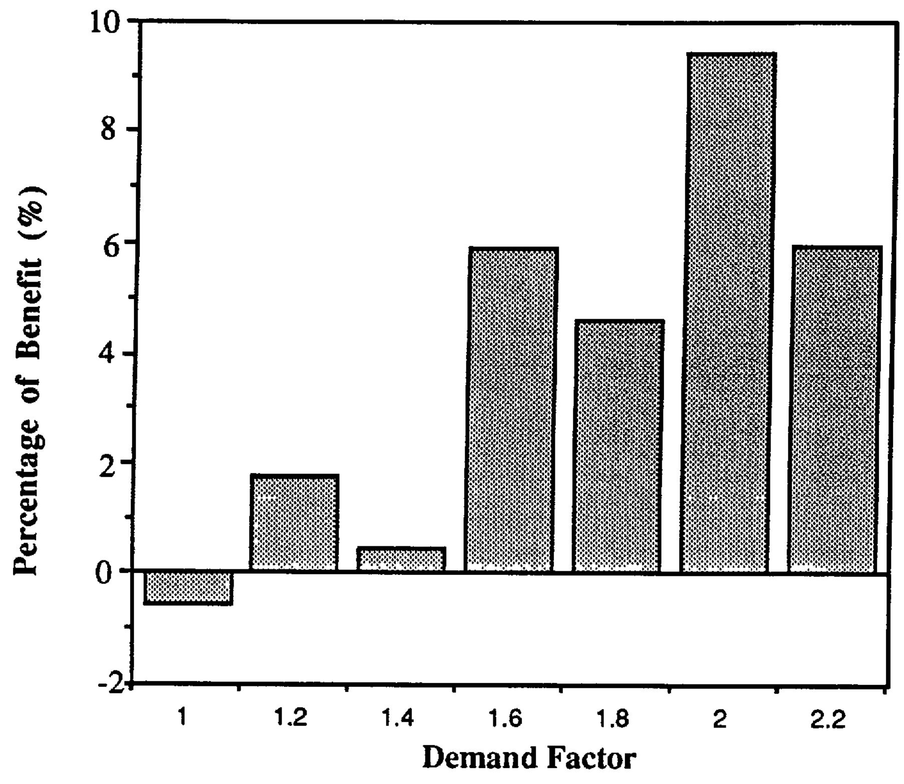


Figure 9. Relative Benefit (in terms of travel time savings) of Equipped Vehicles over Non-Equipped Vehicles under Myopic Switching Rule

Table 7 reports summary statistics on the switching activity. Vehicles with different numbers of switches are reported and the percentage of switching for equipped vehicles is given in the last column. It is interesting that in the myopic cases almost all vehicles make at least one switch, and the percentage of switching is about 90% for all demand levels. The 0.2-band case gives a more reasonable explanation. The percentages for the demand levels considered are 11.77%, 27.47%, 47.49%, 62.29%, 67.77%, 72.90% and 67.66%. While the value of the relative indifference band can be specified from 0.0 to 1.0 in DYNASMART, additional empirical support from behavioral research is necessary to provide definitive values for such models.

Multiple User Classes

To illustrate the capability of DYNASMART to model multiple user classes, four classes of vehicles are specified for the same base scenario considered earlier:

CLASS 1 : non-equipped passenger car, 40%
CLASS 2 : non-equipped truck , 10%
CLASS 4 : equipped passenger car, 40%, and
CLASS 5 : equipped truck, 10%.

All the equipped vehicles have a 0.2 relative indifference band and 1.0 minimum bound. System performance measures, namely the average trip time, average stopped time, and average trip distance are reported in Table 8. The overall average trip time for the MUC case is much worse than the original case as shown in Figure 13. From Table 8, we can see that CLASS 2 always experiences the longest travel time, and CLASS 4 experiences the shortest travel time. The percentages shown for each class in Table 8 are relative to the corresponding overall average values. The average stopped time for trucks is greater in relative terms (to the overall average) than the average trip time.

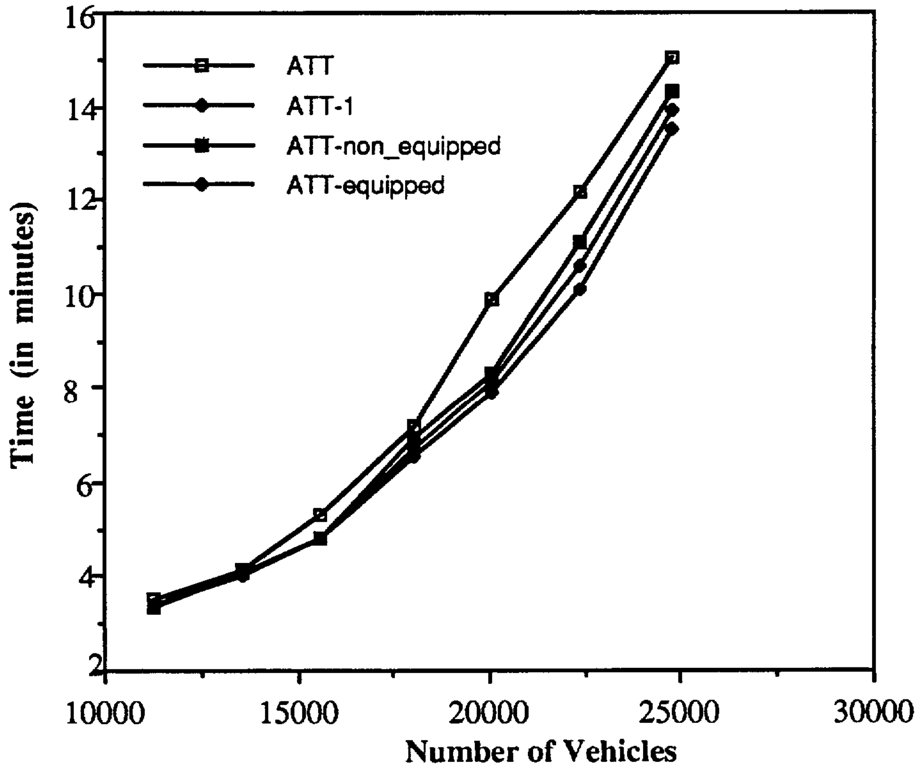


Figure 10. Variation of Average Trip Time for No Information Base Case and Myopic Switching Case

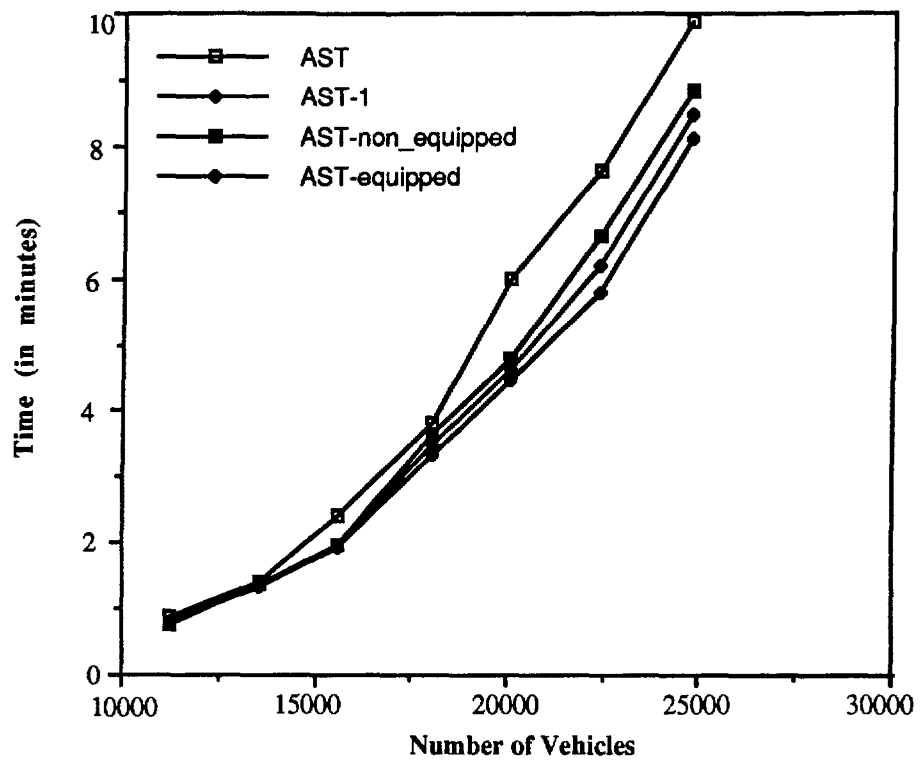


Figure 11. Variation of Average Stopped Time for Base Case and Myopic Case

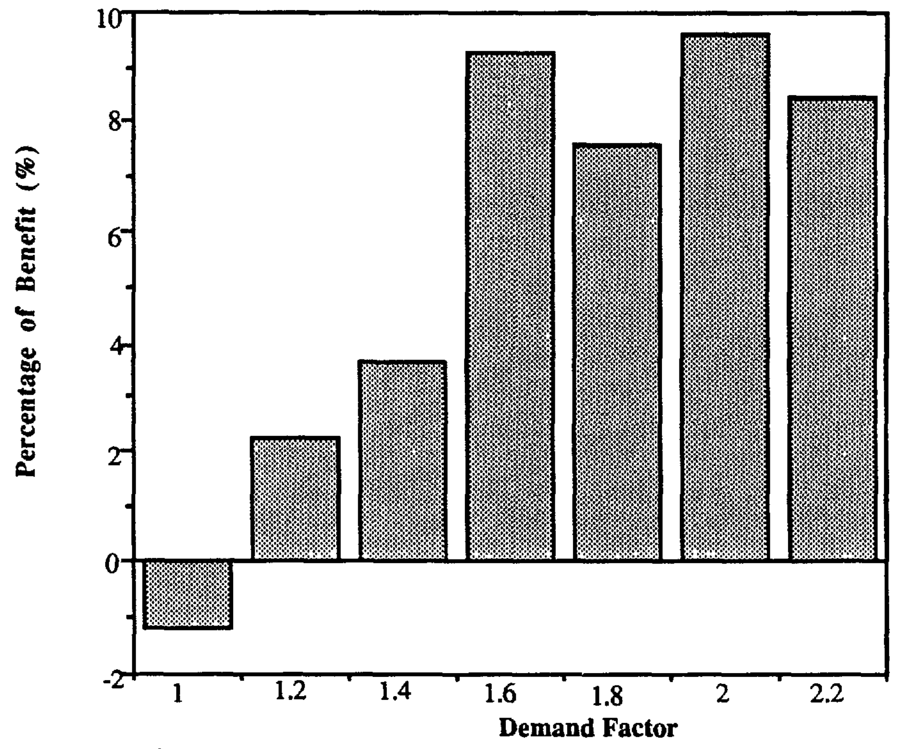


Figure 12. Relative Benefit (in terms of travel time savings) of Equipped Vehicles over Non-Equipped Vehicles under 0.2-Band Switching Rule

Table 7. Comparison of Route Switching under Myopic Rule Case and 0.2-Band Switching Rule

Demand Factor	Indiff Band	Number of Switches 0	1	2	3	4	5--8	Number of Vehicles	% of switches
1	0	130	737	1173	1241	866	721	4868	97.33
	0.2	4295	537	35	0	1		4868	11.77
2	0	205	967	1735	1532	981	612	6032	96.60
	0.2	4375	1351	259	44	3		6032	27.47
3	0	236	1239	2039	1723	1112	562	6911	96.59
	0.2	3629	2277	791	178	27	9	69il	47.49
4	0	408	1846	2401	1805	972	438	7870	94.82
	0.2	2968	3057	1386	375	72	12	7870	62.29
5	0	546	2376	2819	1921	909	315	8886	93.86
	0.2	2864	3561	1779	507	154	21	8886	67.77
6	0	444	2218	3021	2278	1226	657	9844	95.49
	0.2	2668	4108	2191	721	138	18	9844	72.90
7	0	1244	3887	3382	1616	565	231	10925	88.61
	0.2	3533	4428	2310	551	91	12	10925	67.66

Table 8. System Performance for Multiple User Classes

Demand		Ave. Trip Time		Ave. Stop Time		Ave. Trip Dis.	
Factor		(in minutes)	%	(in minutes)	%	(in miles)	%
1	AVERAGE	3.8		1.23		1.326	
	CLASS-1	3.66	96.32	1.08	87.80	1.327	100.08
	CLASS-2	4.62	121.58	2	162.60	1.339	100.98
	CLASS-4	3.62	95.26	1.05	85.37	1.32	99.55
	CLASS-5	4.3	113.16	1.73	140.65	1.329	100.23
2	AVERAGE	4.53		1.81		1.331	
	CLASS-1	4.45	98.23	1.73	95.58	1.328	99.77
	CLASS-2	5.22	115.23	2.48	137.02	1.338	100.53
	CLASS-4	4.32	95.36	1.6	88.40	1.338	100.53
	CLASS-5	5	110.38	2.32	128.18	1.305	98.05
3	AVERAGE	6.07		3.07		1.357	
	CLASS-1	5.96	98.19	2.94	95.77	1.358	100.07
	CLASS-2	6.92	114.00	3.94	128.34	1.341	98.82
	CLASS-4	5.8	95.55	2.8	91.21	1.358	100.07
	CLASS-5	6.79	111.86	3.77	122.80	1.362	100.37
4	AVERAGE	8.29		4.84		1.384	
	CLASS-1	8.25	99.52	4.76	98.35	1.382	99.86
	CLASS-2	9.9	119.42	6.4	132.23	1.38	99.71
	CLASS-4	7.76	93.61	4.34	89.67	1.384	100.00
	CLASS-5	8.89	107.24	5.5	113.64	1.397	100.94
5	AVERAGE	10.18		6.35		1.383	
	CLASS-1	10.24	100.59	6.31	99.37	1.384	100.07
	CLASS-2	12.32	121.02	8.28	130.39	1.407	101.74
	CLASS-4	9.34	91.75	5.66	89.13	1.377	99.57
	CLASS-5	11.06	108.64	7.35	115.75	1.377	99.57
6	AVERAGE	14.89		10.12		1.429	
	CLASS-1	15.03	100.94	10.11	99.90	1.421	99.44
	CLASS-2	16.87	113.30	12.18	120.36	1.39	97.27
	CLASS-4	13.92	93.49	9.22	91.11	1.442	100.91
	CLASS-5	16.28	109.34	11.67	115.32	1.445	101.12
7	AVERAGE	23.06		17.08	.	1.502	
	CLASS-1	23.8	103.21	17.14	100.35	1.473	98.07
	CLASS-2	28.42	123.24	21.36	125.06	1.509	100.47
	CLASS-4	21.84	94.71	15.44	90.40	1.511	100.60
	CLASS-5	25.79	111.84	19.18	112.30	1.573	104.73

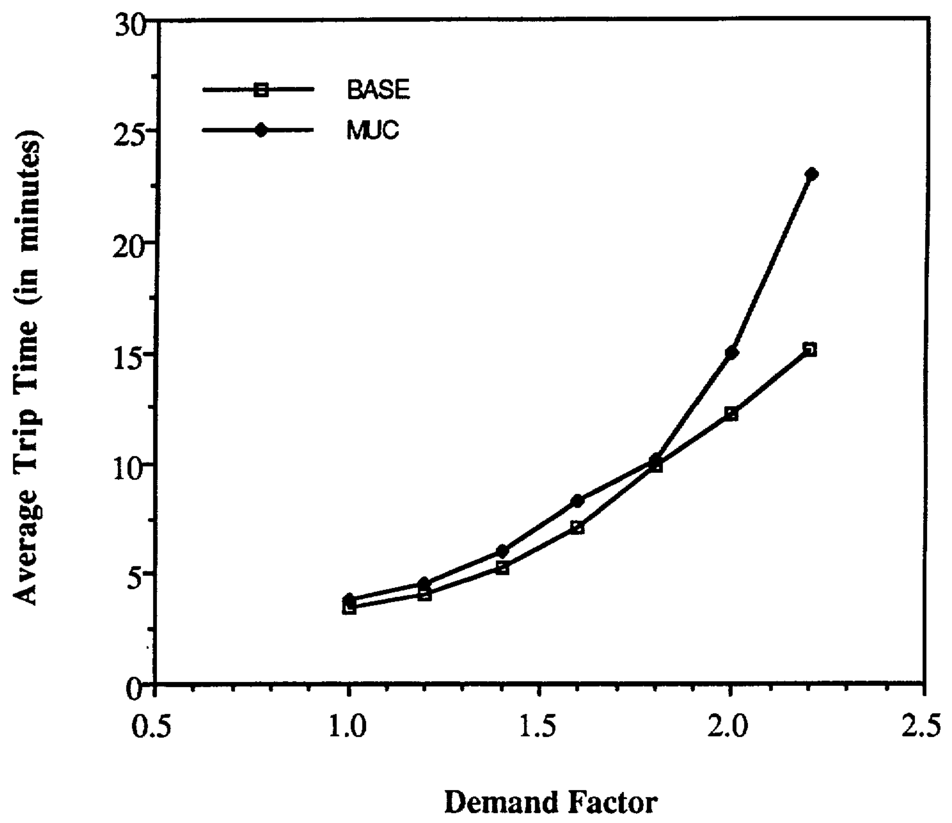


Figure 13. Comparisons of ATT for MUC Case and Base Case (50% market penetration)

Bus Operations

Another special feature of DYNASMART is to simulate buses in a network; however, this ability can be extended to any other vehicle with a fixed departure time and a predefined path. The input information was described in the previous chapter's section on bus operations. A simple case with 18 buses generated within a 30-minute period, over three predefined paths is tested. The average dwell time of each stop is 60 seconds, and all stops are assumed at the near side of blocks. These buses are operating under the case with a demand factor of 2.2. The average trip time is 16.16 minutes compared with 15.03 minutes for the no-bus case reflecting the increase in travel time due to blockage associated with bus operations. A typical bus trajectory is given in numerical form as follows :

```
Veh(# tag inf 0 D ST TT):23154   2  0   1  25   25.00   20.52
node-sequence     2     a    14     20     26     25
cumulative TT  3.30  13.90  15.90  17.40  19.10  20.52
Travel Time    3.30  10.60   2.00   1.50   1.70   1.42
Stop Time      3.30   8.72   1.24   1.08   1.28   1.00

Veh(# tag inf 0 D ST AT TT):23169  2  0  18  25   25.00   32.92
  24    30    29    34    33    32    31    25
  5.10   6.52  24.70  26.40  27.72  29.50  31.50  32.92
  5.10   1.42  18.18  1.70   1.32  1.78   2.00   1.42
  5.10   1.00  17.78  1.17   1.00  1.38   1.58   1.00

Veh(# tag inf 0 D ST AT TT): 23173  2  0  26  2  25.00  24.12
  31    25    19    13    7    1    2
  9.00  12.30  la.50  20.00  21.32  22.72  24.12
  9.00   3.30   6.20   1.50   1.32   1.40   1.40
  8.90   2.47   4.76   1.08   1.00   1.00   1.00
```

COMPUTATIONAL RESULTS

This section discusses the computational performance of DYNASMART, which is an important element in view of its eventual use in a real-time or quasi real-time environment.

Performance Measurement

Several techniques have been applied and tested to analyze and improve the program code. The performance utilities on CRAY are adapted to generate more detailed information to identify the most time consuming components of the program. Two major utilities are used to detect the most time consuming parts : FLOWTRACE and PROF. FLOWTRACE measures the execution time of subroutines, and PROF is able to generate information about the time spent in each loop and in each statement.

Computational Results for the Small Network

Execution Analysis of Subroutines

The code execution analysis for the small network is discussed in this section. Table 9 provides representative results on the computational intensity of the program routines as found in a 74.5 minute simulation with 20,064 vehicles. There are 10 destination nodes in this network, and the multiple-user classes K-shortest path subroutine is used. The time interval for calculating K-shortest paths is two minutes (i.e. 20 simulation time steps), and the path file is updated for every simulation time step (6 seconds). The execution time analysis at the subroutine level generated from FLOWTRACE includes the total time (in CPU seconds), number of calls, average time for each call, percentage of total, the accumulative percentage. In this case, with autotasking and some microtasking directives invoked the total execution time is about 122 seconds. All the subroutines are explained as follows:

COMBINEDLABEL : combine and update multiple path labels for each user class
PARTCO : the main traffic simulation which performs link movement and node transfer
GETLINK : behavioral component, which provide the next link on the path
KSHORTESTPATH : K-shortest path bound calculation
PENCAL : movement penalty calculation
MAIN : the main control program which include input and output statement
BUILDPRIORITIES : build priority for COMBINEDLABEL
ADJUSTAT : adjust the saturation flow rate according to the left-turn ratio
KSHORTESTPATHC : K-shortest path calculation
LABELSUPDATE : update the path label after the KSHORTESTPATHC
INTEGRATEIT : integrate the path to an unified structure
LEFTVAL : evaluate the left-turn capacity according to the current information
PRETIME : calculate the signal cycle for pretimed control intersections
INITIALIZEAR : initialize the array bound before KSHORTESPTAH calculation
SIGFUN : the fork component for signal control calculation
BEGINRT : the initial path assignment
ACTUATED : actuated signal control calculation
OUTPUT : output system performance for multiple user classes
NOCONTROL : process intersection control without signs and signals
INITIALIZAARRAYS : the initialization of KSP arrays
LEFTCON : the interface for calling KSP calculation.
INCIREAD : read incident data
RAMPFUN : process ramp functions
INITIALIZEPRMTS : the initial set up for KSP Calculation.

Table 9. Execution Output from FLOWTRACE

Route Name	TOT TIME (in seconds)	# Calls	Avg. Time (in seconds)	Percentage	Accum %
COMBINEDLABEL	5.35E+01	19020	2.81E-03	44.01	44.01
PARTCO	2.35E+01	644	3.65E-02	19.33	63.34
GETLINK	1.10E+01	1112605	9.87E-06	9.04	72.37
KSHORTESTPATH	9.95E+00	660	1.51E-02	8.18	80.56
PENCAL	9.86E+00	645	1.53E-02	8.11	88.67
MAIN	2.79E+00	1	2.79E+00	2.3	90.97
BUILDPRIORITIES	2.66E+00	990	2.69E-03	2.19	93.15
ADJUSTSAT	2.42E+00	108360	2.24E-05	1.99	95.15
KSHORTESTPATHC	2.38E+00	330	7.20E-03	1.96	97.1
LABELSUPDATE	1.37E+00	660	2.07E-03	1.12	98.23
INTEGRATEIT	5.81E-01	330	1.76E-03	0.48	98.7
LEFTVAL	5.72E-01	66435	8.61E-06	0.47	99.18
PRETIME	3.53E-01	16770	2.11E-05	0.29	99.47
INITIALIZEAR	2.57E-01	660	3.90E-04	0.21	99.68
SIGFUN	9.86E-02	645	1.53E-04	0.08	99.76
BEGINRT	9.46E-02	20141	4.70E-06	0.08	99.84
ACTUATED	7.02E-02	5160	1.36E-05	0.06	99.89
OUTPUT	4.62E-02	1	4.62E-02	0.04	99.93
NOCONTROL	3.61E-02	10320	3.49E-06	0.03	99.96
INITIALIZEARRAYS	3.48E-02	330	1.05E-04	0.03	99.99
LEFTCON	1.15E-02	33	3.48E-04	0.01	100
INCIREAD	2.55E-04	1	2.55E-04	0	100
RAMPFUN	1.96E-04	64	3.06E-06	0	100
INITIALIZEPRMTRS	1.52E-06	1	1.52E-06	0	100

Table 10. Comparison of Execution of DYNASMART under Different Optimization Options

Route Name	BASE (in seconds)	CASE A	Speed-Up of BASE	CASE B	Speed-Up of BASE
COMBINEDLABEL	5.35E+01	1.98E+02	3.70	6.23E+01	1.16
PARTCO	2.35E+01	2.41E+02	10.26	7.93E+01	3.37
GETLINK	1.10E+01	1.58E+01	1.44	1.10E+01	1.00
KSHORTESTPATH	9.95E+00	2.54E+01	2.55	1.01E+01	1.02
PENCAL	9.86E+00	4.35E+01	4.41	1.05E+01	1.06
MAIN	2.79E+00	5.91E+00	2.12	2.77E+00	0.99
BUILDPRIORITIES	2.66E+00	8.42E+00	3.17	2.08E+00	0.78
ADJUSTSAT	2.42E+00	7.10E+00	2.93	2.18E+00	0.90
KSHORTESTPATHC	2.38E+00	7.04E+00	2.96	2.38E+00	1.00
LABELSUPDATE	1.37E+00	4.50E+00	3.28	1.45E+00	1.06
INTEGRATEIT	5.81E-01	3.40E+00	5.85	6.36E-01	1.09
LEFTVAL	5.72E-01	9.74E-01	1.70	5.92E-01	1.03
PRETIME	3.53E-01	1.04E+00	2.95	3.68E-01	1.04
INITIALIZEAR	2.57E-01	1.38E+00	5.37	4.04E-01	1.57
SIGFUN	9.86E-02	1.46E-01	1.48	9.83E-02	1.00
BEGINRT	9.46E-02	2.27E-01	2.40	9.38E-02	0.99
ACTUATED	7.02E-02	2.83E-01	4.03	9.00E-02	1.28
OUTPUT	4.62E-02	9.56E-02	2.07	3.89E-02	0.84
NOCONTROL	3.61E-02	8.26E-02	2.29	3.50E-02	0.97
INITIALIZEARRAYS	3.48E-02	1.38E+00	39.66	7.48E-02	2.15
LEFTCON	1.15E-02	1.22E-02	1.06	1.13E-02	0.98
INCIREAD	2.55E-04	3.05E-04	1.20	2.58E-04	1.01
RAMPFUN	1.96E-04	5.70E-03	29.08	3.76E-04	1.92
INITIALIZEPRMTRS	1.52E-06	1.57E-06	1.03	1.56E-06	1.03
TOTAL	121.59	565.70	4.65	186.50	1.53

In Table 9, we see that about 57 percent of the execution time is spent on multiple user classes path calculations for the 10 destinations network. PARTCO, the main simulator, is about 20 percent, and PENCAL, the interface between KSHORTEST and PARTCO take about 8 percent of execution time. For path processing, the update subroutine is three times faster than recalculating the paths. For the behavioral rules incorporated in GETLINK, the number of calls is 1,112,605 and takes 9% of total execution time. When the vehicles reach nodes, GETLINK will be called to determine the next link on the path. MAIN is the major input and output module, and takes about 2.79 seconds, or about 2.3 percent of the total. Note here that the output information obtained from these tests includes only the system performance statistics, and all the time-dependent information are not reported. All other subroutines jointly account for less than 10% of execution time.

Different speedup comparisons are reported in Table 10, in which two cases are compared with the previous time calculation:

CASE A : all the optimization features of the compiler are turned off (such as autotasking, vectorization)

CASE B : without autotasking and aggressive optimization setting in compiler

The total execution time for CASE A is 565.7 CPU seconds which is 4.6 times slower than the optimal one, and CASE B is 1.5 times slower than the optimized version. The speed up ratio of the optimized version is also reported in column 4 and column 6. In column 6, we can see the speed-up of optimized program is quite significant. All the subroutines obtain l-5 times speed up, while PARTCO experiences about 10 times speed up. In CASE B, AUTOTASKING and AGGRESS setting have been turned off. Through the autotasking capability of CRAY, we can obtain about 1.5 speed-up for DYNASMART.

The fact that a majority of the computation time is incurred in the path processing component is encouraging, because the present experiments included very extensive path computations that most certainly exceed practical requirements. For instance, instead of updating all K paths every simulation interval, the update may take place only every other interval, thereby halving the computation associated with this task. More important, the time frame for recomputing the K paths need not be as short as it is here: going from 2 to 4 minutes would mean a 50% reduction in execution time associated with that portion.

Comparison for Different Demand Factors

Figure 14 illustrates the variation of execution time for different demand levels. When the congestion level is increased, the number of vehicles and simulation periods are also increased as reflected in the execution time. Recall that the network is simulated until all loaded vehicles clear the network; as more vehicles are generated over the same generation duration, congestion in the network increases and it takes longer to clear all vehicles. For this reason, the total simulation period is longer at higher demand and so is the execution time. The time statistics, which include two (large) time-dependent output information files, are shown in Figure 14. These execution times are not directly comparable to those in the previous section, because of the additional output. For instance, the execution time for the same factor, 1.8, is now 157 which is 35 seconds more than the previous analysis, indicating that the I/O (input-output) part can be very time-consuming, depending on how much detailed information is requested by the user.

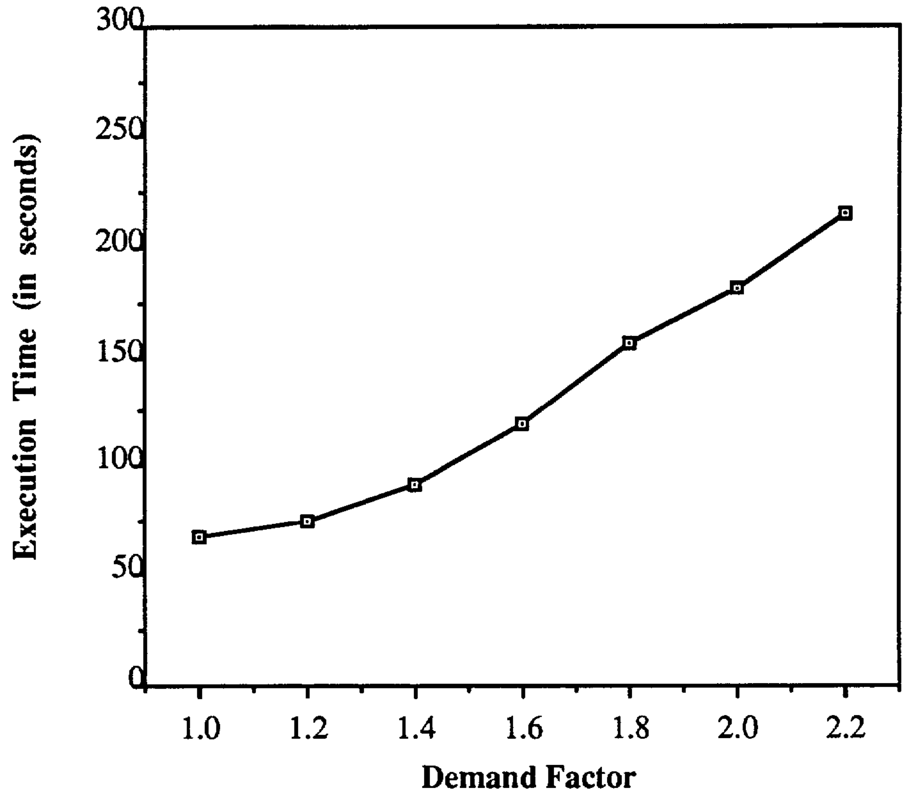


Figure 14. Execution Time for Different Demand Factors

Single User Class DYNASMART Execution Analysis

In some applications involving only one class of users, it would be more efficient computationally to execute a special version of DYNASMART with streamlined data structures intended for a single class of users.

Some other specifications :
Number of Nodes : 50
Number of Links : 168
Destination Zones : 38
Destination Nodes : 38
Simulation Interval : 0.1 minutes
Total Number of vehicles : 15007
Simulation Time : 44.7 minutes
Average Travel Time : 5.05 minutes

The execution time is shown in Table 11. The single K-shortest path subroutine takes 50 percent of execution time, and PARTCO takes about 30 percent of execution time. It is also expected that the execution time will increase with the increase in the number of vehicles and associated congestion levels.

DYNASMART is written in CRAY FORTRAN (CFT), which is readily portable to other environments (the code has been successfully executed on workstations). Being written in CF177, DYNASMART can be executed on various other hosts with little modification. The execution time for running on CRAY YMP, CONVEX, and RISC 6000 are reported in Table 12. Note that the execution time is obtained with only the overall system performance measures output. CONVEX and RISC 6000 are two front end machines of the CRAY in the University of Texas System Center for High Performance Computing (CHPC). Surprisingly, the performance of RISC 6000 is two times faster than CONVEX machine. In these runs, the execution time on CRAY is 2-7 times faster than on RISC-6000, with CRAY providing much better relative performance as the number of vehicles increases.

Execution Analysis for the Austin Network

Although the previous test network is relatively small, the total execution time described in the previous section provides some idea about the time spent on each subroutine. Some of the experiments for the Austin core network (676 nodes, 1882 links, 32 destinations and 36 demand zones) without movement considerations, are reported in Table 13. These tests are made without explicit left-turning movement penalties. There are 479 stop/yield signs and 132 signalized intersections. For a total of 17,712 vehicles, DYNASMART executed in 621.1 seconds for an 80-minute simulation period.

Table 11. Execution Analysis for Single User Class Version of DYNASMART

Routine Name	Tot Time	# Calls	Avg. Time	Percentage
KSHORT	2.69E+01	874	3.07E-02	48.03
PARTCO	1.74E+01	446	3.89E-02	31.04
MAIN	7.04E+00	1	7.04E+00	12.58
PENALTY	1.22E+00	446	2.73E-03	2.18
INTEGRAT	1.22E+00	874	1.39E-03	2.17
ADJUSTS	6.24E-01	75096	8.31E-06	1.12
GETLINK	5.79E-01	173362	3.34E-06	1.04
LEFTVAL	3.55E-01	42465	8.36E-06	0.63
INITIALA	1.72E-01	874	1.97E-04	0.31
PRETIME	1.37E-01	11622	1.18E-05	0.24
READLEFT	9.33E-02	1	9.33E-02	0.17
BEGINRT	8.17E-02	15007	5.44E-06	0.15
SIGFUN	6.77E-02	447	1.51E-04	0.12
ACTUATED	4.70E-02	3576	1.32E-05	0.08
RAMPFUN	4.18E-02	44	9.50E-04	0.07
NOCONTROL	2.81E-02	7152	3.93E-06	0.05
LEFTCON	6.14E-03	23	2.67E-04	0.01
TITLE	3.90E-03	1	3.90E-03	0.01
Totals	5.60E+01			

Table 12. Performance Analysis on Different Hosts for Single User Class Version of DY NAS MART

Number of vehicles	Simulation Time (in minutes)	Average Travel Time (in minutes)	CRAY (in seconds)	CONVEX (in seconds)	RISC6000 (in seconds)
15007	44.7	5.05	49	242.3	119.6
22507	67.9	9.12	65.3	438.9	244.8
30029	107.1	22.9	95.3	1029.9	703.5

Table 13. Execution Analysis for the Austin Network without Movement Considerations

Number of Vehicles	Simulation Time (in minutes)	Average Travel Time (in minutes)	EXEC Time (in CPU seconds)
12431	45.2	4.77	462.4
15046	62.3	6.38	563.3
17712	81.4	7.13	621.1

Table 14 provides the representative results on the computational performance of the single user class DYNASMART with explicit left-turn representation in an 87-minute simulation of the Austin network. The program is compiled with autotasking, microtasking, aggressive, and inline insertion. The total execution time is 724 CPU seconds on CRAY YMP. The subroutines are briefly described as follows:

LEFTCON	left-turn K-SP calculation
LABELSUPDATE	path update
PARTCOMU	the core of traffic simulation
BUILDPRIORITIES	build priorities for LABELSUPDATE
PENCALSN	penalty calculation
MAINSUC	main program
INTEGRATEIT	integrate the path into a single structure
GETLINK	path selection component
ADJUSTSAT	saturation flow rate adjustment
INITIALIZEARR	initialize arrays for KSP calculation
LEFTVAL	check the left-turn capacity from the built-in tables
BEGINRTS	assign the initial paths to vehicles
PENCALMU	initial phase of penalty calculation

In this analysis, ten best paths are calculated every two minutes (20 simulation time steps), and these paths are updated every 0.1 minute (i.e. every simulation time step). We can see that 70% of execution is spent on LEFTCON, LABELSUPDATE and BUILDPRIORITIES, as compared to about 11% in vehicle simulation. Again, this execution time can be readily reduced by less frequent path calculations with limited loss of accuracy. In particular, execution runs in the order of 300 seconds were obtained for the same simulation case with path recalculation every 40 time steps and path updates every four time steps with very comparable overall system performance. Most important from the standpoint of real-time system optimal (or user equilibrium or multi user class) dynamic assignment algorithms, where DYNASMART is used as a simulator, is that execution time for comparable runs to that above is about 180 CPU seconds (Austin network) when the vehicle paths are pre-specified, as they are in the dynamic assignment algorithms presented in the next two chapters.

Table 14. Execution Analysis for the Austin Network

Routine Name	Tot Time	# calls	Avg. Time	Percentage
LEFTCON	2.84E+02	3.90E+01	7.28E+00	39.25
LABELSUPDATE	1.46E+02	4.93E+03	2.97E-02	20.23
PARTCOMU	7.64E+01	7.70E+02	9.92E-02	10.55
BUILDPRIORITIES	6.94E+01	1.25E+03	5.56E-02	9.59
PENCALSN	6.08E+01	7.70E+02	7.90E-02	8.41
MAINSUC	4.80E+01	1.00E+00	4.80E+01	6.63
INTEGRATEIT	2.13E+01	1.25E+03	1.70E-02	2.94
GETLINK	5.05E+00	4.80E+05	1.05E-05	0.7
ADJUSTSAT	4.5 lE+00	1.46E+06	3.09E-06	0.62
SIGFUN	4.06E+00	7.71E+02	5.27E-03	0.56
INlTIALIZEARR	2.63E+00	1.25E+03	2.1lE-03	0.36
LEFTVAL	9.03E-0 1	1.19E+05	7.60E-06	0.12
BEGINRTS	1.44E-0 1	7.95E+03	1.81E-05	0.02
PENCALMU	8.38E-02	1 .00E+00	8.38E-02	0.01
Totals	7.24E+02	2 0 7 5 5 5 0		

APPLICATIONS AND EXTENSIONS

DYNASMART provides a flexible framework to analyze traffic network performance under real-time information, traffic control actions and user behavior strategies. In the form described in this and the previous chapter, it simulates network conditions over a given (peak) period on a given day. DYNASMART is being extended along three important dimensions: (1) day to day system dynamics and evolution, (2) real-time adaptive traffic control; and (3) responsiveness to road pricing options.

The first dimension is an essential one in the analysis of the effect of information, as it considers the changes in departure time from day to day. Consideration of day to day decisions of tripmakers (departure time and route) allows a more complete evaluation of the evolution of a particular system under a particular information supply or traffic management strategy. This capability requires the specification of appropriate decision rules in the user behavior component.

The second capability is to a large extent already available, though specific control modules need to be incorporated. More important, explicit consideration of control actions in an algorithmic procedure jointly with routing choice is of primary importance to ATMS applications.

With regard to road pricing, the structure of DYNASMART already provides the flexibility to incorporate user response rules to congestion pricing schemes. Of course, developing such behavioral rules requires an observational basis, presently being pursued under a separate study.

As noted in the first chapter, one of the primary uses of DYNASMART is as a simulator in the context of algorithmic procedures to solve for a set of paths followed by drivers in order to achieve either a system optimum or user equilibrium in a given network with time dependent demand. These algorithms and their implementation are described in the next two chapters.

CHAPTER 4

THE SYSTEM OPTIMAL DYNAMIC TRAFFIC ASSIGNMENT PROBLEM

Chapter 4 introduces and details the system optimal dynamic traffic assignment problem in the context of ATIS/ATMS applications. This chapter first presents a brief introduction to the problem being addressed and then discusses the dynamic assignment capabilities envisaged for the ATIS/ATMS context. The existing literature in the area is then briefly reviewed and the issues that influence the formulation of the problem are discussed. The body of the chapter presents formulations for the problem based on different information availability scenarios for the controller, and describes a simulation-based solution algorithm developed for the single user class system optimal dynamic traffic assignment problem: the modifications necessary to obtain the user equilibrium solution are also stated. The remainder of the chapter reports and analyzes results from experiments designed to evaluate alternative information supply strategies in the context of ATIS operations, followed by concluding comments.

INTRODUCTION AND PROBLEM DEFINITION

Approaches incorporating advances in communication technologies, information processing systems, electronics and automation, broadly labeled as Intelligent Vehicle Highway Systems (IVHS), continue to generate considerable interest for their potential to alleviate urban and suburban traffic system congestion. Advanced Traveler Information Systems (ATIS) provide travelers with real-time information on existing traffic conditions and/or route selection recommendations from their current location to their destinations. Successful implementation of ATIS, especially at high market penetration levels, involves the dynamic assignment of vehicles to “optimal” paths to reduce overall system user costs.

The system optimal dynamic traffic assignment problem is directly relevant to the normative assignment problem encountered in connection with ATIS/ATMS operations. It addresses the problem where a central controller with known or predicted time-dependent origin-destination (O-D) trip desires over the horizon of interest solves for paths to provide users in order to attain some system-wide objectives. A system optimal assignment does not generally represent an equilibrium flow pattern because some users may be able to obtain (possibly very slight) individual travel cost savings by unilaterally changing routes. Its significance to the ATIS context lies in providing a benchmark against which other assignments or information supply strategies can be gauged, thereby yielding an upper bound on the benefits attainable with real-time traffic information. In addition, it provides a solution basis from which to obtain actual route assignments (recommendations) to drivers, after the controller has applied certain reasonableness constraints to individual routings (e.g. with regard to circuity or excessive trip time).

A number of factors influence system performance in the context of dynamic traffic assignment for real-world ATIS/ATMS systems. The fraction of users with capability for one-way or two-way communication (market penetration) with a central controller, the various information supply strategies or assignment rules, and the user response behavior to supplied information are critical determinants of the particular dynamic assignment strategy in any realistic scenario for implementing ATIS/ATMS. An ideal scenario from a controller's perspective in the ATIS context is one where all users of the system have full access to information, are provided route guidance instructions based on a system optimal strategy, and comply fully with the supplied information, thereby extracting the best possible performance from the system. This chapter defines the system optimal dynamic traffic assignment problem and discusses the formulations and solution algorithm developed for the single user class case. The associated user equilibrium assignment problem is also studied so as to derive insights into the performance of prescriptive versus descriptive strategies in the context of dynamic traffic assignment under in-vehicle information systems. The next chapter addresses, among others, the multiple user class dynamic traffic assignment problem (which includes both users that follow UE paths and users that follow SO paths) by extending the formulations and solution methodology for the single user class problem.

DYNAMIC TRAFFIC ASSIGNMENT CAPABILITIES FOR ATIS/ATMS

As discussed in Chapter 1, two capabilities are envisaged for dynamic traffic assignment in the context of ATIS/ATMS operations. A normative perspective determines a solution that seeks to achieve some overall objectives for the system. A descriptive perspective seeks to describe the traffic conditions that will occur in a network under a particular loading pattern. The former usually requires a capability for the latter. User equilibrium and system optimal assignments are associated with a set of conditions that must be satisfied by the routes followed by users in a network. An extensive and comprehensive discussion of formulations for the SO and UE static assignment problems is provided in Sheffi (1985). In a system optimal strategy, the aim is to solve for routes that “optimize” the overall system performance, subject to reasonableness and fairness conditions for individual users. Hence, it corresponds to a normative perspective. A descriptive simulation-assignment capability is necessary to perform a normative assignment and test alternative information and routing strategies.

The User Equilibrium Case

The time-dependent UE formulations are not of immediate relevance to the real-time assignment needs in the ATIS/ATMS context, where the controller’s objective is to optimize some system-wide performance measures. The pertinence of UE to the problem lies in its historical significance to the classical static assignment problem, where equilibrium analyses are performed for long-term planning applications. Under UE, which is claimed to represent a reasonable construct for user behavior, every user is assumed to try to minimize his/her own travel cost when traveling from origin to destination. A Wardrop UE holds when no user can improve his/her individual cost by unilateral route switching, and thus represents an equilibrium condition. There is no empirical evidence that UE conditions actually hold in real networks, though the UE solution is considered a reasonable and useful construct for the evaluation of long-term capacity improvements.

Under real-time descriptive ATIS information on network conditions, a time-dependent UE pattern could be viewed as the result of the long-term evolution of the system, as users somehow learn and adjust under the supplied information. However, it is not at all clear that such convergence would be attained under inherently dynamic conditions (exacerbated by supplying information to users). Thus it is not known what the UE solution may represent from the standpoint of ATIS operation and evaluation, Actual user behavior and system performance under real-time descriptive information may be better or worse than the corresponding time-dependent UE solution in terms of the overall system cost. Nevertheless, a time-dependent UE pattern may be considered as a useful proxy for a favorable scenario of long-term network performance under real-time descriptive information.

The System Optimal Case

A SO solution, by definition, is the best one could achieve in terms of an overall measure of performance. In the context of traffic assignment, it can be attained by routing vehicles on the least marginal cost paths to their destinations, paths which impose the least penalty on the system due to vehicles traveling on them. A system optimal assignment does not generally represent an equilibrium flow pattern, or a model of actual user behavior, because some users may be able to obtain individual advantages simply by changing routes, though imposing a greater marginal cost to other users in the system in the process. Its significance to the ATIS context lies in the value of the SO objective function serving as a yardstick by which other possible problem formulations as well as simple-to-implement heuristic control schemes can be evaluated, thereby yielding an upper bound on the benefits attainable with real-time traffic information.

In the context of real-time assignment, a controller seeking to optimize overall system performance is constrained among others by individual considerations of reasonableness, fairness, equity, reliability and credibility. A SO solution may not necessarily be equitable, in that some users may be guided on to longer routes in order to reduce the travel time for other tripmakers. Consequently, the SO solution provides only a starting point for actually providing route guidance information. A slightly modified strategy envisions an additional level of processing by which users are assigned paths within a certain threshold of the best path. Previous observational work conducted in the Austin network (Mahmassani et al, 1990) has indicated the possibility of an abundance of “good” paths between an O-D pair. This ensures almost everyone good paths, though not necessarily the shortest path. In addition, to the extent that one’s route assignment may be randomized over drivers, all drivers will ultimately (i.e. in the long run) be better off, *on average.*

BACKGROUND REVIEW

Dynamic network assignment is under active development, for both the user equilibrium and SO problems. Existing formulations are not entirely satisfactory in terms of the underlying assumptions and/or cannot be solved for realistic networks.

The bulk of the contributions to the system optimal dynamic assignment problem have addressed the situation where known time-dependent flows are assigned from multiple origins to a *single* destination through the links of a network so as to minimize total system cost. The first mathematical programming approach to this problem is due to Merchant and Nemhauser (1978). Their model was formulated as a discrete-time, non-linear, non-convex mathematical program and the corresponding algorithm solved a piece wise linear version of it. Congestion was treated explicitly using conventional link performance functions.

Carey (1986) reformulated the Merchant-Nemhauser problem as a well-behaved convex nonlinear program, which offered mathematical and algorithmic advantages over the original formulation. Extensions to multiple destinations or multiple commodities remain problematic because of non-convexity issues. Multiple destinations require the models to explicitly seek to satisfy a “first-in, first-out“ requirement that is essential from a traffic viewpoint. This requirement introduces additional constraints that complicate the formulation, and destroy many of its nice properties, as they generally yield a non-convex constraint set (Carey, 1992).

A more recent line of work has considered constrained optimal control theory. The O-D trip rates are assumed to be known continuous functions of time, and the link flows are sought as continuous functions of time. Friesz et al. (1989) discuss optimal control formulations for both system optimal and user equilibrium problems. They propose a dynamic generalization of Beckmann’s equivalent optimization problem for static user optimized traffic assignment in the form of an optimal control problem. Ran and Shimazalci (1989) used the optimal control approach to develop a general model of dynamic system optimal traffic assignment for an urban transportation network with many origins and many destinations. Ran and Boyce (1993) formulated a continuous dynamic user optimal traffic assignment model in which exit flows are treated as a set of control variables rather than as functions, so as to overcome difficulties posed by the non-linearity of the exit flow function for multiple origin-destination networks. Wie (1990) extended the model by Friesz et al. (1989) to include elastic time-varying travel demand, which leads to the implicit consideration of departure time choices. Wie also enumerates several limitations of this approach.

Boyce et al. (1991) used the optimal control theory approach to obtain a convex model for dynamic user equilibrium assignment by defining inflows and exit flows on links to be control variables. They discussed a methodology to solve the discretized version of the problem using the Frank-Wolfe algorithm and an expanded time-space network representation. However, the use of static link performance functions is a limitation of this model as such functions do not adequately model the dynamics of congested traffic behavior. Furthermore, the authors have not reported any implementation of this approach even on a test network. Both Ran, Boyce and co-workers on one hand, and Friesz, Wie and co-workers on the other, have continued to develop the mathematical theory underlying different UE formulations and interpretations. While these continue to advance the scientific body of work, they have not resulted in practical solution algorithms for general networks and have several essential traffic modeling issues that remain to be solved satisfactorily.

Another direction of work with feedback regulation was introduced by Papageorgiou et al. (1990). A multivariable feedback regulator with integral parts and a simple bang-bang controller was developed and tested for a particular network traffic model. However, the formulation does not establish the underlying mathematical basis with regard to the solution properties and lacks a first-in, first-out requirement.

Ghali and Smith (1991) proposed a formulation for the system optimal dynamic traffic assignment problem for multiple origin-destination demands in which congestion arises exclusively at specified bottlenecks modeled as deterministic queues. A solution procedure is proposed by analogy with the static SO problem, using marginal link costs. Although the approach does not ensure system optimality, and has limitations due to certain assumptions on queuing, it addresses several of the troublesome traffic modeling issues which seriously limit the realism and validity of previous formulations. Smith (1991) proposed a dynamic user equilibrium model for peak period traffic flows on congested capacity-constrained urban road networks. Motivated by the first-in first-out property for traffic, the model specifies a “no overtaking” condition and determines the relative priorities of vehicles at each node seeking to proceed along the various paths containing that node based on the past history of the vehicle (represented by a binary numbering scheme). Smith also proposed an algorithm for solving the model, although there is no proof that the algorithm converges to an equilibrium.

In summary, the state of art is fragmented along several lines of work, none of which is entirely satisfactory in terms of the realism of the underlying assumptions. Key weaknesses remain in terms of representing dynamic traffic phenomena which are of the essence in congested networks. A comprehensive review and discussion of dynamic assignment and traffic simulation models for ATIS/ATMS are given in Mahmassani et al. (1992). In the next section, the principal elements of SO traffic assignment problem formulations for ATIS/ATMS applications are identified, along with the key issues faced in their solution.

SYSTEM OPTIMAL FORMULATIONS - ISSUES FOR ATIS CONTEXT

This section identifies and discusses principal issues involved in formulating the system optimal dynamic traffic assignment problem for ATIS/ATMS applications.

Information Available to Controller

Different scenarios are possible based on the extent and type of information on O-D desires and network traffic conditions assumed to be available to the controller. If complete information is available on the origin, destination and timing of all trips for the entire duration of interest, path assignment can be made for all time intervals in the beginning. Partial information on O-D desires in the future can be modeled using a rolling horizon framework, possibly in connection with a stochastic formulation in which O-D trips are modeled as random variables. Solution of the complete information formulation is necessary to obtain a benchmark and a lower bound on system costs for other, partial information formulations.

Information Available to Travelers

Users with equipped vehicles are given information on the condition of the network and/or instructions on the path to be taken to their desired destinations. In the system optimal problem, the controller provides users with routes to their destinations. However, compliance influences the performance of the system. Under descriptive (as opposed to normative) information supply strategies (see Mahmassani and Jayakrishnan, 1991, for a discussion of such strategies), system performance depends on user decisions in response to the specific information supplied. A related (normative) problem faced by the controller in the ATIS context is the determination of the optimal information supply strategy, namely what kind of predicted trip times should be supplied to which users in order for the resulting path choices to attain certain system-wide objectives.

Evaluation of the Objective Function

The system optimal dynamic assignment problem aims at optimizing some system-wide criterion like the total system travel time. The time-dependent nature of the assignment considerably complicates the computation of the objective function. For instance, the paths followed by future O-D desires are likely to share common links with paths assigned to current trips (generally upstream), and thus influence the travel times experienced by current assignments. The path travel times experienced by vehicles are the net result of the complex spatial and temporal interactions taking place in the system over a period of time, virtually precluding the ability to evaluate the objective function analytically. Furthermore, the analytic evaluation of the objective function would entail correct analytic representation of the various dynamic traffic flow phenomena (queue formation and discharge, congestion build-up and dissipation), a task which is far from the capability of the state of the art in traffic flow modeling. For these reasons, simulation suggests itself as a plausible candidate for evaluating the objective function.

Traffic Flow Modeling

This point follows directly from the difficulty just mentioned of correctly representing the dynamics of traffic flow using the kind of analytic functions typically used in static equilibrium assignment models. Furthermore, the representation of flow as a continuum is not appropriate in the time-dependent case. Users entering the system at different times will experience different network conditions and will be assigned different paths. Continuous flow does not allow for distinction of vehicles based on whether they are equipped or not.

Path-Based and Link-Based Formulations

In virtually all existing traffic assignment models, static or dynamic, the link flows are the variables being solved for. However, for the ATIS/ATMS context, path-based assignments are called for because of the need to provide paths to the tripmakers. The problem with obtaining path flows from link flows using link-path incidence relationships is that uniqueness is not guaranteed. Furthermore, the solution of path-based formulations is likely to require partial enumeration of paths for each O-D pair, which is computationally burdensome.

Flows on arcs and paths are mathematically related through definitional identities known as the link-path incidence relationships. While relatively straightforward in the static case, link-path incidence relationships are far from trivial in the time-dependent case. In the dynamic problem, unlike flows at steady-state, vehicles assigned to a path at a given time are not simultaneously present on all links forming that particular path. Therefore, link-path incidence relationships must recognize the time at which vehicles are actually present on a link.

Holding of Traffic

In a network, it may often be advantageous, from a systemwide total delay standpoint, to favor certain traffic streams or movements over others (e.g. holding back traffic at the minor approach of an intersection in favor of the major approach). Unless otherwise specified, the solution of a SO assignment formulation may entail holding of traffic on one path in favor of traffic on other paths for some significant amount of time at points where the paths overlap or intersect. In other words, vehicles may be artificially delayed on a link for a time that exceeds what may be considered “fair” or “reasonable”. Such a solution is probably not acceptable socially nor realistic operationally. When traffic simulation is used to model traffic movements to evaluate network performance for a given assignment, unintended holding is implicitly precluded, and no additional explicit constraints are needed to take care of this problem.

First-In, First-Out Requirement

The physical behavior of traffic on a roadway link exhibits the so-called “first-in, first-out“ (FIFO) property, creating a particularly vexing difficulty in the solution of mathematical programming formulations of the network assignment problems. The FIFO requirement states that traffic that enters a road at a particular time exits from the facility, *on average,* before traffic which enters in later periods. While individual vehicles may travel at different speeds and do pass each other, FIFO should not be violated when considering travel time, averaged over a reasonable number of vehicles entering the link in a given time interval. The problem does not arise in static assignment problems (single or multiple destinations) nor in dynamic assignment with a single destination. However, in dynamic assignment problems with multiple destinations, vehicles on different paths (from different O-D pairs) who share one or more common links may be moved across this arc in a manner that violates FIFO, for instance, if the downstream arc along one path is blocked but not along the other path(s). This problem arises for both SO and UE assignment formulations. For SO problems, total travel costs could be lowered if some commodities (e.g. traffic between given O-D pair) could be temporarily held back on an arc, while allowing some other traffic types to proceed to downstream arcs. This form of holding back would violate FIFO, and is not generally physically possible, especially under congested conditions, as vehicles cannot make such “jumps” over traffic ahead of them.

FIFO is a serious liability from a mathematical programming standpoint. Carey (1992) proposed possible additional mathematical constraints to impose the FIFO rule. However, these constraints make the feasible set non-convex, destroying many of the computational and mathematical (analytic) advantages of the formulation. Carey suggests solving the problem without introducing an explicit FIFO restriction, then analyzing it for the seriousness of FIFO violations. However, no explicit procedure is proposed for this purpose. Smith (1991) proposes a DUE assignment model which approximately bypasses the FIFO issue.

The above issues further highlight the relative advantages of a simulation-assignment strategy. Simulation moves vehicles based on their current location and speed, and FIFO is implicitly satisfied.

Temporal Issues

The treatment of time in various aspects of the formulation and solution of dynamic assignment problems is an essential and subtle element of these problems. When time is discretized, the size of the time interval for assignment decisions, and its relation to the time step that may be used in the simulation of traffic movement, need to be determined.

The size of the assignment interval affects the size of the “packet” of O-D desires to be assigned jointly. If the time interval is relatively large, there will be several vehicles going from a particular origin to a particular destination in that time interval. However, their travel “experience” in the network may not be identical as vehicles at the beginning of the interval may experience different time-varying traffic conditions. Assigning all of them to the same path would be incorrect in terms of achieving the objectives. A smaller assignment time interval implies more intensive computation, giving rise to the usual trade-offs between computational intensity and accuracy.

Central and Distributed Control

The system optimal dynamic traffic assignment problem is a large scale optimization problem with large memory requirements and intensive computational needs in real-time, especially from the perspective of in-vehicle electronic navigation systems. One way of overcoming the huge computational requirements is to decentralize the control process. In the distributed control scenario, local controllers in different zones of a network assign vehicles to the network based on the local traffic conditions and information on the network traffic conditions provided by the central controller. Hence, unless there is good coordination in the transfer of information between zones, this will not lead to solutions as efficient as when a central controller makes decisions based on the entire network traffic conditions. Hence, decentralized control requires a well integrated information communication system for efficient performance. However, as stated above, decentralized control makes the real-time handling of information easier as local controllers deal with a lesser amount of data. Also, the idea of local control is especially appealing in scenarios with incidents. Since in general an incident is not global in nature, local controllers can efficiently route vehicles entering the zone with incident.

Another strategy is an integrated central-local controller scheme, where local controllers make local assignment decisions and transfer control to the central controller when decentralized control becomes inefficient as is possible under highly congested conditions. The central controller can then route vehicles for all O-D pairs and transfer control back to the local controllers when global effects are less critical. The issue of centralized versus decentralized control needs to be explored further to determine the relative advantages of local control in real-time assignment of vehicles. This issue is intrinsically related to the ATIS/ATMS control system architecture.

Link Interactions

Mathematical programming formulations generally assume that travel time on a given link depends only on flow through that link and not on the flow through any other link. This assumption fails in reality when heavy traffic occurs on two-way streets, unsignalized intersections, and left-turning movements in signalized intersections. In such cases, link interactions cannot be ignored.

Link interactions can be either symmetric or asymmetric. Symmetric interactions ascribe identical marginal effects of links flows on each of the two links to the travel time of the other. In the static case, when link interactions are asymmetric, there is no known equivalent minimization program that can be used to obtain the equilibrium flow pattern. Almost all analytical models to date on dynamic assignment problems avoid considering link interactions in their problem formulation as they lead to a much higher degree of complexity. Even in the static case, only in the past decade have approaches like variational inequality formulations for the asymmetric link interactions been discussed-Fisk and Boyce (1983). Computational aspects of this problem have been investigated by Nagumey (1984, 1986) and Mahmassani and Mouskos (1988, 1989).

This issue reemphasizes the advantage of a simulation-based approach in addressing the dynamic traffic assignment problem. The traffic flow simulator implicitly accounts for link interactions when a capability for turning movement penalties is incorporated in it, which is significant for the path processing aspects of the problem.

The above section has illustrated the difficulties involved in modeling the system-optimal assignment problem for the ATIS/ATMS context, and how a simulation-assignment strategy overcomes these problems.

FORMULATION OF THE SYSTEM OPTIMAL DYNAMIC ASSIGNMENT PROBLEM

Problem Statement

Consider a traffic network represented by a directed graph G(N, A) where N is the set of nodes and A the set of directed arcs. A node can represent a trip origin, a destination and/or a junction of physical links. We consider a network with multiple origins and destinations. The time experienced by a vehicle to traverse a given link depends on the interactions taking place among vehicles in the traffic stream along this arc. The analysis period of interest, taken here as the peak period, is discretized into small equal intervals $t = 1, \ldots\ldots\ldots, T$. Given a set of time-dependent O-D vehicle trip desires for the entire duration of the peak period, expressed as the number of vehicle trips r_{ij}^{t} leaving node i for node j in time slice t, $\forall\ i, j \in N$ and $t = 1, \ldots\ldots\ldots, T$, determine a time-dependent assignment of vehicles to network paths and corresponding arcs. In other words, find the number of vehicles r_{ijk}^{t} that follow path $k = 1, \ldots\ldots\ldots, K_{ij}$ between i and j at time t, $\forall\ i, j \in N$ and $t = 1, \ldots\ldots\ldots, T$, as well as the associated numbers of vehicles on each arc $l \in A$ over time. We consider a normative route guidance information supply strategy, whereby a controlling agent (controller) with information on the system assigns users to various routes in the network so as to satisfy some systemwide objectives. Three formulations are presented in this section, corresponding to three scenarios in terms of the extent of information available to the controller.

Information Availability Scenarios for the Controller

For a normative assignment capability, the "ideal" scenario would be a fully informed system, where the central controller has complete a priori information about every tripmaker in terms of origin, destination and the timing of the trip, and uses this information to develop an integrated scheme that assigns to each user a path to the desired destination so as to achieve some system-wide objectives. These functional capabilities are envisioned to be available at the so-called "coordinated stage" of ATIS development (Mobility 2000, 1990a; IVHS America, 1992).

It is unlikely that the controller will have full information on O-D trip desires for the complete duration for which the assignment is to be made. A more probable scenario is one where information is available for a short duration into the future through detectors and advance information from drivers. A rolling horizon approach with forecasted future O-D desires is used to obtain models for system optimal assignment for this scenario.

An alternative scenario is that the controller has O-D trip desires only for the present period, and future O-D desires are treated as random variables with known probability distributions (based on historical data), giving rise to a stochastic programming formulation of the problem. Formulations resulting under each of the above information availability scenarios are presented next.

Definition of Variables and Notation

The following variables and notation are used in the various formulations :

i = subscript for origin node

j = subscript for destination node

n = node in the network, $n \in N$

a = arc (or link) in the network, $a \in A$

k = subscript for a path in the network

τ = subscript denoting the time interval in which assignment is made (i.e. departure time)

t = subscript denoting current time interval

Δ = length of a time interval

T' = total duration (peak period) for which assignment is to be made

r_{ij}^{τ} = number of vehicles who wish to depart from i to j in period τ

r_{ijk}^{τ} = number of vehicles who wish to depart from i to j in period τ assigned to path k

$\delta_{ijk}^{\tau ta}$ = dynamic arc-path incidence indicator, equal to 1 if vehicles going from i to j assigned to path k at time τ are on link a at beginning of period t, i.e.

[$\delta_{ijk}^{\tau ta}$ = 1, if r_{ijk}^{τ} is on arc a at beginning of period t
= 0, if arc a does not belong to path k
= 0, if $\tau > t$
= 0, if r_{ijk}^{τ} is not on arc a at beginning of period t]

T_{ijk}^{τ} = path travel time for vehicles going from i to j assigned to path k at time τ

$x_{ijk}^{\tau ta}$ = number of vehicles (i to j) assigned to path k in period τ which are on link a at the beginning of period t

$d_{ijk}^{\tau ta}$ = number of vehicles (i to j) assigned to path k in period τ which enter arc a in period t

$m_{ijk}^{\tau ta}$ = number of vehicles (i to j) assigned to path k in period τ which exit link a in period t

x^{ta} = total number of vehicles on link a at the beginning of period t

d^{ta} = total number of vehicles which enter link a in period t

m^{ta} = total number of vehicles which exit link a in period t

C(n) = set of links directed towards node n

B(n) = set of links directed away from node n

Deterministic Full Information Scenario

This formulation represents the scenario in which the central controller has complete a priori information on O-D desires for the entire duration for which assignment is to be made. The controller assigns to each vehicle a path to its desired destination as the vehicle enters the network so as to optimize some indicator of system performance subject to the applicable constraints.

In the ATIS context, this formulation corresponds to the system optimal (SO) dynamic traffic assignment problem. The formulation incorporates dynamic link-path incidence variables which relate path flows to link flows. The fundamental difficulty in solving dynamic assignment problems (SO or otherwise) is that the dynamic incidence variables are themselves a function of the assignment, giving rise to a complicated fixed-point problem. Essentially, the resulting formulation, which involves nonlinearities in the objective function as well as in the constraints, yields generally undesirable mathematical properties that preclude the guarantee of global optimality. In addition, as explained in the previous section, the dynamic link-path incidence is not trivial. In the static case, flows assigned to a path exist on all the links along that path simultaneously, leading to constant known link-path incidence matrix. Such an assumption in the dynamic case would be clearly flawed, as vehicles starting along a path at a given time are not simultaneously present on all links of the path. The incidence matrix changes dynamically, considerably complicating vehicle conservation constraints and travel time calculations. The advantages of a rigorous formulation using dynamic link-path incidence relationships are offset by the difficulty and intricacies involved in attempting to solve it.

The formulation is detailed below :

Given:

r^{τ}_{ij} , $\forall$ i, j and $\tau = 1,............., T'$

Objective function:

$$\text{Min. } \Sigma_{\tau} \Sigma_i \Sigma_j \Sigma_k (r^{\tau}_{ijk} . T^{\tau}_{ijk})$$

or

$$\text{Min. } [\, T(r^{\tau}_{ijk}), \forall\, i, j, k, \tau]$$

Subject to:

1. $r^{\tau}_{ij} = \Sigma_k \; r^{\tau}_{ijk}$, $\forall$ i, j, τ
2. $\Sigma_c \, m^{tc} = \Sigma_b \, d^{tb}$, $\forall$ t, $c \in C(n)$, $b \in B(n)$, $n \neq i$ or j
3. $x^{ta} = x^{t-1a} + d^{t-1a} - m^{t-1a}$, $\forall$ t, a
4. $x^{ta} = \Sigma_k \Sigma_{\tau} \Sigma_i \Sigma_j \, (r^{\tau}_{ijk} . \delta^{\tau ta}_{ijk})$, $\forall$ t, a
5. $T^{\tau}_{ijk} = \Sigma_t \Sigma_a \, [\delta^{\tau ta}_{ijk} . \Delta]$, $\forall$ i, j, k, τ
6. $\delta^{\tau ta}_{ijk} = f(r^{\tau}_{ijk})$, $\forall$ i, j, k, τ, t, a
7. $d^{ta} = \Sigma_k \Sigma_{\tau} \Sigma_i \Sigma_j \; d^{\tau ta}_{ijk}$, $\forall$ t, a
8. $m^{ta} = \Sigma_k \Sigma_{\tau} \Sigma_i \Sigma_j \; m^{\tau ta}_{ijk}$, $\forall$ t, a
9. $\tau \leq t$
10. $\delta^{\tau ta}_{ijk} = 0 \text{ or } 1$
11. All variables (other than $\delta^{\tau ta}_{ijk}$) ≥ 0

There are two alternative forms for the objective function in the above formulation. The first states that the total travel time of the assigned vehicles in the system is aggregate of the product of the number of vehicles assigned to a particular path (from a given origin to a given destination at a particular time) and the corresponding path travel time. This assumption is realistic when assignment intervals are reasonably small (in which case there are not more than two or three vehicles to a particular path from an origin to a destination). The nonlinearity of the objective function arises from the fact that the travel time on the path is itself a complicated non-explicit function of the number of vehicles assigned to the various paths of the network, via the dynamic link-path incidence.

The second form of the objective function simply states that the total travel time of all vehicles assigned to the various paths during the duration of ATIS application is some function of the assignment. This objective function can be evaluated by any available means. We do it through simulation.

Constraint (1) is a definitional constraint stating that O-D desires assigned to the various paths should sum up to the demand (conservation at the origin). Constraint (2) states that vehicles cannot be stored at intermediate nodes, that is, the number of vehicles exiting from all links incident on an intermediate node should equal the number of vehicles entering all links incident from that node at any given time. Constraint (3) represents the conservation of vehicles on a link and states that the total number of vehicles on any link at the end of the current time interval is the net algebraic sum of vehicles on that link at the end of the previous time period, vehicles entering that link during the current period and vehicles exiting that link during the current period.

Constraints (4), (5) and (6) represent the time-dependent link-path incidence relationships which fundamentally characterize the dynamic assignment problem. Constraint (4) represents the dynamic relationship between the number of vehicles assigned to various paths and their aggregation on links. Constraint (5) illustrates the calculation of the path travel times using the dynamic link-path incidence variables. The number of time steps in which the dynamic incidence variable takes a value 1 implies the number of discrete time steps that a vehicle (or a group of vehicles) spent in the system, and multiplying with A gives the actual travel time in the system. One of the most commonly used indicators of system performance is the total time spent by vehicles in the system, and the path travel times conveniently allow the evaluation of this indicator.

Constraint (6) states that the dynamic link-path incidence variables are a function of the assignment. As noted, this fundamental fact expresses the essence of the dynamic assignment problem.

Constraints (7) and (8) are definitional constraints for the number of vehicles entering and exiting links in the various time intervals. Constraint (9) defines temporal correctness. Constraint (10) restricts the dynamic incidence variables to take values of 0 or 1. Constraint (11) represents the non-negativity requirement.

Rolling Horizon Formulation

This formulation represents a more realistic scenario of the information available to the controller. It assumes that information will be available for a "short" duration into the future. This provides the opportunity to use a rolling horizon approach with forecasted future O-D desires. The basic idea behind the rolling horizon approach **is** that current events will not be influenced by events "far" into the future. In the context of the ATIS problem, this is analogous to stating that vehicles currently assigned will not be influenced by vehicles assigned "far" into the future as the currently assigned vehicles will probably be out of the system by that time. The stage length h in Figure 15 depicts that length of time (its value in actual problems is network specific). The roll period 1 represents the short duration into the future for which O-D desires are available with reasonable certainty. To make an assignment of vehicles to various paths for the current period, the controller requires knowledge of O-D desires for the rest of the stage length as these O-D desires are expected to influence current assignments. These O-D desires may be forecasted based on historical data and current information. The O-D desires beyond the stage length h are assumed to be zero. The situation is now analogous to the complete information availability scenario, albeit, only for the duration covered by the stage length h. The system is solved for optimality only for the duration of the stage length and O-D desires for the roll period (which are known with certainty) are assigned to the paths determined. The time frame is now "rolled" forward by a length equal to the roll period and the above process is repeated till the end of the duration for which ATIS is applied to the system. Hence, a series of optimizations are performed till the planning horizon is covered. The formulation is illustrated below :

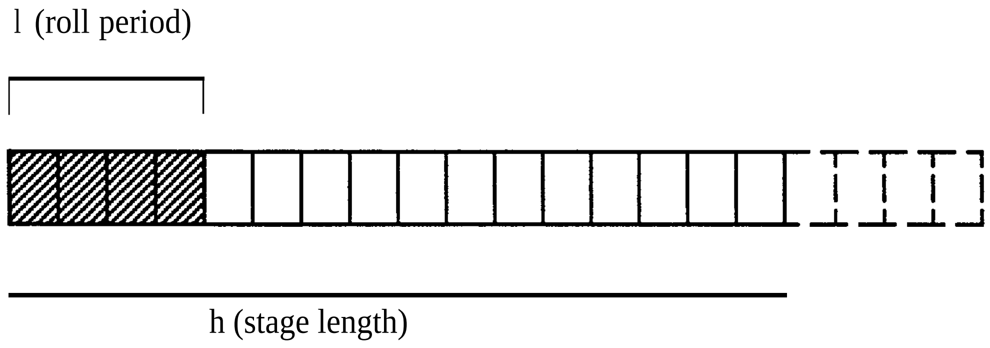


Figure 15. Rolling Horizon Approach

ℓ = The roll period(in number of time steps)

h = The stage length(in number of time steps)

η_ℓ = The current stage number

Given:

1. r^τ_{ij}, r^τ_{ijk}, $\forall$ i, j, k and $\tau = 1,2,\ldots\ldots\ldots\eta_\ell.\ell$, all O-D desires and assigned paths up to current period

2. r^τ_{ij}, $\forall$ i, j and $\tau = \eta_\ell.\ell+1,\ldots\ldots\ldots\ldots, \eta_\ell.\ell+\ell$, O-D desires for a roll period of length ℓ for which assignment is to be made

3. r^τ_{ij}, $\forall$ i, j and $\tau = \eta_\ell.\ell+\ell+1,\ldots\ldots\ldots\ldots,\eta_\ell.\ell+h$, O-D desires forecast for the rest of current stage based on historical and current information

4. $r^\tau_{ij} = 0$, $\forall$ i, j, $\tau > \eta_\ell.\ell+h$, current assignments are not affected by the O-D desires "far" into the future

Objective function:

$$\text{Min. } \Sigma_{\tau=\eta_\ell.\ell+1}^{\eta_\ell.\ell+h} \Sigma_i \Sigma_j \Sigma_k (r^\tau_{ijk} \,.\, T^\tau_{ijk})$$

Subject to:

constraints (1) - (11)

This formulation is identical to the deterministic scenario formulation except for assumptions on the amount of information available to controller and the time frame over which the objective function is evaluated. The formulation is shown for the current stage number η_ℓ.

The path assignments in each stage are determined for the entire stage, but implemented for only the roll period (as only the demand for this period is available with certainty). A number of pertinent questions arise at this point. How far is "far"? What is the "optimal" stage length h? What is a good value for the roll period ℓ? How accurate are the forecasted values for future O-D desires? Is there a need for feedback to check if the assumptions made were realistic? How robust is the solution vis-a-vis the predicted O-D desires? These questions need to be addressed while implementing the solution methodology for the rolling horizon framework. The values of the various parameters are expected to be problem specific. The formulation also emphasizes the need for "good" O-D demand forecasting models.

Stochastic Formulation

Under this scenario, the controller possesses O-D trip desires for the current period only. One way of approaching this problem is by assuming known distributions for future O-D desires. The O-D desires for the remaining assignment planning horizon are a vector of random variables in each interval. A number of vectors are generated for each future interval and a Monte-Carlo simulation for various sequences of random variables is performed to obtain average values for the assignment of the O-D desires to various paths on the network for the current period. The additional variables used in this formulation are :

$\alpha(t)$ = optimal policy for the desired assignments in period t

x^t = vector containing the number of vehicles on each arc at the beginning of period t, based on decisions $\alpha(1), \alpha(2), \ldots\ldots, \alpha(t-1)$ up to time t

$\psi(t \mid x^t)$ = The minimum total travel time from t to end of the planning horizon given x^t.

The formulation is as follows:

Given:

1. $r^{\tau}_{ij}, r^{\tau}_{ijk}$, $\forall$ i, j, k and $\tau = 1,2,\ldots\ldots\ldots\ldots, t-1$, O-D desires and assigned paths up to interval t-1
2. r^{t}_{ij}, $\forall$ i, j, current O-D desires
3. $x^t = \{ x^{ta}, \forall a\}$, current state of network
4. Known probability distributions for future O-D desires

Objective function:

$$\psi(t \mid x^t) = \text{Min. } \{\Sigma_i \Sigma_j \Sigma_k (r^t_{ijk}(\alpha(t)) \,.\, E[T^t_{ijk} (\alpha(t))] + E(\Psi(t+1 \mid xt+1) \mid x^t, \alpha(t))\}$$

Subject to:

constraints (1) - (11)

The objective function $\psi(.)$ evaluates the minimum total travel time from the current period to the end of the planning horizon given the O-D desires for the current period. The objective function consists of two terms. The first term is the product of the assignments in the current period based on the optimal policy $\alpha(t)$ obtained by minimizing the current objective function and the average travel time on the paths (obtained from the MC simulation). The second term is the expected value of the objective function for the next time period given the current O-D desires and the current optimal policy for assignment of O-D desires to paths. Hence, at each time step the objective function evaluates the best decision for the current period in such a way that the expected future objective function is also minimized. The constraints are identical to the constraints in the previous formulation. The objective function is a complicated nonlinear expression and the methodology for its evaluation needs further research.

The solution methodology for the first formulation (full information) is presented next. It may also be applied for the second (rolling horizon).

SOLUTION METHODOLOGY

Simulation-Assignment Approach

A simulation based algorithm is used to solve the system optimal dynamic traffic assignment problem described in the aforementioned problem statement. A traffic simulator is used to evaluate the objective function, ensuring consistency with realistic traffic behavior (FIFO, no holding back of traffic). The procedure assigns vehicles to various paths directly, obviating the need to infer a path assignment from the solution to a link-based formulation. The DYNASMART (Dynamic Network Assignment-Simulation Model for Advanced Road Telematics) assignment-simulation model developed at The University of Texas at Austin is used to simulate traffic.

This section describes the algorithm for SO and UE assignment strategies. It consists of a heuristic iterative procedure in which a special-purpose traffic simulation model is used to represent the traffic interactions in the network, and thereby evaluate the performance of the system under a given assignment. The algorithmic steps for UE assignment are virtually identical to those for the SO solution except for the specification of the appropriate arc costs and the resulting path processing component of the methodology. The algorithm is first summarized for the SO case, followed by a brief description of the modification for the UE problem.

The use of a traffic simulation model to evaluate the SO objective function and model system performance circumvents the principal difficulties that have precluded solutions to realistic formulations of the problem, by obviating the need for link performance functions, link exit functions and implicitly ensuring that the first-in, first-out property holds on traffic facilities and that no unintended holding back of traffic takes place at nodes. The algorithm uses the DYNASMART simulation-assignment model. DYNASMART has the capability to simulate the movement of individual vehicles through the network, with path selection decisions possible at every node or decision point along the way to the destination, as supplied by the user decision rules reflecting driver behavior in response to real-time information. In this work, vehicular paths are pre-assigned exogenously to DYNASMART, as determined by the steps of the SO or UE solution algorithms. Thus DYNASMART is used primarily as a simulator to replicate the dynamics of traffic phenomena in response to a given assignment of vehicles to paths. A detailed description of the various capabilities of DYNASMART is provided in the previous two chapters.

The simulation results provide the basis for a direction finding mechanism in the search process embodied in the solution algorithm for this nonlinear problem. The experienced vehicular trip times from current simulation are used to obtain a descent direction for the next iteration. The time-dependent shortest travel time paths and least marginal travel time paths are obtained using the time-dependent algorithms described in Ziliaskopoulos and Mahmassani (1992), and discussed in Chapter 7. Note that the solution methodology avoids complete path enumeration between O-D pairs.

Description of the Approach

The overall solution methodology is shown in Figure16 for the formulation under which O-D desires are assumed known for the whole assignment duration. It can be suitably modified for the rolling horizon approach which involves repeatedly solving deterministic sub-problems as discussed previously. The algorithm is an extension of well-known solution methods for the static assignment problem, with key differences in each component of the algorithm and significant additional implementation challenges. A brief summary of the approach is as follows:

1. Set the iteration counter I = 0. Obtain the time-dependent historical paths (paths obtained from database) for each assignment time step over the entire duration for which assignment is sought.
2. Assign the O-D desires (which are known a priori for the entire peak period) for the entire duration to the given paths and simulate the traffic patterns that results from the assignment using DYNASMART.
3. Compute the marginal travel times on links using time-dependent experienced or estimated link travel times and the number of vehicles on links obtained as post-simulation data (from step 2).
4. Using a special-purpose time-dependent least cost path algorithm, compute the least marginal time paths for each O-D pair for each assignment time step based on the marginal travel times obtained in step 3.
5. Perform an all-or-nothing assignment of O-D desires to the least marginal time paths

computed in the previous step. The result is a set of auxiliary path vehicle numbers for each O-D pair for each assignment time step t = l,..........., T.

6. Update paths and the number of users assigned to those paths. Update of paths is done by checking if the path identified in step 4 already exists (i.e., has carried vehicles in at least one prior iteration) for that O-D pair and including it if it does not. The update of the number of vehicles (assignment of vehicles to the various paths currently defined between the O-D pair after the path update) is performed using the Method of Successive Averages (MSA), which takes a convex combination of the current path and corresponding auxiliary path numbers of vehicles, for each O-D pair and each time step. A detailed description of MSA is provided in Sheffi and Powell (1982). Note that other convex combination schemes could equally be used.

7. Check for convergence using an E -convergence criterion (in terms of the difference between iterations in number of users on each path).

8. If convergence criterion is satisfied, stop the program. Otherwise, update the iteration counter I = I + 1 and go to step 2 with the updated data on paths and the number of vehicles assigned to each of those paths.

The complexity of the interactions captured by the simulator when evaluating the objective function generally preclude the kind of well-behaved properties required to guarantee convergence of the algorithm in all cases.

It should be noted, to help clarify certain aspects that pertain to the implementation of this algorithm, that the assignment time interval is typically different from the simulation time step used in DYNASMART. The latter is intended to provide an accurate depiction of traffic phenomena, and has a resolution of a few seconds (6 seconds is our default value). On the other hand, the assignment interval corresponds to a period over which O-D demands are not expected to vary much; the decision variables (number of vehicles assigned to alternative paths) are defined for the assignment intervals, which are expected to be of the order of minutes, say 3 to 5 minutes, Therefore, an assignment interval will typically consist of 30 to 50 simulation time steps. All post-simulation information from DYNASMART is available for every simulation time step. Path processing (shortest path) algorithms may proceed at any resolution between the simulation time step and the assignment interval, with different implications for computational efficiency and possibly the accuracy of the procedure. Guidelines regarding these aspects can only be obtained through extensive numerical experimentation.

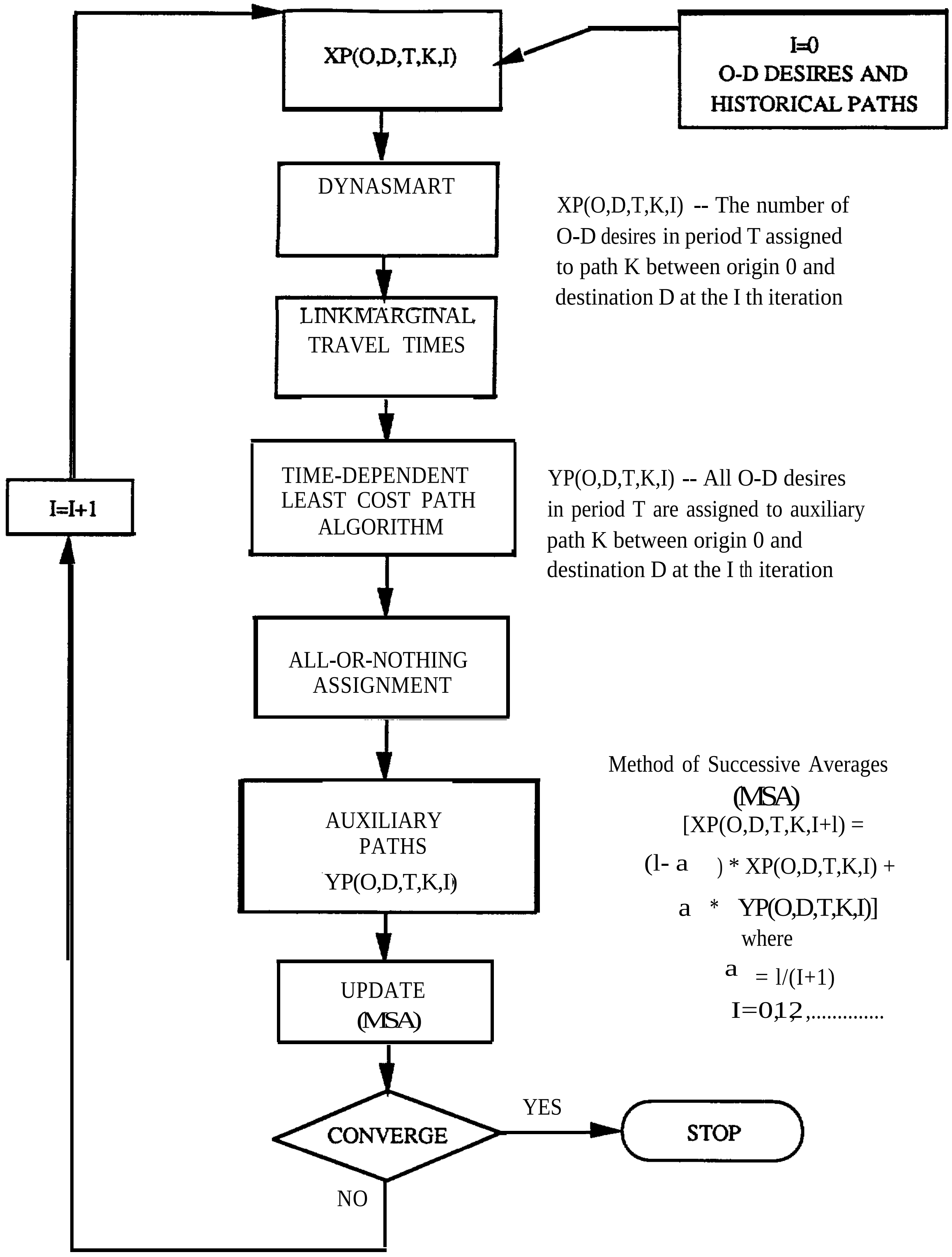

XP(O,D,T,K,I)
I=0
O-D DESIRES AND
HISTORICAL PATHS
DYNASMART
XP(O,D,T,K,I) -- The number of O-D desires in period T assigned to path K between origin 0 and destination D at the I th iteration
LINK MARGINAL
TRAVEL TIMES
I=I+1
TIME-DEPENDENT
LEAST COST PATH
ALGORITHM
YP(O,D,T,K,I) -- All O-D desires in period T are assigned to auxiliary path K between origin 0 and destination D at the I th iteration
ALL-OR-NOTHING
ASSIGNMENT
Method of Successive Averages
(MSA)
[XP(O,D,T,K,I+1) =
(1- a) * XP(O,D,T,K,I) +
a * YP(O,D,T,K,I)]
where
a = 1/(I+1)
I=0,1,2,..............
AUXILIARY
PATHS
YP(O,D,T,K,I)
UPDATE
(MSA)
CONVERGE
YES
STOP
NO

Figure 16. Solution Algorithm for the System Optimal Dynamic Traffic Assignment Problem

Finally, note that the "planning horizon" here is simply the period of analysis; it is subdivided into a number of assignment intervals for which O-D information is expected to be generated.

Modification to Obtain User Equilibrium Solution

As previously discussed, the solution to the time-dependent UE problem is obtained by assigning vehicles to the shortest average travel time paths instead of the least marginal paths in the direction finding step (step 5). In other words, use the (time-dependent) average travel times on links instead of the marginal travel times in the shortest path calculations. In the above solution procedure, this simplifies step 3 and modifies step 4 as indicated.

Discussion of Methodology

This section describes the various components of the simulation-assignment methodology to solve the system optimal problem. The section starts with a brief introduction to DYNASMART. This is followed by an illustration of the approach used to obtain the time-dependent marginal travel times. Next, the time-dependent least cost path algorithm is briefly addressed. The details of the updating mechanism are described, followed by the path assignment procedure which interfaces the update mechanism with DYNASMART.

The Simulation Model -- DYNASMART

DYNASMART is a fixed time step macroscopic simulation-assignment model for IVHS applications, as described in the previous two chapters of this report. When DYNASMART is used in conjunction with the above algorithm, vehicles follow the paths corresponding to the current solution in the execution of the algorithm. This means that the user decisions component is essentially inactive, as user choice consists of following a given path. In the multiple user class formulation, some drivers follow paths determined by the solution algorithm while others follow other rules, possibly including compliance characteristics.

Similarly, the k-shortest path processing algorithms in the simulator need not be executed when DYNASMART operates in a "pure" simulator mode for a given path assignment solution. However, other path processing capabilities, namely time-dependent least time and time-dependent least cost path algorithms are used as part of tire solution framework.

DYNASMART is the basis on which the SO assignment solution methodology is developed. In addition to the previously stated advantages of using simulation to overcome problems like FIFO, holding back of traffic, and computation of an otherwise analytically intractable objective function, DYNASMART furnishes important post-simulation data which forms the basis for the remaining components of the algorithm. After the current simulation, DYNASMART provides data on the average travel times, predicted travel times and number of vehicles on each link of the network for each simulation time step. The above information is used directly or indirectly in the marginal path component, the time-dependent shortest path component, the update component, and the path assignment component of the solution methodology.

Marginal Travel Times

This section discusses the significance of marginal travel times to the SO assignment problem and the approach used to compute approximate path marginal travel times in our solution methodology.

Significance of Marginals in SO Assignment. The SO dynamic traffic assignment problem in the current context aims at minimizing the total system travel time. A global path marginal travel time denotes the travel time increment to the system by the addition of one vehicle to that path. Hence, the solution to the SO dynamic assignment problem would entail assigning O-D desires to the time-dependent shortest global marginal path in order to obtain a descent direction towards the desired minimum time solution (see Ghali and Smith (1992)). The computation of global marginals would entail computationally intensive brute force approaches to capture secondary effects that arise from network interactions over different time periods. The approach used here calculates the marginal costs in only an approximate manner that ignores some of the spatial and temporal interactions taking place in the network. In particular, the marginal cost imposed by an additional vehicle on a given path at a particular time is assumed here to be limited to impeding vehicles on the links constituting that particular path (still correctly recognizing the time-dependent incidence of that vehicle on the links). It may therefore be possible to improve on the solution obtained by using a more elaborate procedure to estimate the marginal costs.

A “first-order” approximation to the marginals is proposed by limiting the marginal travel time on a link to the travel time contribution of an additional traveler on that link to the total travel time on that link. The path marginal total travel times are obtained by a summation of the time-dependent marginal link travel times (the marginal link travel time includes the actual time-dependent link travel time and the time-dependent marginal contribution of an additional traveler) for all links on that path.

Methodology for Obtaining Marginals. At the end of the current simulation, the time-dependent link travel times and the number of vehicles present on each link are obtained from DYNASMART. The marginal link travel times are obtained according to the following relationship :

For each O-D pair,

mltt(a,t) = tt(a,t) + itt(a,t) . x(a,t)

where

mltt(a,t) = marginal travel time in period t for link a

tt(a,t) = travel time (experienced/estimated) in period t for link a

itt(a,t) = increment in travel time in period t to traveler already on link a due to the additional traveler

x(a,t) = number of vehicles on link a at time t

The product of itt(a,t) and x(a,t) gives total increment in the link travel time due to an additional traveler on link a in period t, which is the sum of the additional increase in travel time that each of the currently present x(a,t) vehicles on link a experience. The key problem here is the evaluation of itt(a,t) which is the derivative of tt(a,t) with respect to x(a,t). The method to evaluate itt(a,t) is illustrated in Figure 17.

Figure 17 shows a plot of travel time tt(a) on link a versus the number of vehicles x(a) on link a. In the static case, the link travel times and flows are assumed constant. Hence, the calculation of the derivative using a static link-performance function is trivial. In the dynamic case, the travel times and the number of vehicles on a link are time-dependent. Consequently, the derivative is time-dependent and this makes its evaluation problematic.

The approach we use assumes that the time-dependency of the derivative is due to "time-dependent" link performance functions. This means the tt(a) vs x(a) curve for link a depends on the conditions on the link ***at*** that time, The link-performance curve changes gradually over time which is to be expected due to the dynamic nature of the problem. If the time interval between successive evaluations of marginals is small, it appears reasonable to assume that three consecutive points in time are on the same link performance curve. This assumption is made in the evaluation of itt(a,t) and is illustrated in the graph where three successive time points (intervals t-1, t and t+1) are collapsed onto the same curve. A quadratic fit using the three points results in the time-dependent link performance curve at time t and the slope of this curve at the time t gives itt(a,t) as indicated in the graph. Once itt(a,t) is evaluated, the calculation of mltt(a,t) is straightforward as the values of other variables are obtained from the simulation. The time-dependent link performance curves are obtained assuming that three successive time periods are relatively close to each other. However, the consideration of small time intervals (in the order of a few seconds) may cause some instability in the curves because the values of travel times and the number of vehicles in successive intervals may show "jumps" at times. Hence, there is a trade-off between the approximate correctness of the curves and the robustness of the curves with the use of very short time intervals.

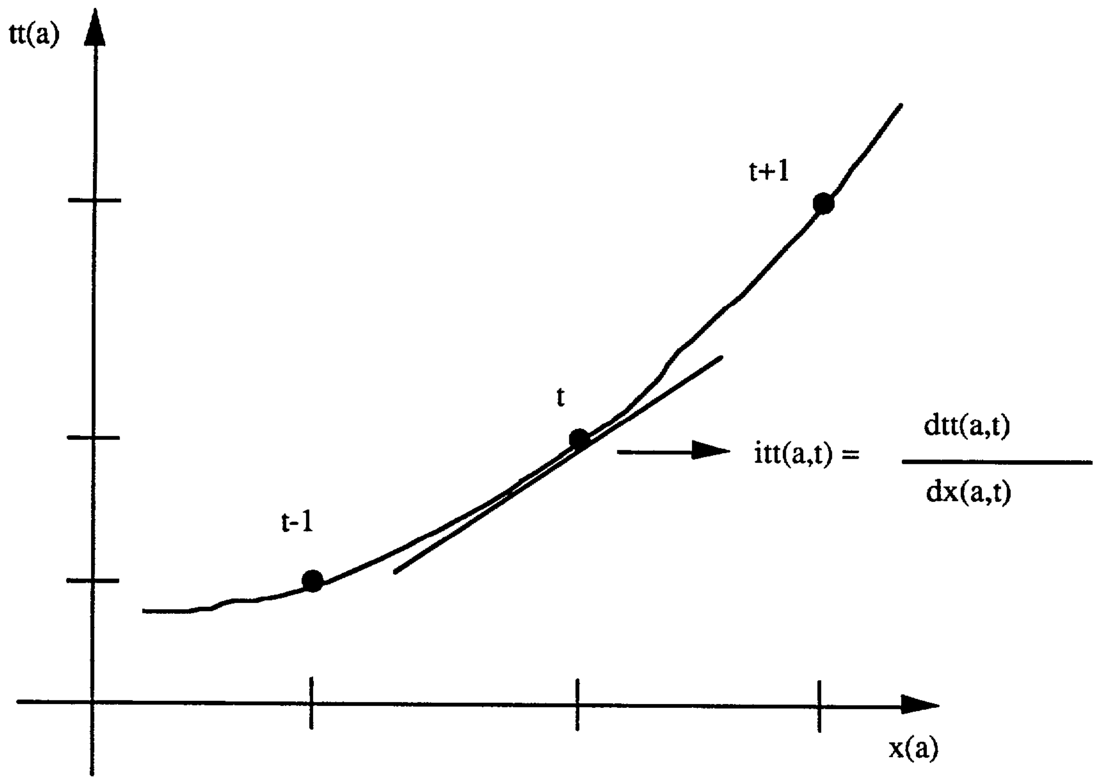


Figure 17. Computation of Marginals

A number of issues arise at this point. First, the robustness of the marginal travel time values over time is crucial to the stability of the curves. This robustness is achieved through the use of averaging techniques. Averaging can be done over time intervals, number of vehicles or both. The simulation time interval we used in DYNASMART is 6 seconds. This interval is too small for update of paths for a given O-D pair as no appreciable change takes place in the system in such a short time. An assignment interval of 3 to 5 minutes is used for updating the paths. The marginal values are necessary for assignment intervals only and not for simulation intervals. One averaging technique to obtain the marginal travel time for the assignment intervals is by averaging the marginal travel time values for the simulation intervals between successive assignment intervals and using that value for the latest assignment interval. Computing a moving average of marginals over the last n assignment time intervals is another averaging technique (that has also been tested in ***some*** versions of our implementation). The marginal values can also be computed by taking an average of the marginal travel times for all vehicles which are present on a link sometime during the duration of an assignment interval.

Another issue with regard to the computation of marginal travel times is the value of travel time tt(a,t) that should be used to compute the marginals. Post-simulation data gives two types of travel times, "average" or ("estimated") and "experienced". The average travel time on a link (for a given interval) is based on an analytical model relating speed to the concentration on the link, as well as the estimated queue discharge time. Alternatively, the net effect of the various traffic phenomena interacting at a given location could be captured by the experienced travel times of vehicles in the simulated system. The "experienced" time is the difference between the respective times of exit and entry of a given link by a certain vehicle.

Improvement of the approximate marginal cost calculation methods and selection of appropriate time intervals and averaging techniques are the subject of continuing numerical tests.

Time-Dependent Shortest Marginal Path Computation

An essential element in the application of IVHS to congested traffic networks is the time-dependence of travel times. So-called "anticipatory" real-time route guidance aims at routing vehicles in real-time in a network based on the travel times they would experience on the various links of their path (as opposed to routing based on current travel times). The problem consists of finding the shortest path from a node to all other nodes in a directed graph with time-dependent travel times. Dreyfus (1969) proposed that the problem could be addressed by using Dijkstra's algorithm in an expanded static representation of the time-varying problem for deterministic travel times. Kaufman and

Smith (1990) made explicit an assumption that is sufficient for the validity of Dreyfus' approach and discussed this issue from an IVHS perspective. This problem is reviewed in more detail in Chapters 6 and 7.

The marginal travel times on paths are obtained by a summation of marginal link travel times for all links on that path. Time dependency of the marginal link travel times requires a time-dependent least cost path algorithm to calculate the paths. A state-of-the-art time-dependent least cost path algorithm developed at The University of Texas at Austin which is coded for efficiency in the computational time and customized for use with DYNASMART in the solution methodology is used to obtain the shortest marginal time-dependent (auxiliary) travel paths based on the marginal link travel times ***and*** average link travel times. The various capabilities of the algorithm including recognition of turn movements and use of very efficient data structures are discussed in Ziliaskopoulos and Mahmassani (1992a, 1992b), as well as in Chapters 6 and 7 of this report, An all-or-nothing assignment of the O-D desires is made to the auxiliary paths.

Figure 18 highlights the procedure for the computation of the shortest marginal paths. The time-dependent least cost path algorithm requires average ***and*** marginal link travel times as inputs. This seemingly unimportant detail reflects a subtle but conceptually important point for correct calculation of time-dependent marginal shortest paths. A least marginal path calculation based solely on marginal link travel costs is incorrect because marginal link travel time does not have a physical interpretation. The correct shortest marginal path computation uses marginal link travel times as link penalties and average (or experienced/estimated) link travel times as link movement costs, as illustrated in the following example.

The portion below the flow chart in Figure 18 shows a path from i to j. Starting at node i at time t, link 1 (i-k) is chosen as the next link on the path based on the link penalty mltt(l,t) which is the marginal link travel time on link 1 at time t. However, node k is reached at a time att(1 ,t) and ***not*** mltt(1 ,t) as att(1 ,t) is the actual time taken to move on link 1. Consequently, the marginal link travel time considered on link 2 (k-j) is mltt(2,t+att(1 ,t)) and ***not*** mltt(2,t+mltt(1 ,t)).

The step following an all-or-nothing assignment is the update of paths. It addresses the distribution of O-D desires for a particular O-D pair at a given time to the various "optimal" paths at that time. It was indicated previously that an all-or-nothing assignment would be used in some manner for distributing vehicles to paths. The next section illustrates this approach.

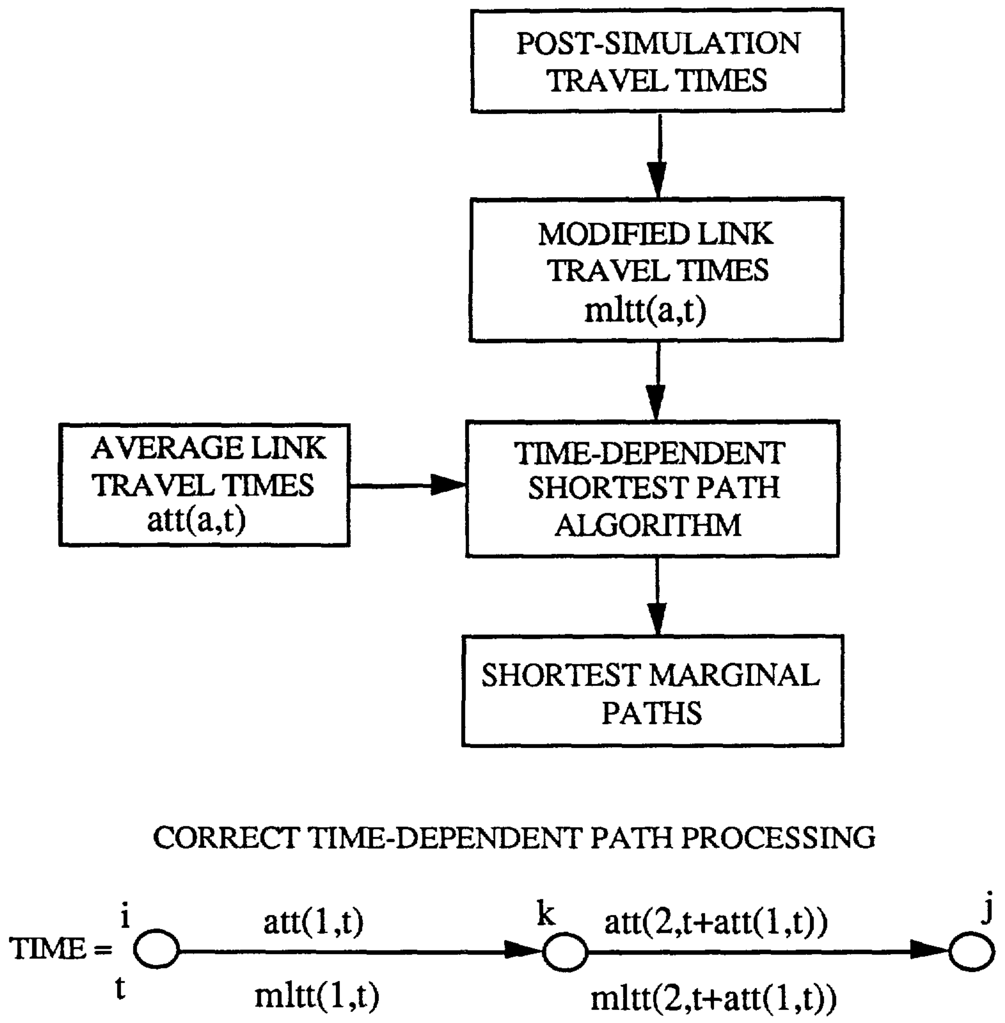

POST-SIMULATION TRAVEL TIMES
MODIFIED LINK TRAVEL TIMES mltt(a,t)
AVERAGE LINK TRAVEL TIMES att(a,t)
TIME-DEPENDENT SHORTEST PATH ALGORITHM
SHORTEST MARGINAL PATHS
CORRECT TIME-DEPENDENT PATH PROCESSING
TIME = t
i
att(1,t)
mltt(1,t)
k
att(2,t+att(1,t))
mltt(2,t+att(1,t))
j

Figure 18. Computation of Shortest Marginal Paths

Update of Paths and Vehicle Assignments

A fundamental requirement for the system optimal dynamic assignment problem in the ATIS context is the ability of the controller to assign vehicle demands for a given origin, destination and start time to various paths in the network. In the solution algorithm, auxiliary paths are generated and an all-or-nothing assignment of the O-D desires is made to these paths during each iteration to obtain a descent direction. Of course, the all-or-nothing assignment does not in itself necessarily represent a better solution than the assignment strategy at the beginning of the current iteration, especially if the time intervals considered are not very small. Assignment on an all-or-nothing basis only could lead to flip-flops from iteration to iteration. Instead, the new solution is obtained by combining the "current" assignment (at the beginning of the iteration) with the auxiliary all-or-nothing solution.

In our approach, the update (smoothing) mechanism is the so-called Method of Successive Averages (MSA). The method of successive averages (MSA), proposed by Sheffi and Powell (1982) for the stochastic user equilibrium problem. It is based on a predetermined move size along the descent direction. The move size α_n is determined a priori and not on the basis of some characteristics of the current solution. The move size α_n has to satisfy certain requirements for an algorithm to converge, and one of the simplest move-size sequences satisfying those requirements is used by MSA. At each iteration, MSA uses the inverse of the iteration number (n) as the move size. Hence,

$$\alpha_n = \frac{1}{n}$$

The vehicle assignments for the next iteration are obtained by using this move size in the search procedure with the descent direction:

$d(n) = y_n - x_n$, where x_n and y_n represent the current solution values and auxiliary values respectively for iteration n. The solution for the next iteration x_{n+1} is obtained as $x_{n+1} = x_n + \frac{1}{n} (y_n - x_n)$ which is equivalent to $x_{n+1} = (1 - \frac{1}{n}) \cdot x_n + (\frac{1}{n}) \cdot y_n$. A comprehensive discussion on MSA is given in Sheffi (1985).

Figure 19 illustrates the details of the update procedure. There are key advantages to updating paths (partial path enumeration) and updating vehicle assignments (splitting of O-D demands) during each iteration of the solution methodology. First, paths are included for a given O-D pair (in an assignment interval) only when they are generated as an auxiliary path. If an auxiliary path (obtained from the shortest marginal paths component) is not already stored for the O-D pair for that assignment interval, it is included. Thus, only partial enumeration of essential paths for the O-D pair is performed.

Path Assignment

In the update component of the algorithm, paths and vehicle assignments to paths are updated for each O-D pair for every assignment interval. The next step in the process, which is an implementation issue, is the assignment of a path to each vehicle in the simulation based on the above values. This interface between the update component and DYNASMART is done through the path assignment component.

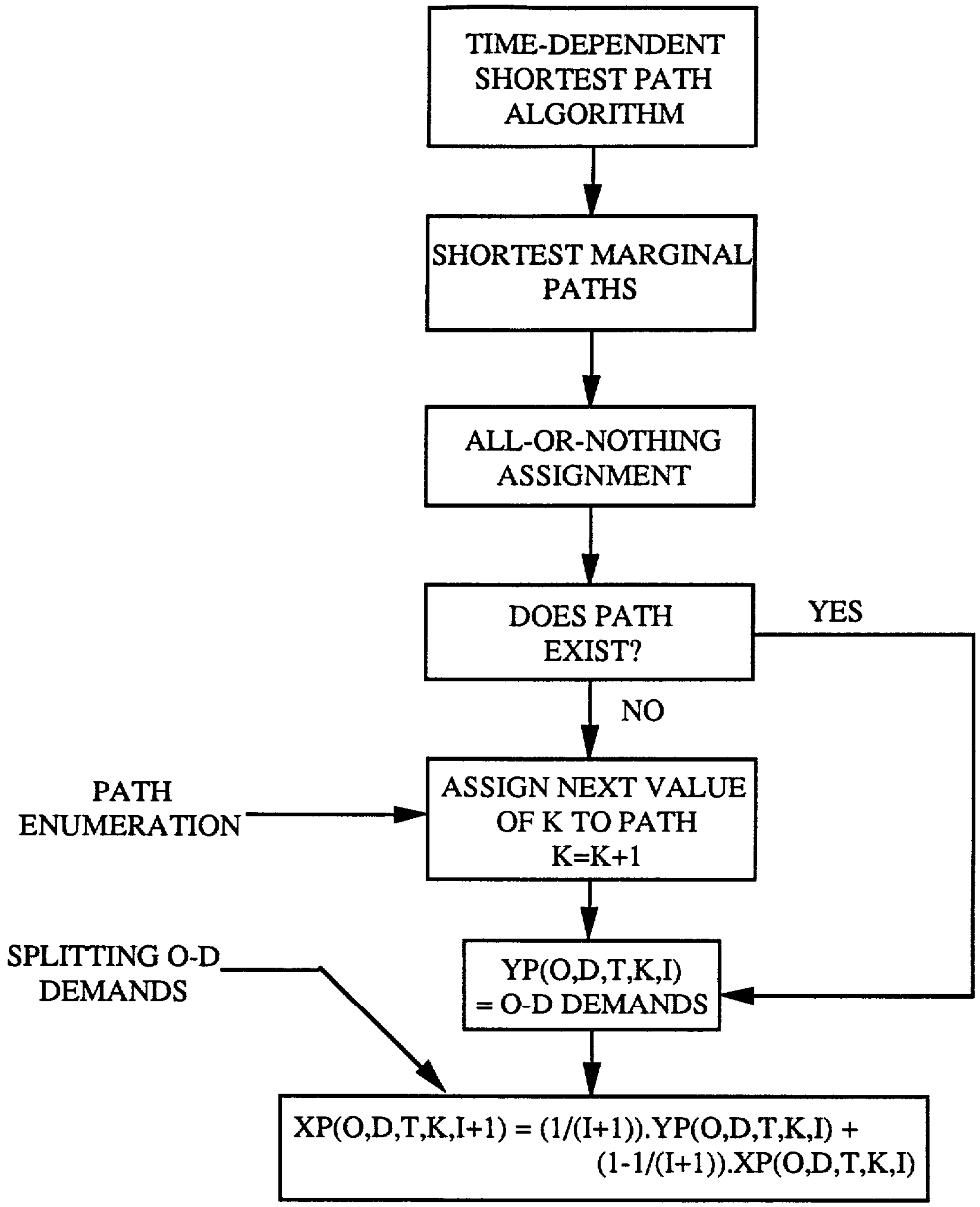

TIME-DEPENDENT SHORTEST PATH ALGORITHM
SHORTEST MARGINAL PATHS
ALL-OR-NOTHING ASSIGNMENT
DOES PATH EXIST?
YES
NO
PATH ENUMERATION
ASSIGN NEXT VALUE OF K TO PATH K=K+1
SPLITTING O-D DEMANDS
YP(O,D,T,K,I) = O-D DEMANDS
XP(O,D,T,K,I+1) = (1/(I+1)).YP(O,D,T,K,I) + (1-1/(I+1)).XP(O,D,T,K,I)

Figure 19. Update of Paths and Vehicle Assignments to Paths

For every assignment interval and for each O-D pair, the time-dependent O-D desires, the set of paths and the splits of vehicles to paths are available at this point. Figure 20 describes the interface with DYNASMART. The path assignment component randomly assigns vehicles to the various paths for a given O-D pair in a given interval while satisfying the requirement on the relative splits of O-D desires to the various paths.

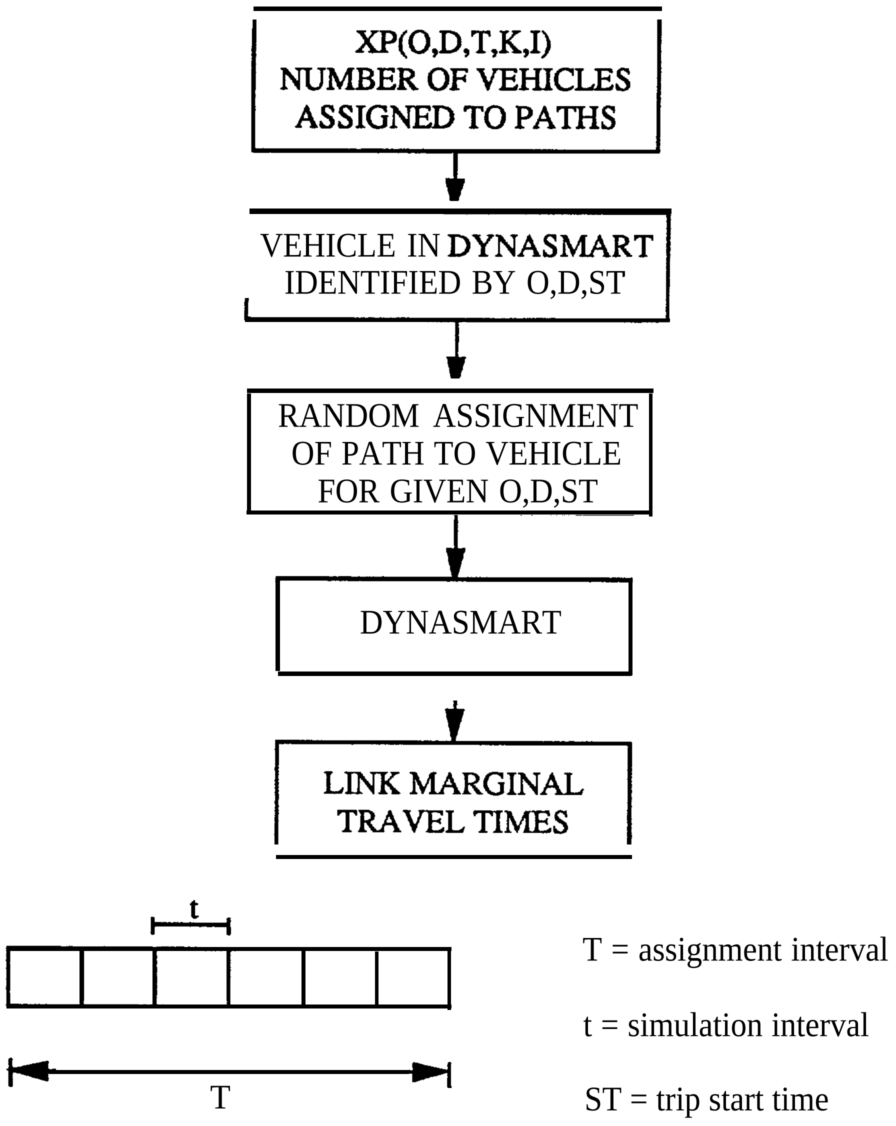


Figure 20. Assignment of Path to Each Vehicle

EXPERIMENTAL ANALYSIS

This section describes two sets of experiments for the single user class dynamic traffic assignment problem. A number of experiments (Experiment Set I) have been conducted **to** derive insights on the dynamic system performance under alternative assignment strategies and under different intensities of network loading, thereby prescribing directions for the focus that ATIS information supply strategies should take, and characterizing the circumstances when alternative assignment strategies will be effective. **A** principal objective of this set of experiments is to provide a comparative assessment of system performance under the system optimal and user equilibrium dynamic traffic assignments. Additional experiments (Experiment Set II) have been conducted to investigate the system performance under another assignment strategy that provides users with descriptive real-time information and assumes users to follow boundedly-rational path switching rules, and to compare the effectiveness of this strategy vis-a-vis the SO and UE strategies. In addition, this set of experiments also tests the sensitivity of the system performance to key parameters such as temporal loading patterns and market penetration. Another principal objective of these two sets of experiments is to analyze time-dependent relationships among network traffic flow descriptors to characterize the vastly varying network traffic conditions during peak periods of traffic flow and to obtain insights into the quality of service afforded under alternative information supply strategies.

Experiment Set I

Motivation and Objectives

The performance of a traffic network employing the solution methodology discussed in a prior section of this chapter is analyzed under both system optimal and user equilibrium time-dependent assignments. As in the static case, system optimal and user equilibrium dynamic assignments involve similar algorithmic steps, differing primarily in the specification of path travel costs that form the basis of the corresponding assignments. System optimal (SO) dynamic assignment is accomplished using time-dependent marginal travel times (see Ghali and Smith, 1991) whereas a user equilibrium (UE) assignment is attained using the time-dependent average travel times. We analyze the system performance under the above assignment schemes for different intensities of network loading covering the spectrum of network states from uncongested networks to very highly congested networks. In addition, the numerical experiments illustrate the extent of the differences between SO and UE time-dependent assignments in terms of total system cost, at varying levels of network congestion. This question is of fundamental importance to ATIS operations, with regard to the relative benefits of normative versus descriptive information supply strategies.

As noted previously, interpretation of the time-dependent UE solution is not evident from the standpoint of ATIS. It is considered here as a useful proxy for a favorable scenario of long-term network performance under real-time descriptive information.

It is known from static network equilibrium theory that SO and UE lead to identical solutions only for situations where the shortest paths taken by users are simultaneously the best paths from a system viewpoint. Such situations are observed when networks are relatively uncongested so that link operating speeds are unaffected by the flows on the links (limited vehicle interactions). At the other extreme, under very highly congested conditions, system performance is not likely to be markedly different under the two assignment schemes because the opportunities for SO to sufficiently ameliorate the traffic situation would probably be limited.

For network conditions between the two extremes, the extent of the differences between SO and UE solutions, particularly in terms of overall system cost, are not known. This is very important for ATIS, because if the two solutions are not perceptibly different, coordinated cooperative SO route guidance imposed by a central controller may not be necessary, and less complicated and simpler to implement descriptive information to non-cooperating drivers may be sufficient. If this were the case, there would be important implications for the focus that ATIS information supply strategies should take, with more attention directed to ways of guiding the system towards UE convergence and away from wide fluctuations. However, if SO indeed holds promise for meaningful gains over UE, then normative route guidance and/or strategies to induce the system near its SO should be pursued. Of course, it is also desirable to ascertain network and traffic conditions under which differences between SO and UE are meaningful.

The overall user cost and network performance under time-dependent SO and UE assignment patterns are examined in a series of numerical experiments performed on a test network under different loading levels. The system performance is gauged using average network level traffic flow descriptors, in addition to the standard parameters like average travel time. The time-dependent nature of the problem further complicates the already intricate problem of characterizing traffic flow performance at the network level, previously addressed only under steady-state conditions, as discussed hereafter.

Network Traffic Flow Theory

Mahmassani, Williams and Herman (1984, 1987) generalized the definitions of speed, flow and concentration to the network level and examined their interrelation in their model of network traffic performance. These concepts are extended to the dynamic case in the current analysis, in order to characterize the vastly varying network 'traffic conditions (especially for medium to high network loading levels) during the peak period. Average network speed V (kmph) is obtained as the ratio of total vehicle-kilometers to total vehicle-hours in the network over the duration of interest. The average network concentration K (vehicles per lane-km), for the duration of interest, is the time average of the number of vehicles per unit lane-length in the system. However, the concentration varies dramatically with time in dynamic traffic networks. Hence, the time-dependent network concentration is examined by taking 5-min averages of number of vehicles per unit lane-length in the system. An overall measure of network concentration K over the duration of the period of interest is obtained by taking the arithmetic average of the 5-min averages. Similarly, time-dependent network flow, interpreted as the average number of vehicles per unit time that pass through a random point along the network, is examined by taking 5-min averages; an overall measure of network flow Q over the peak period is obtained by taking the simple average of $(E\ l_i q_i) / (E\ l_i)$, where q_i and l_i respectively denote the 5-min average flow and the length of link i, and the summations are taken over all network links.

Two fundamental relationships between these three network traffic flow variables are investigated in this study. The first relates average network speed, V, and average network concentration, K. For arterials or single roadways, a qualitative trend of decreasing speed with increasing concentration is well established. The same general trend was observed to hold at the network level in the simulation experiments of Mahmassani et al. (1987), though the complexity of network interactions preclude the analytic derivation of such a relation directly from the link-level relations. The second relationship analyzed is the basic identity $Q = KV$. Formally established for single roadways, it was shown to also hold at the network level in the previously mentioned steady-state experiments (1987). These experiments were performed keeping the network concentration level constant for the duration of interest by treating the network as a closed system. The NETSIM package was used for the study and vehicular behavior was governed by the comprehensive microscopic rules embedded in NETSIM. The present study replicates the network traffic conditions of a rush hour traffic situation, and uses DYNASMART. The Q = KV identity is expected to hold only approximately for time-varying network traffic flow.

Experimental Design and Set-up

This section first details the network configuration and traffic characteristics of the test network used in this study. This is followed by an illustration of the experimental set-up.

Network Configuration and Traffic Characteristics

The test network used in this study consists of a freeway with a street network on both sides as shown in Figure 21. It has 50 nodes and 163 links. Nodes within the freeway section are neither origin nor destination nodes. 38 origin nodes and 38 destination nodes are obtained by excluding freeway nodes (nodes l-37 and 44). Freeway nodes are connected to the street network through entrance and exit ramps. Unless otherwise indicated, all arcs shown are two-directional. All links are 0.83 km (0.5 miles) long and have two lanes in each direction except for the entrance and exit ramps which are directed arcs with a single lane. The freeway links have a mean free speed of 91.67 kmph (55 mph) and the other links have a 50 kmph (30 mph) mean free speed. In terms of traffic signal characteristics, 25 intersections have pre-timed signal control, 8 have actuated signal control and the remaining 17 nodes have no signal control.

Experimental Set-up

The comparative assessment of system performance for system optimal and user equilibrium assignments is conducted under different network loading levels, which generate different levels of network congestion. We define the network loading factor as the ratio of the total number of vehicles generated in the network during the assignment period to a given reference number (19403 vehicles over a 35-minute period in our experiments). Table 15 shows the different loading factors considered in this set of experiments, and the corresponding number of vehicles generated on the test network during the duration of interest (35 minutes in all cases). In addition, it shows the corresponding number of “tagged” vehicles (vehicles generated for the 30 minute duration after the 5 minute start-up time) for which relevant performance statistics are accumulated. The loading factors range from 0.6 (very low congestion with 11616 vehicles) to 2.4 (extremely high congestion with 46674 vehicles). Under each loading

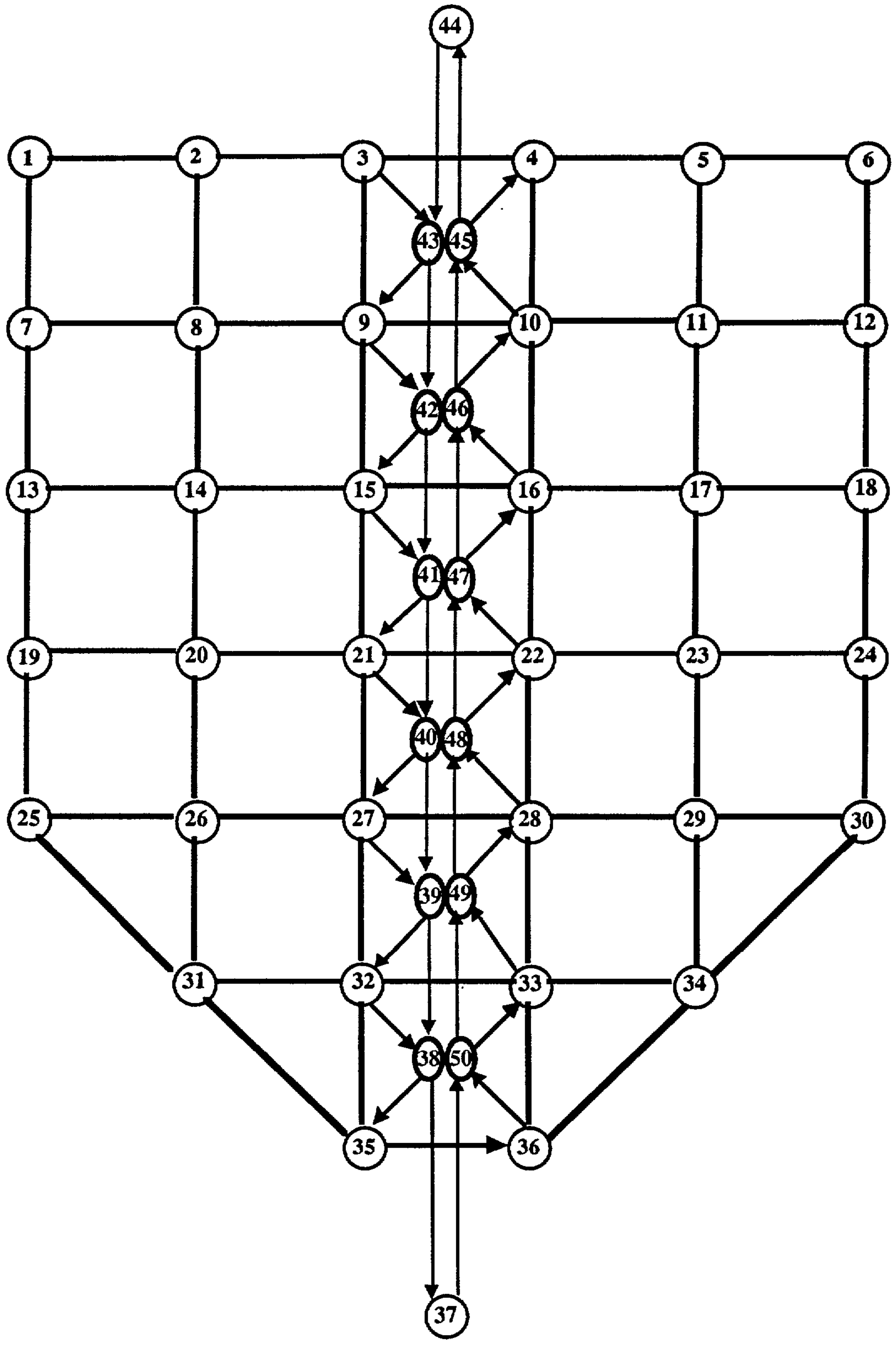


Figure 21. Network Structure for Experiment Set I

Table 15. Loading Factors and the Corresponding Numbers of Generated Vehicles and Tagged Vehicles for the First Set of Numerical Experiments

Loading Factor	Number of Generated Vehicles	Tagged Vehicles
0.6	11616	10585
0.8	15509	14098
1.0	19403	17621
1.2	23305	21145
1.4	27196	24697
1.6	3 1090	28205
1.8	34978	31726
2.0	3887 1	35258
2.1	40818	37014
2.2	42769	38784
2.4	46674	42322

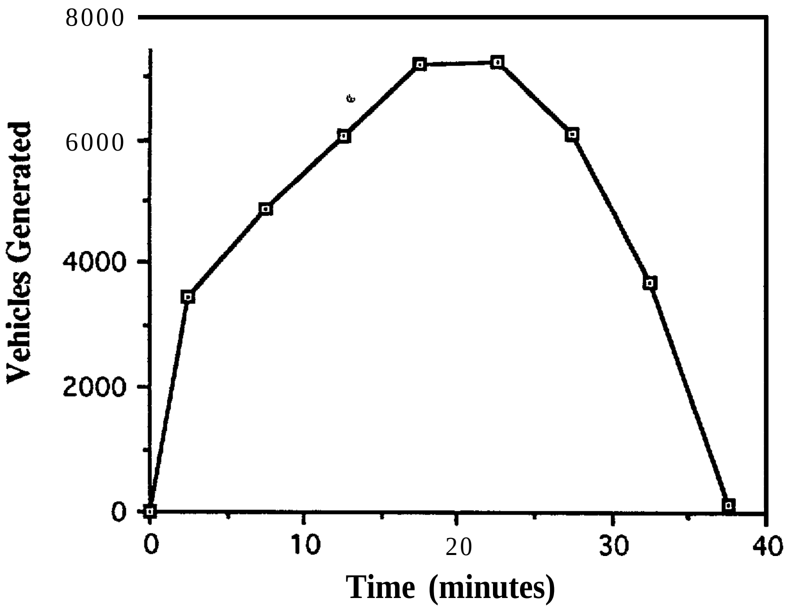


Figure 22. Time-Dependent Vehicle Generation (shown as S-minute aggregates) for Loading Factor 2.0

level, the UE and SO solutions are obtained, and the resulting time-dependent link flow patterns are obtained from DYNASMART. Figure ***22*** shows a sample time-dependent loading pattern for a loading factor of 2.0. The indicated points on the graph correspond to the number of vehicles generated in the 5-minute interval centered on the location of each point; the lines connecting the points are physically meaningless and are included only for visual convenience. The shape of the loading curve for other network loading levels is approximately the same, though appropriately scaled in magnitude. This temporal pattern emulates real-world network loading for the peak period, with an initially increasing generation rate until a peak is reached, followed by a decreasing vehicle generation rate.

In the present study, a start-up time of 5 minutes is provided in DYNASMART for the network to be reasonably occupied, followed by a 30 minute peak period generation of traffic (for which performance statistics are accumulated). Another aspect of the experimental set-up which critically influences the system performance is the spatial distribution of the O-D demand pattern. The vehicles generated are about evenly distributed spatially, both in terms of their origins and destinations, except for nodes 37 and 44 which generate/attract only about 25% the number of vehicles originating/destinated to a typical origin/destination node (i.e. nodes l-36).

Analysis of Results

The results from the various experiments are viewed from two principal perspectives. First, they form the basis for comparison of system performance, particularly user costs under UE and SO assignment schemes, thereby addressing the questions relevant to ATIS information strategies described in Experiment Set I. Secondly, they are used to investigate network level traffic flow characteristics and relations using network-wide traffic descriptors. This investigation is conducted primarily for the SO flow pattern. An additional element of the study is the time-dependent analysis of the travel time gains of SO over UE, also of significance to ATIS operation.

The results provide several key insights from both of the above perspectives. They manifest a clear qualitative and quantitative distinction in the solution provided by the SO assignment scheme as opposed to the time-dependent UE assignment procedure to route vehicles in a traffic network. The results also reveal important and robust macroscopic relationships among network level traffic variables which parallel those for single roadways. Of course, it must be kept in mind that these results are based on a single network topology, and should not be generalized indiscriminately. The primary purpose of these experiments is to illustrate the algorithmic procedure developed for time-dependent SO and UE assignment, and demonstrate its applicability to investigate important substantive questions of network traffic performance.

Table 16. Summary Statistics for System Optimal Assignment

Loading Factor	Av. Trip Time (minutes)	Total Trip Time (hours)	Average Trip Distance (km)	Total Trip Distance (km)	Average Speed (kmph)
0.60	3.85	**679.54**	3.03	**32096.25**	47.23
0.80	3.90	916.05	3.02	42411.67	46.30
1.00	4.03	1183.06	3.03	36820.42	45.22
1.20	4.40	1549.48	3.07	64728.75	41.77
1.40	4.86	1999.10	3.08	76207.92	38.12
1.60	6.04	2837.07	3.20	90222.08	31.80
1.80	7.65	4042.9 1	3.28	103997.50	25.72
2.00	10.46	6149.46	3.32	117013.33	19.03
2.10	13.08	8071.91	3.35	123996.67	15.37
2.20	16.57	10710.93	3.32	128811.67	12.03
2.40	24.95	17601.78	3.55	149978.33	8.52

NOTE: 1 km = 0.6 mile

Table 16 reports summary statistics on the system performance for the SO assignment for the different loading factors. As expected, at low levels of network loading, when the network is relatively uncongested, the average travel times of vehicles in the network are relatively close across the different loading levels. As the load is increased, the effects of congestion become more prominent and the average travel times in the network increase at an increasing rate with the loading factor. At very high loading levels, the marginal effect of additional demand on system performance is very high. The results also indicate that there is only limited variation in the average distance traveled by vehicles under the various network loading levels, implying that greater congestion and not longer travel routes is the primary cause of the higher system trip times (the objective function seeks to minimize total system travel time only). Nevertheless, the average travel distance does increase with the loading level, reflecting an increasing percentage (though small in magnitude) of drivers assigned to longer travel routes.

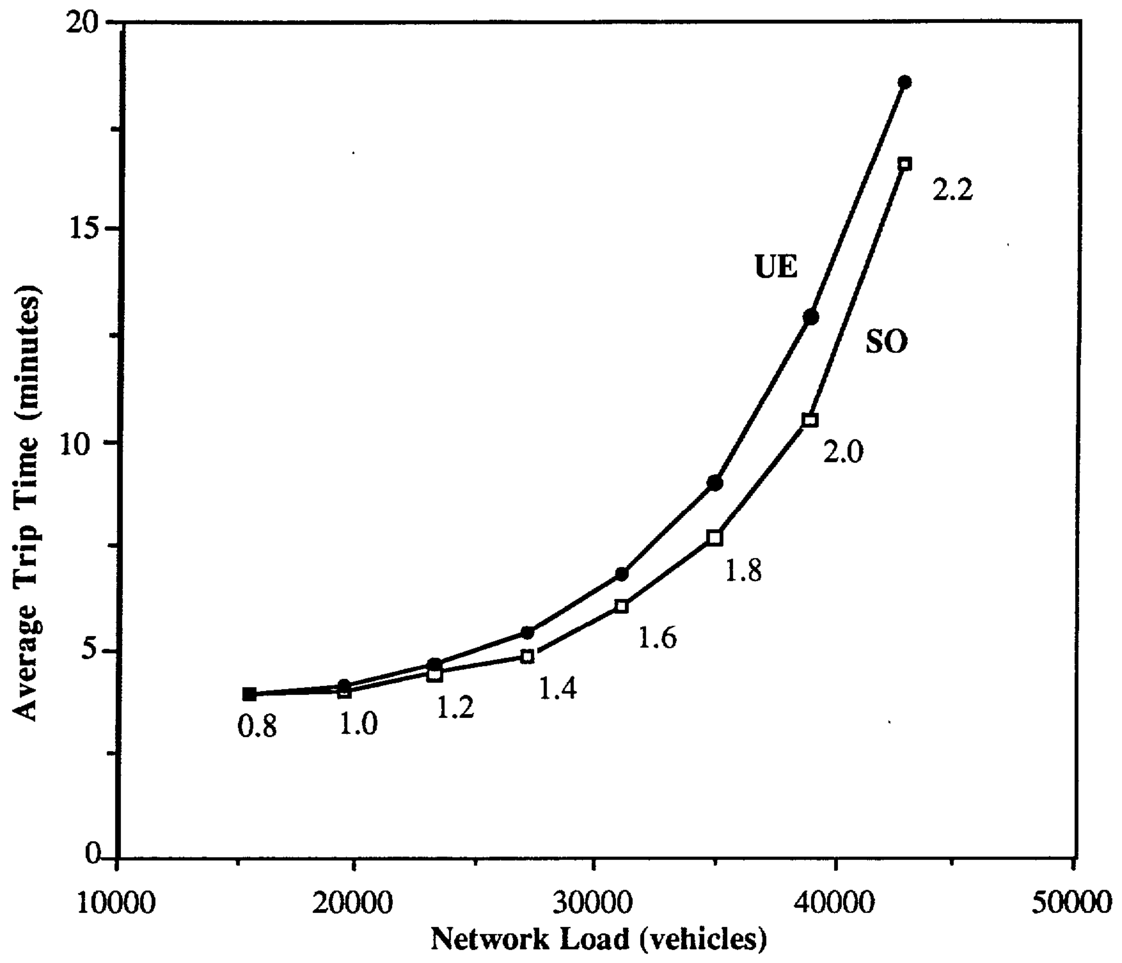
Table 17 presents similar summary statistics for the UE assignment. The trends are similar to those described above for the SO case. The average travel distances under UE for various network loading levels are smaller than the corresponding distances for SO, indicating a smaller percentage of long travel routes under UE. This may be explained by some users being assigned to longer routes in order to reduce congestion elsewhere so as to reduce systemwide travel times.

Table 17. Summary Statistics for User Equilibrium Assignment

Loading Factor	Av. Trip Time (minutes)	Total Trip Time (hours)	Average Trip Distance (km)	Total Trip Distance (km)	Average Speed (kmph)
0.60	3.86	681.52	3.00	31839.58	46.72
0.80	3.92	920.8 1	2.97	41898.33	45.50
1.00	4.15	1219.46	2.98	52656.25	43.18
1.20	4.60	1622.47	3.02	63731.25	39.28
1.40	5.43	2236.52	3.00	74289.58	33.22
1.60	6.79	3192.16	3.08	87 165.42	27.30
1.80	9.00	4762.95	3.13	99513.33	20.88
2.00	12.91	7587.70	3.27	115249.17	15.18
2.10	14.94	9215.69	3.22	119132.50	12.93
2.20	18.55	11993.56	3.32	128605.00	10.72

NOTE: 1 km = 0.6 mile

Figure 23 shows comparatively the average trip times under various network loads for UE and SO assignments. As discussed above, both curves illustrate the increasing marginal effects of additional demand on system trip times. Figure 23 highlights the difference in the quality of the solutions provided by the two assignment rules for time-dependent network flows. This is further illustrated in Figure 24 which depicts the percentage improvement in average travel time of SO over UE (as a fraction of the UE travel time) for the various average network concentrations corresponding to the various levels of network loading. At low loading levels, SO and UE provide essentially

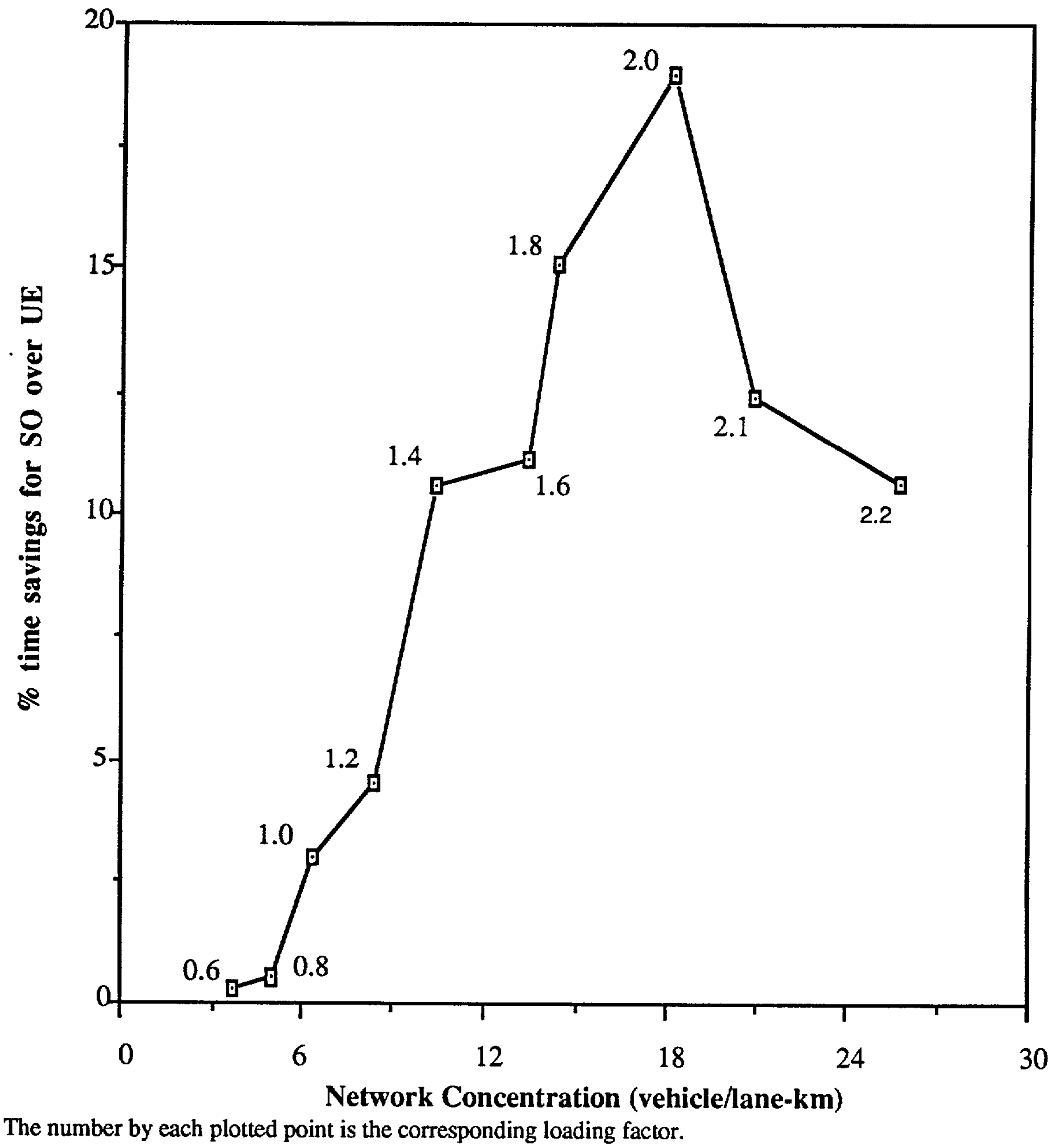


NOTE: The numbers by the plotted points are the corresponding loading factors

Figure 23. Comparison of Average Trip Times (minutes) of SO and UE Assignments for Various Levels of Network Loading

identical solutions. For loading factors 0.6 and 0.8, SO shows improvements of 0.3% and 0.5% respectively over UE. At such low concentration levels, average link speeds remain relatively unchanged due to limited interactions among vehicles, and the marginal travel time on the link is essentially identical to the average travel time, leading to almost identical solutions under the two assignment schemes. When network congestion increases slightly, to loading factors of 1.0 and 1.2, the corresponding SO trip time improvements are 3.0% and 4.5%, respectively, over the UE solution. As the network becomes moderately congested, system benefits under the SO assignment become more pronounced, with 10.6% and 11.2% improvements over UE for loading factors of 1.4 and

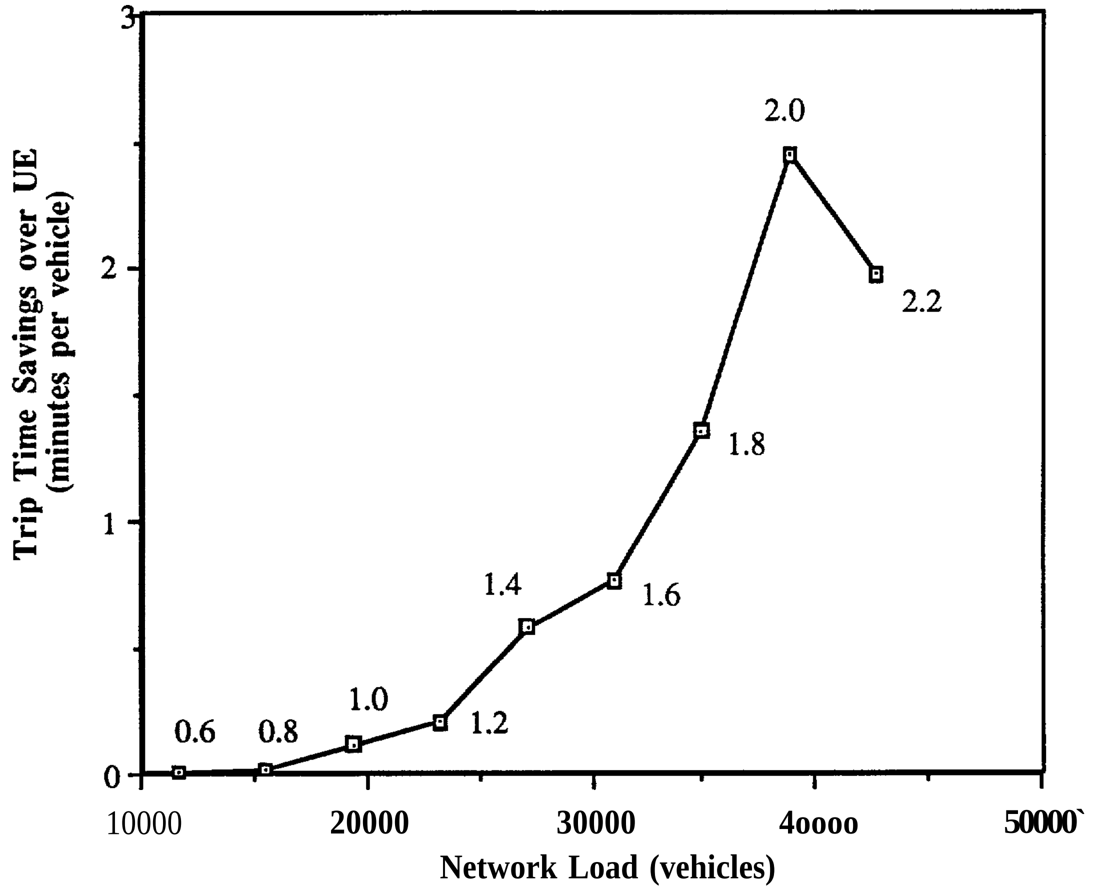
1.6 respectively. For heavily loaded networks, very substantial gains are obtained, with 15.1% and 19.0% improvements in system travel times using SO, for loading factors 1.8 and 2.0 respectively.



The number by each plotted point is the corresponding loading factor.

Figure 24. Percentage Total Trip Time Savings of SO over UE Obtained as a Fraction of Total UE Trip Time for Different Loading Factors Versus Average Network Concentration.

As the levels of network loading are increased further, the system reaches very high levels of congestion that near gridlock, and overall network throughput drops, making it increasingly difficult to discharge all vehicles from the system in a reasonable amount of time. Under these conditions, the ability to improve overall conditions by re-routing certain vehicles to paths with lower marginal costs diminishes, as all links become highly congested. Thus, the advantage of an SO assignment relative to UE begins decreasing, as reflected by reduced improvements of 12.4% and 10.7% for loading factors of 2.1 and 2.2 respectively. The gains begin dropping rapidly beyond this point, with higher loading levels eventually yielding negligible differences in the quality of the solution provided by the two schemes.



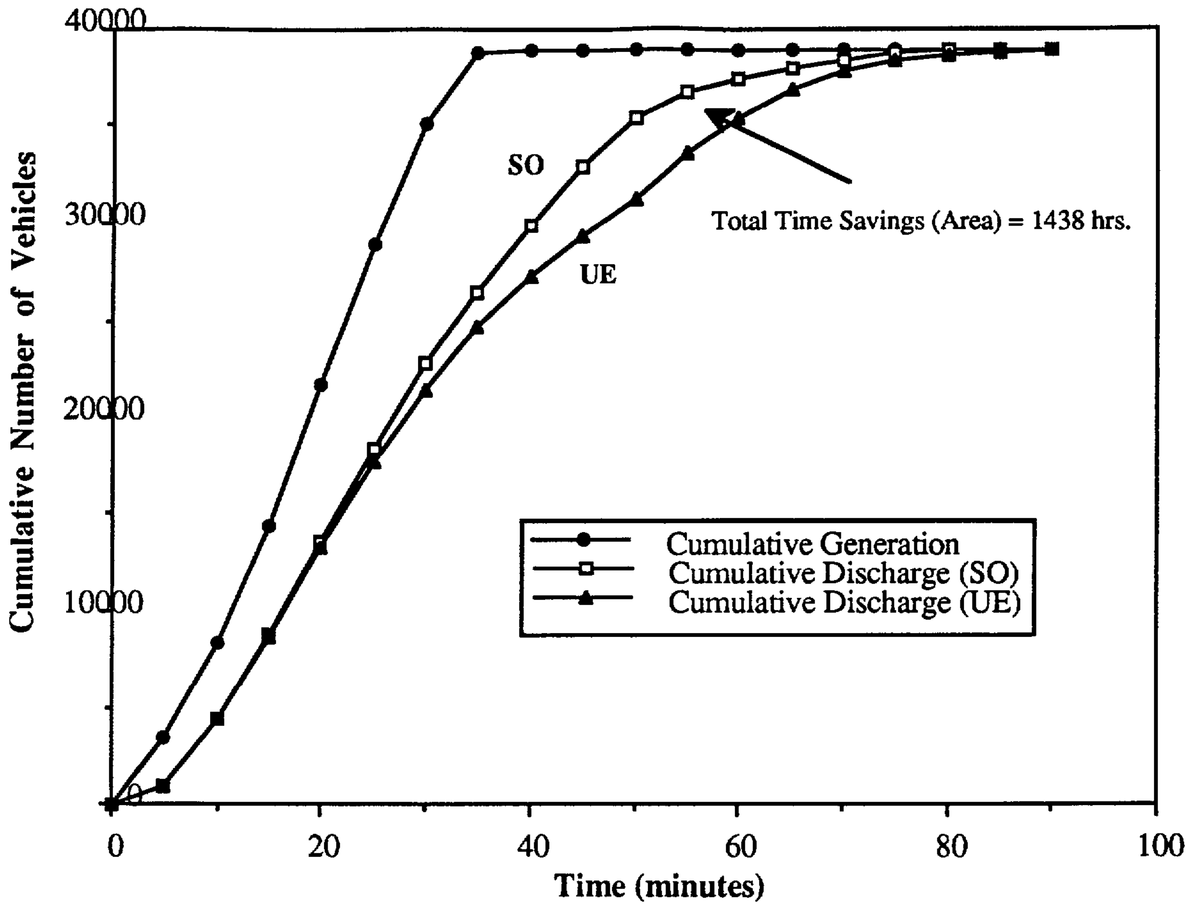
NOTE: The number by each plotted point is the corresponding loading factor

Figure 25. Trip Time Savings for SO over UE (in minutes/vehicle) as a Function of Network Load (the savings are assumed to be equally distributed among all the vehicles generated for that loading factor)

Figure 25 represents the average trip time improvement per vehicle under SO assignment for various levels of network loading. The results mirror the conclusions from Figures 23 and 24. Of course, this improvement in trip time is not experienced uniformly by all vehicles; in particular, it varies over the vehicle's time of departure during the peak period. The time-dependent nature of the travel time savings is examined below.

Figure 26 depicts the cumulative demand generation as a function of time under the 2.0 loading factor along with the cumulative discharge curves under the SO and UE assignments. The various points on the plot are obtained by accumulating the statistics available for each S-minute interval. The area on the plot between the two discharge curves represents the time savings of SO over UE, in this case about 1438 hours. The figure illustrates the time-dependent nature of the benefits generated by SO over UE. When the network is in the early stages of loading (for about the first 20 minutes), it is not sufficiently congested to produce meaningful differences between SO and UE assignments. Most of the savings of SO are accrued between thirty and seventy minutes into the peak period as the network is close to peak congestion levels. Beyond seventy minutes, there appear to be virtually no significant gains of SO over UE as the network is again relatively uncongested. Thus the benefits of route guidance based on SO assignment over UE routing are not accumulated uniformly over time — rather they are gained when the network is relatively well congested.

Figure 27 depicts the time savings per vehicle for SO over UE as a function of the vehicle's time of departure under different loading factors. To capture the time-dependency of the benefits in a systematic manner, travel time savings are accumulated based on the start times of the vehicles. In the figure, O-5 on the y-axis (start time) refers to all vehicles that start between zero and five minutes. Vehicles that start during the first five minutes do not face congested conditions and hence SO does not yield savings over UE for these vehicles. Vehicles that start during the intervals 10-15 and 15-20 minutes accrue time savings at an increasing rate as the (cumulative) loading level increases. Over their trip, these vehicles encounter significant congestion that increases with the loading factor. For vehicles starting between 20 and 35 minutes, the benefits increase with network loading at an increasing rate until the 2.0 loading factor level, and then dip down.



NOTE: The points on the curve represent 5 minute updates of the cumulative number of vehicles. The area between the SO and UE discharge curves represents the time savings for SO over UE.

Figure 26. Cumulative Generation Curve and SO and UE Cumulative Discharge Curves for a Loading Factor of 2.0

The time-varying nature of the savings of SO relative to UE and its dependence on the network load is further illustrated in the Figure 28, which depicts two-dimensional plots of savings as a function of departure time, with each plot corresponding to a different loading factor. The Figure 29 represents essentially similar information but in cumulative form. At a loading factor of 1.2, benefits are just perceptible for vehicles which enter the network during the latter half of the peak period as they face lightly congested conditions. A clearer picture emerges for a loading factor of 1.6 where the network is moderately congested for some duration. Vehicles departing in the first fifteen minutes do not encounter sufficient congestion in the network to obtain significant benefits for a SO assignment relative to UE. As congestion builds up, the SO assignment provides substantial benefits, until a peak is obtained for vehicles starting between twenty and twenty-five minutes. Hence, benefits begin diminishing for vehicles entering the network towards the end of the peak period. At a loading factor of 2.0, the same general trend is observed as above, though it is more marked because of the higher levels of congestion. Very high levels of congestion are observed for some period of time for a loading factor of 2.2, leading to reduced relative effectiveness of SO compared to UE for vehicles that face those congestion levels. This is reflected in the sudden drop of savings for vehicles starting between twenty and thirty minutes.

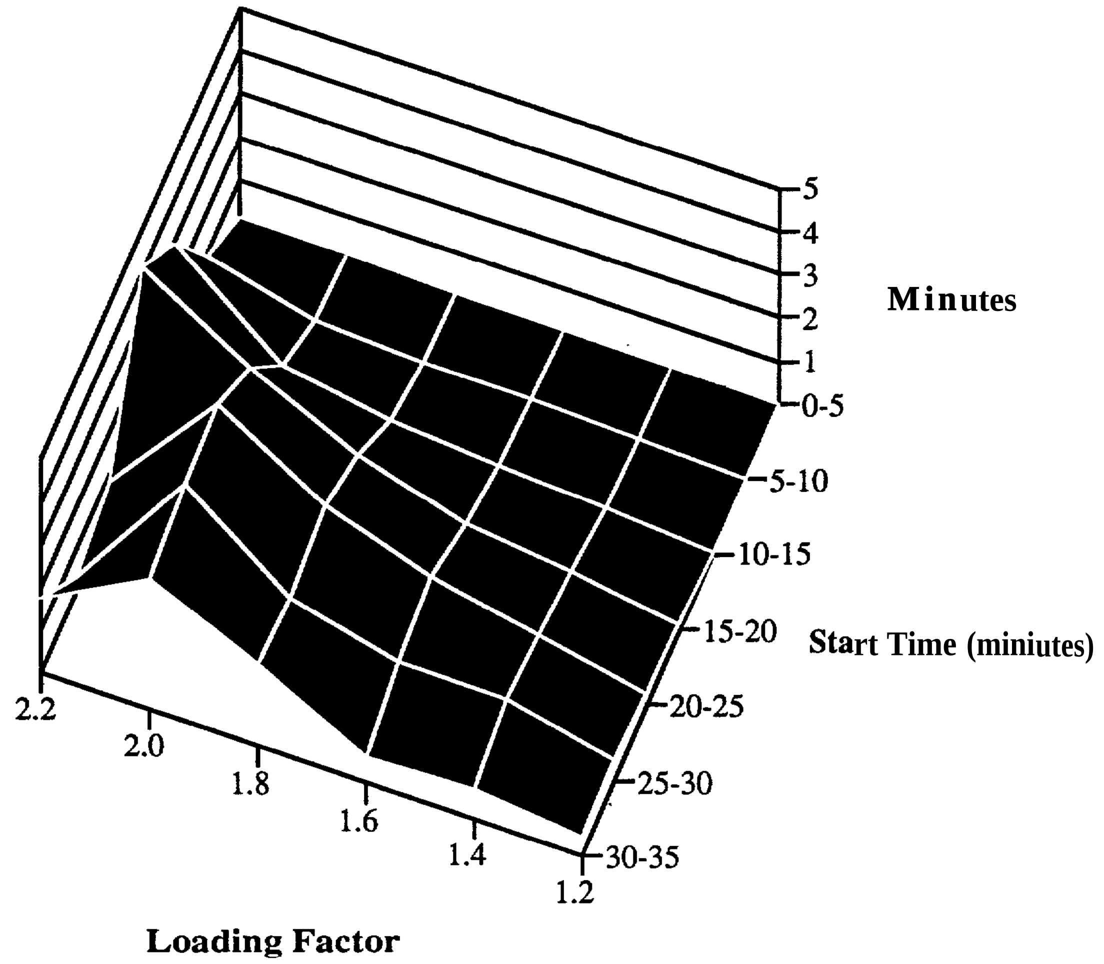


Figure 27. Trip Time Savings (of SO Relative to UE) Per Vehicle (in minutes) as a Function of Loading Factor and Start Times (in minutes) of Vehicles

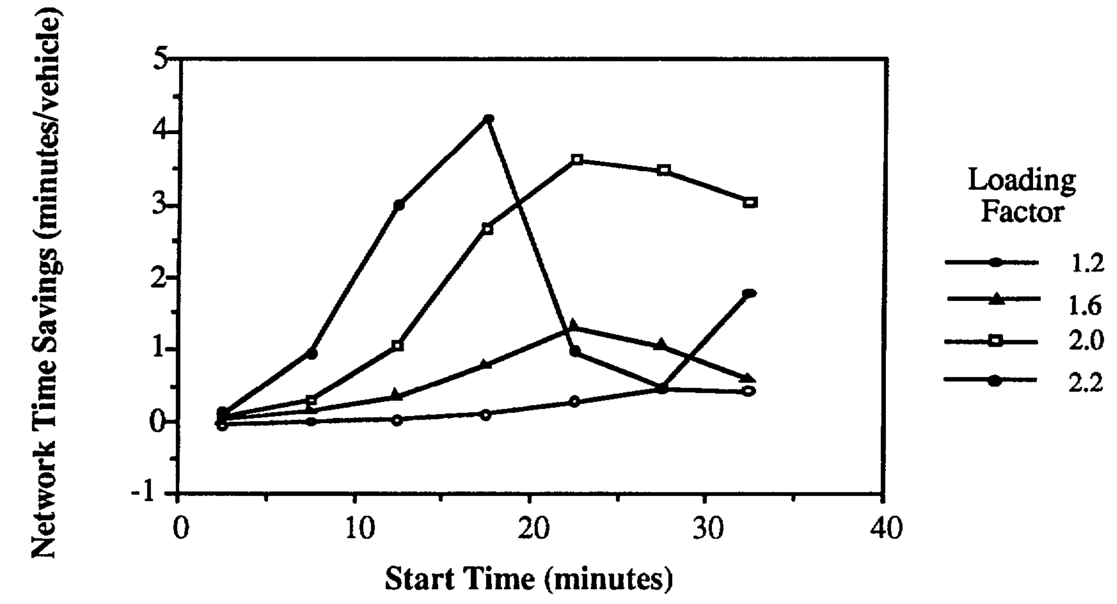


Figure 28. Time Savings Per Vehicle (in minutes) as a Function of Start Time

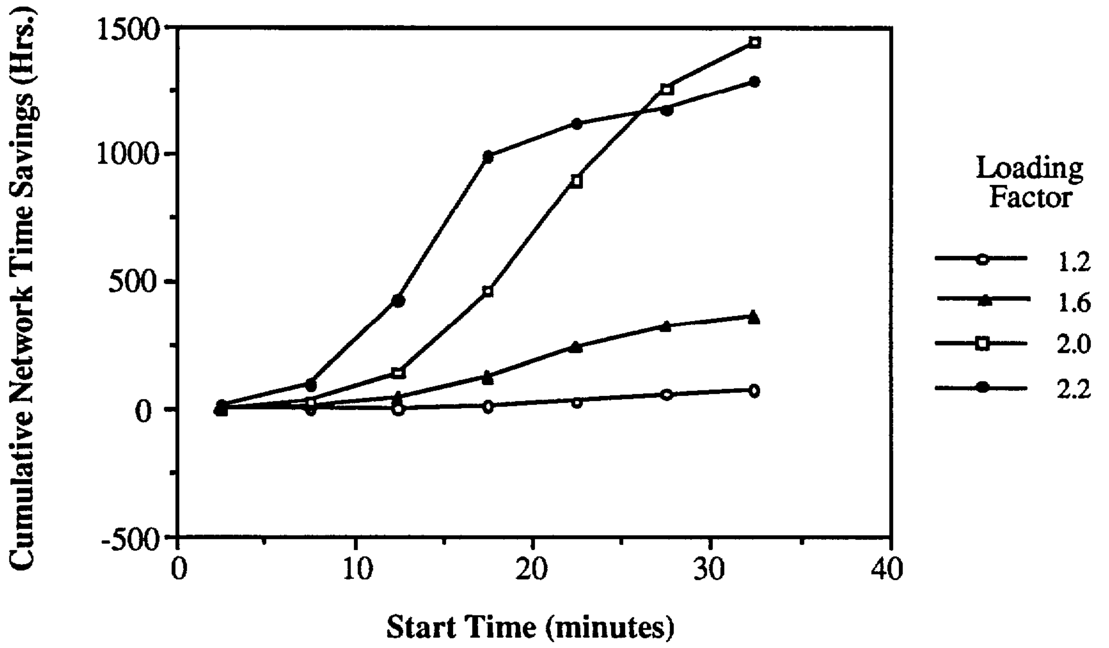


Figure 29. Total Travel Time Savings (in hours) as a Function of Start Time

Network Flow Relations

The second aspect investigated through the experimental results relates to the macroscopic network level traffic theoretic relationships among network-wide traffic descriptors for dynamic traffic networks under consideration. The pertinent traffic variables and their averages over time and space were defined previously. As noted, while mathematical relationships among traffic flow variables are reasonably well established for arterials and intersections, the intricacies of interactions at the network level preclude analytic derivability of network-wide traffic relationships from the link-level traffic models. However, the simulation results extend the previous findings of Mahmassani et al. (1984, 1987) that the basic trends captured by the single roadway relationships seem to also hold at the network level for the dynamic case.

Figure 30 shows the average network speed and average trip time under different network loading levels for the SO assignment. Both curves are smooth indicating relatively robust performance characteristics at the network level, and clearly illustrating the increasing marginal effect of additional demand on the system performance.

The network level speed-concentration relationship for the SO assignment is depicted in Figure 31. Each point on the plot corresponds to a simulation run for the whole assignment period under a particular loading level. The figure clearly illustrates decreasing average network speed with increasing network concentration, paralleling the K-V relationship for an individual roadway. Note that the plot has a point of inflection corresponding approximately to the 1.8 loading factor. This qualitative trend has been observed previously in the simulation experiments of Mahmassani et al. (1984) on a regular test network using the NETSIM package.

Table 18 examines the Q = KV relationship, which holds as an identity for a single roadway. Results indicate that Q and KV differ by less than 5% for all cases, which is well within the error introduced by the manner in which the time averages were computed. As described in the first section, the average network flow and concentration were calculated as an overall average of 5-minute averages, whereas the average network speed was determined through quantities accumulated every 0.1 minute (length of a simulation interval) of the simulation.

Figures 32 and 33 represent the network flow-concentration and speed-flow relationships respectively. The plots indicate that the Q-K and V-Q relationships parallel those for single roadways up to moderate levels of congestion, diverge somewhat as congestion increases, and become confluent for very high congestion levels.

An essential element to be noted in the network level analysis is the time-dependent nature of the phenomena of interest. Averaging quantities like network flow and concentration over the duration of the peak period is likely to mask the time-dependency of network performance. For example, overall network concentration is obtained by averaging low levels of concentration at both ends of the peak period and high levels in between, as shown in Figure 34 which depicts the time-dependent variation of concentration (normalized by dividing by a jam concentration of 96 veh./lane-km) over the duration of interest.

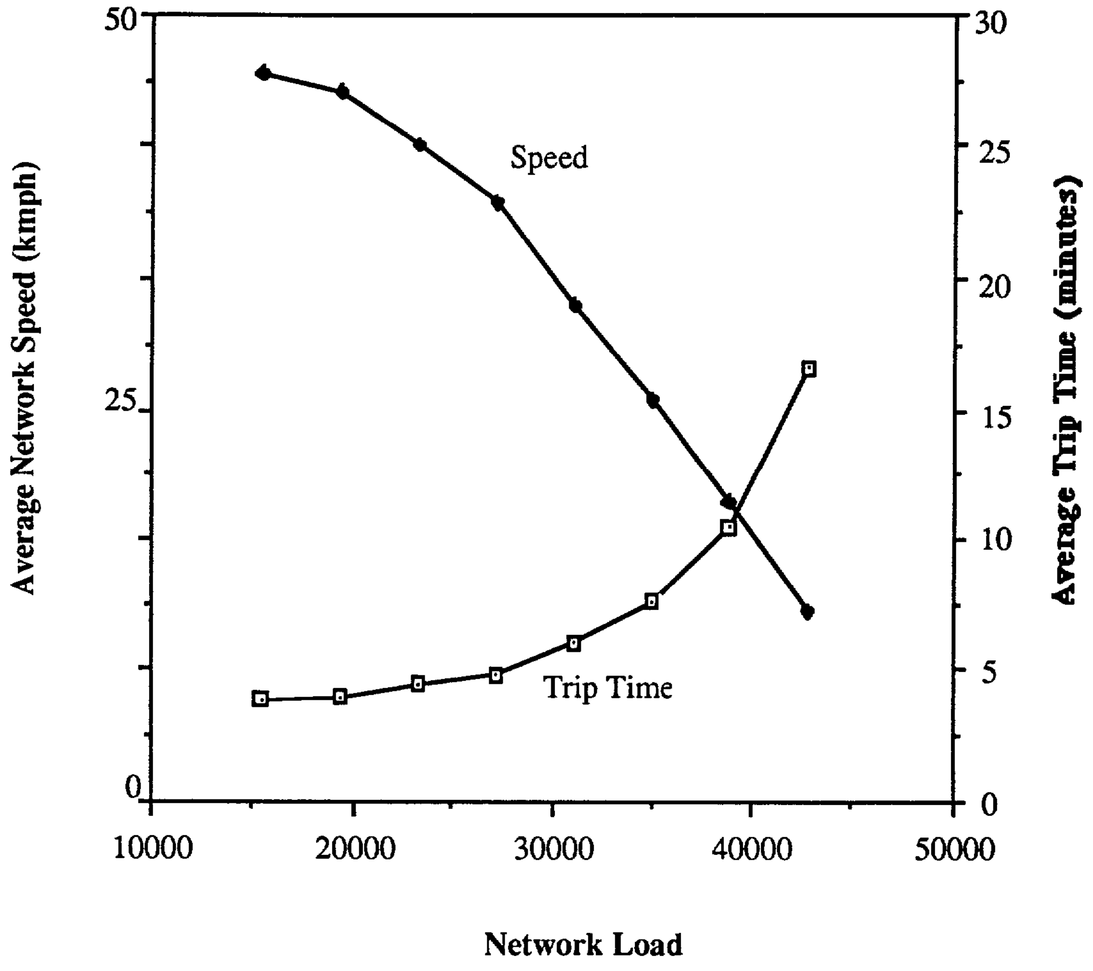


Figure 30. Average Network Speed (kmph) and Average Trip Time (minutes) for the System Optimal Case as a Function of Network Load (in number of vehicles)

Table 18. Results of the Q, KV Comparison

LF	K	V	KV	Q	% Difference
	veh/lane-km	km/hr	veh/lane-hr	veh/lane-hr	(KV-Q)/Q
0.6	3.79	47.23	178.83	170.70	4.76
0.8	5.01	46.30	231.96	222.51	4.24
1.0	6.38	45.22	288.39	275.89	4.53
1.2	8.41	41.77	351.34	335.34	4.77
1.4	10.38	38.12	395.65	378.24	4.60
1.6	13.42	31.80	426.82	408.16	4.57
1.8	14.36	25.72	369.39	353.53	4.49
2.0	18.19	19.03	346.25	331.11	4.57
2.1	20.89	15.37	321.04	306.88	4.61
2.2	25.77	12.03	310.10	296.82	4.47
2.4	31.75	8.52	270.37	259.80	4.07

NOTE: 1 km = 0.6 mile

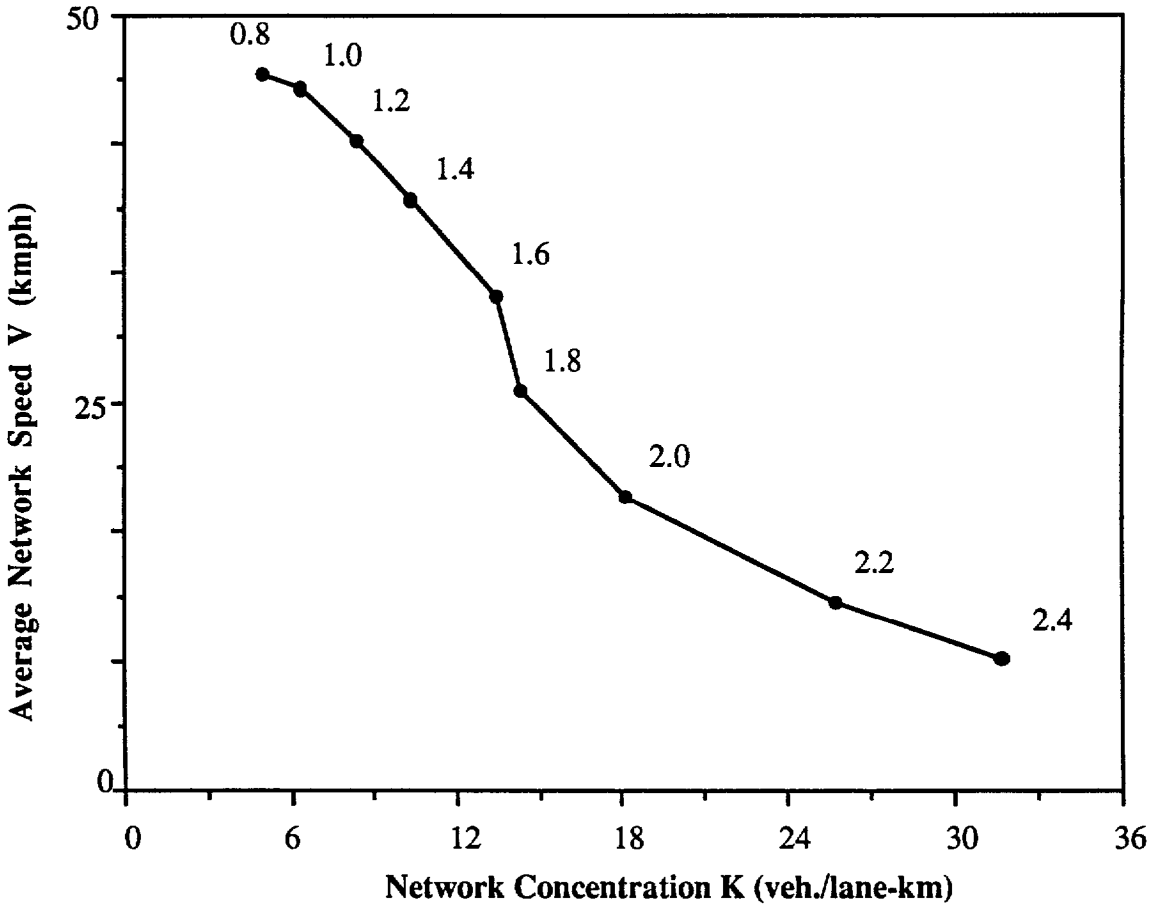


Figure 31. Average Network Speed V (kmph) as a Function of Average Network Concentration K (vehicles/lane-km) for the System Optimal Case

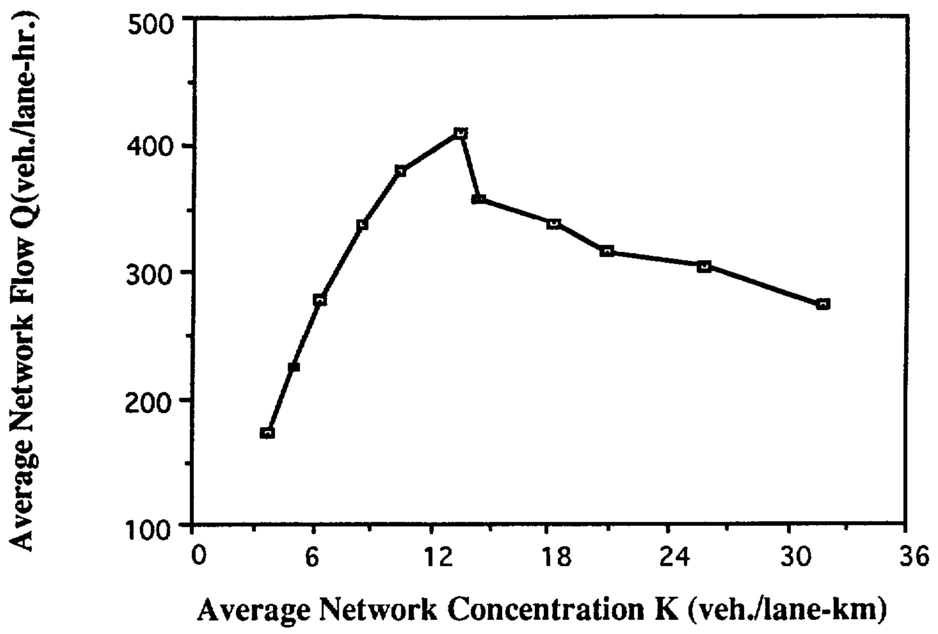


Figure 32. Average Network Flow Q (vehicles/lane-hour) as a Function of Average Network Concentration K (vehicles/lane-km) for the SO Case

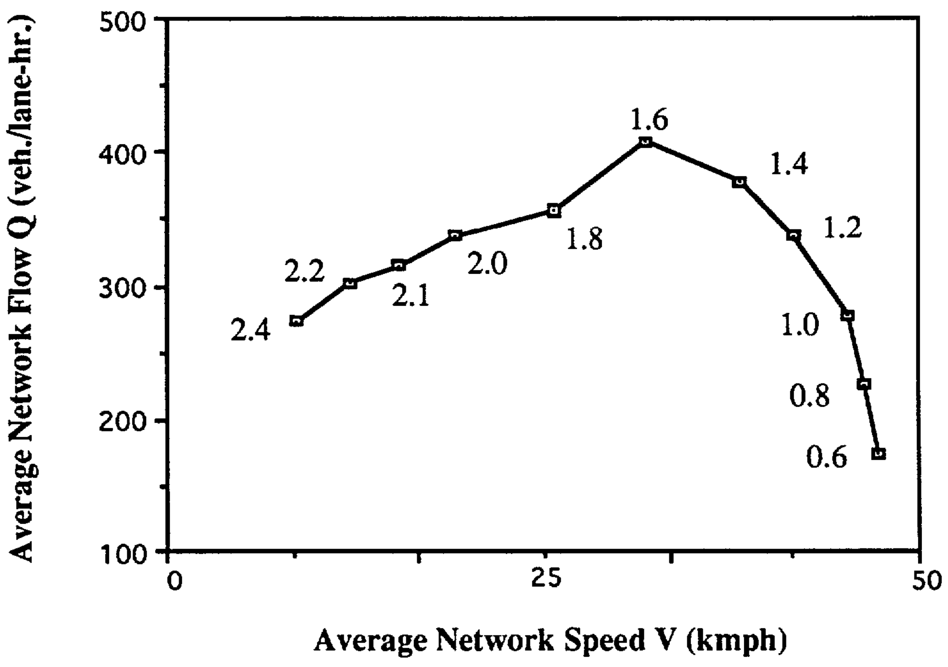


Figure 33. Average Network Flow Q (vehicles/lane-hour) as a Function of Average Network Concentration K (vehicles/lane-km) for the SO Case

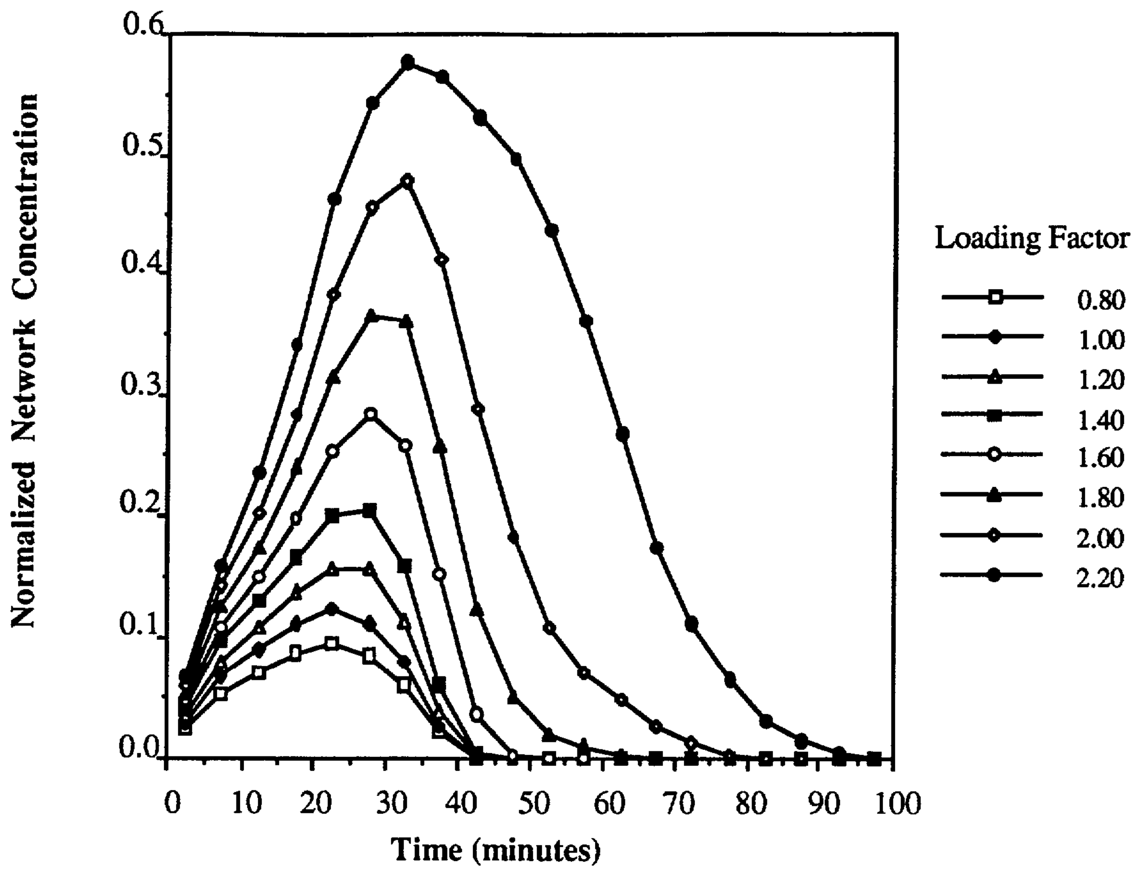


Figure 34. Normalized Network Concentration (network concentration as a fraction of network jam concentration) as a Function of Time for Different Loading Factors

Experiment Set II

Objectives

This set of experiments examines a number of key issues besides those studied in the previous section. System performance is evaluated under a third information supply strategy, along with the SO and UE assignment strategies. This strategy consists of providing users with descriptive information on prevailing link trip times, and allowing them to make route choice decisions (both at the origin and en-route) based on prevailing network conditions. The users are assumed to make route decisions based on boundedly-rational path switching rules whereby the user switches from the current path at a decision point (typically a node on the network) if travel times savings on an alternative route exceed a threshold value. This switching rule was described in Chapter 2, in conjunction with the user decisions component of DYNASMART. System performance under this descriptive information supply strategy is compared to the performance under SO and UE assignment strategies.

Another extension over the previous set of experiments is the study of time-dependent relationships among network level traffic flow descriptors. Here vt, kt and qt denote the time-dependent averages of speed, concentration and flow respectively, corresponding to a five-minute interval starting at time t. The overall aggregate averages are correspondingly denoted by V, K and Q. The time-dependent average network speed vt (kmph) is defined as the ratio of total vehicle-kilometers to total vehicle-hours in the network over each five minute interval t. The overall average network speed V (kmph) is similarly calculated but over the entire duration of interest. The average network concentration kt (vehicles per lane-kilometer) is the time average of the number of vehicles per unit lane-length in the system for time interval t; K is similarly defined over the entire duration of interest. The time-dependent average network flow qt is taken as the average number of vehicles per unit time that pass through a random point (uniformly located) along the network during interval t, and is calculated taking the simple average of $(\Sigma l_i q_{it} / \Sigma l_i)$ where qit and li respectively denote the 5-min average flow and length of link i, the summations being performed over all network links. The overall Q is similarly obtained over the entire duration.

Experimental Design and Set-Up

This section first describes the structure and traffic characteristics of the test network, followed by an overview of the experimental design and the assumptions made in this set of experiments.

Network Characteristics

Figure 35 depicts the test network used in this set of experiments. It is very similar to the previous one, with a few additional links and minor changes to provide better circulation under the myopic switching options. In particular, 5 links have been added, for a total of 168 links. With regard to the intersection signal control, 26 nodes have pre-timed signalization, 8 have actuated signal control and the rest have no signal control. The pre-timed signals have a 60 second cycle length with two phases, each with 26 seconds of green time and 4 seconds of amber time. The actuated signals have 10 seconds of minimum green time and 26 seconds of maximum green time.

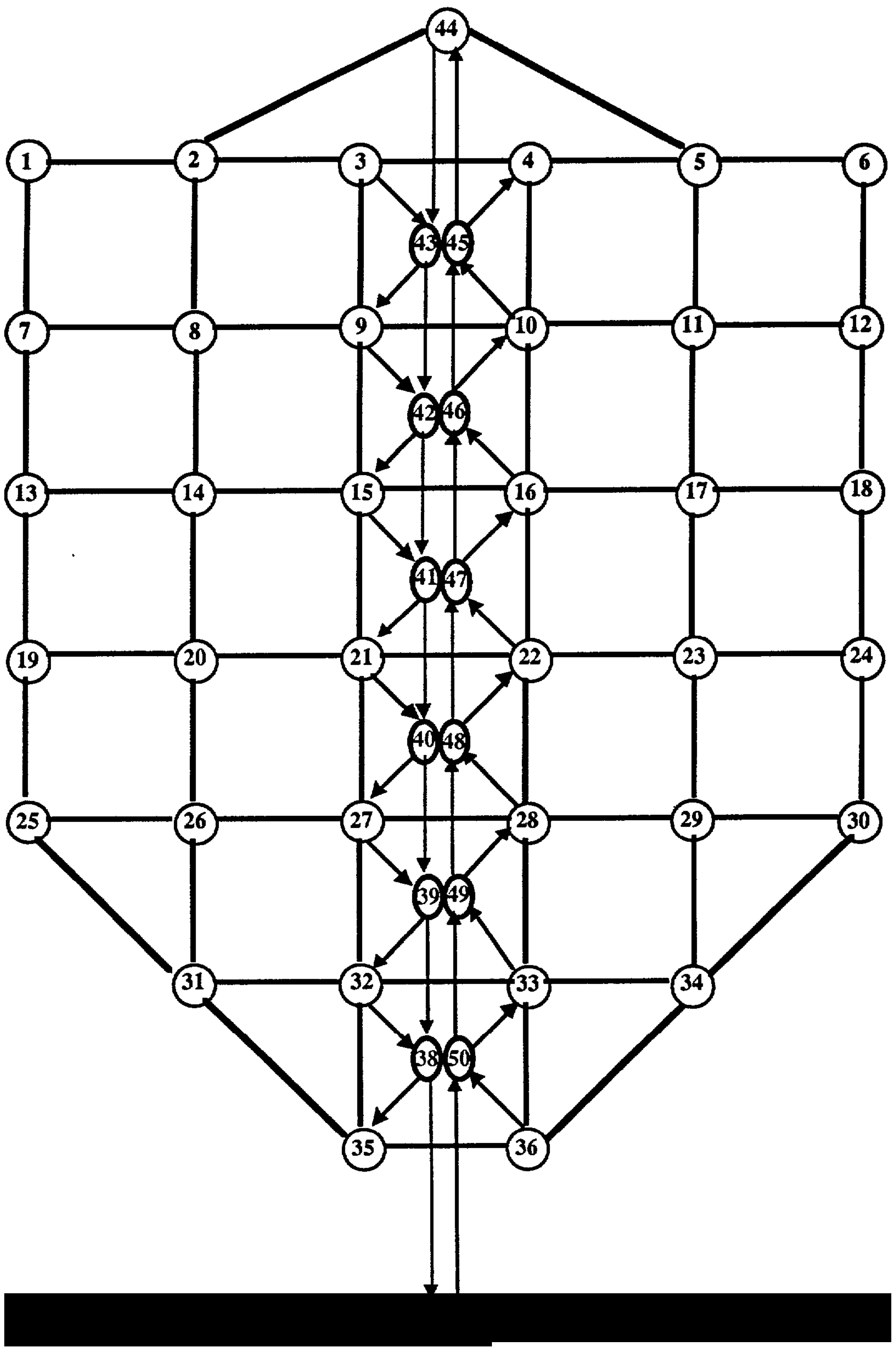


Figure 35. Network Structure for Experiment Set II

Experimental Factors and Design

A number of experiments are conducted to examine network performance under various information supply strategies and assignment rules in the context of electronic route guidance systems.

Experimental Factors. The experimental factors considered in this study can be separated into three primary categories:

1. Loading Patterns: Two loading patterns considered are referred to as loading profiles I and II. Loading profile I generates vehicles uniformly over the assignment duration. Loading profile II impacts the network with relatively large number of vehicles over a ten minute period which is preceded and succeeded by low levels of uniform loading for the rest of the assignment duration. The two profiles are designed so as to represent extremes in loading conditions for the way in which they influence the system performance. A typical peak period loading pattern would most likely lie between these two benchmarks. In both cases, as in the first set of experiments, vehicles are generated over a 35 minute period which includes a 5-minute start-up generation time, followed by a 30 minute generation of vehicles for which statistics are accumulated. With regard to the spatial distribution of the O-D trip desires under the two loading patterns, vehicles are generated about evenly in space, both in terms of their origins and destinations, except for nodes 37 and 44 which incur only about 25% the volume (both as origins or destinations) compared to a typical node (nodes l-36).
2. Demand Levels: The loading factor (LF) is defined as the ratio of the total number of vehicles generated in the network during the assignment period compared to a base value of about 19220 (which represents a loading factor of 1.0). Five different loading factors are considered in the experiments, namely, 1.0, 1.4, 1.8, 2.0, and 2.2. The corresponding number of vehicles generated for each loading factor is detailed in the Table 19. These represent various levels of network congestion ranging from low (for LF = 1.0) to moderately high (for an LF of 2.2). The two loading profiles discussed earlier are designed so that a given loading factor generates about the same number of vehicles under the two profiles.

Table 19. Loading Factors and the Corresponding Numbers of Generated Vehicles for the Numerical Experiments

Loading Factor	Number of Generated Vehicles
1.0	19220
1.4	26936
1.8	34656
2.0	38506
2.2	42371

3. Information Availability (market penetration): Vehicles are differentiated into two classes based on their ability to communicate in real-time with a central controller, and are referred to as equipped and non-equipped vehicles. The properties of each class of vehicles and their driver's assumed behavior are illustrated below:

(i) Equipped vehicles: This class of vehicles has the ability to communicate with a central controller in real-time. Hence, these users make decisions on the selection of their future path to their destinations, achieved through switching from their current path (if deemed so) in light of the information received, based on a set of boundedly-rational user behavioral rules, described in Chapter 2. The parameters used for these rules are assumed to be fixed throughout the experiments. The mean relative indifference band (η) has a value of 0.2 and the absolute minimum threshold bound is assumed to be one minute for all users. The quantity η_j is treated as a random variable, and assumed to follow a triangular distribution, with mean η and a range $\eta/2$.

(ii) Non-equipped vehicles: This class of users does not have the ability to communicate with a central controller, and these vehicles are assumed to follow the initial paths prescribed when they enter the network. Hence, these users do not have the ability to make decisions on switching and the only assumption on their behavior is complete compliance with regard to the initial path they are assigned.

Design of Experiments. Three different types of experiments are designed in the current set, based on the assignment rules and information supply strategies considered:

1. Descriptive information supply strategy with user response rules: This strategy assigns vehicles initially to their current best path, from which they may switch if appropriately equipped. The percentage of equipped vehicles, or market penetration, is an experimental

factor in this set, and takes values of 0.00, 0.10, 0.25, 0.50, 0.75 and 1.00. Hence, for a factor of 1.00, all vehicles have access to real-time information.

2. SO and UE assignment rules: For these rules, all vehicles are assumed to follow the respective paths determined by the corresponding algorithms. The only assumption on user behavior is complete compliance with the supplied information.

3. Descriptive strategy with UE solution as the paths initially assigned: This set of experiments uses the UE solution from (2) as the initial set of paths assigned to vehicles that then follow strategy (1). For the current set of experiments, all vehicles are assumed to be equipped.

The results of these experiments are discussed in the next section.

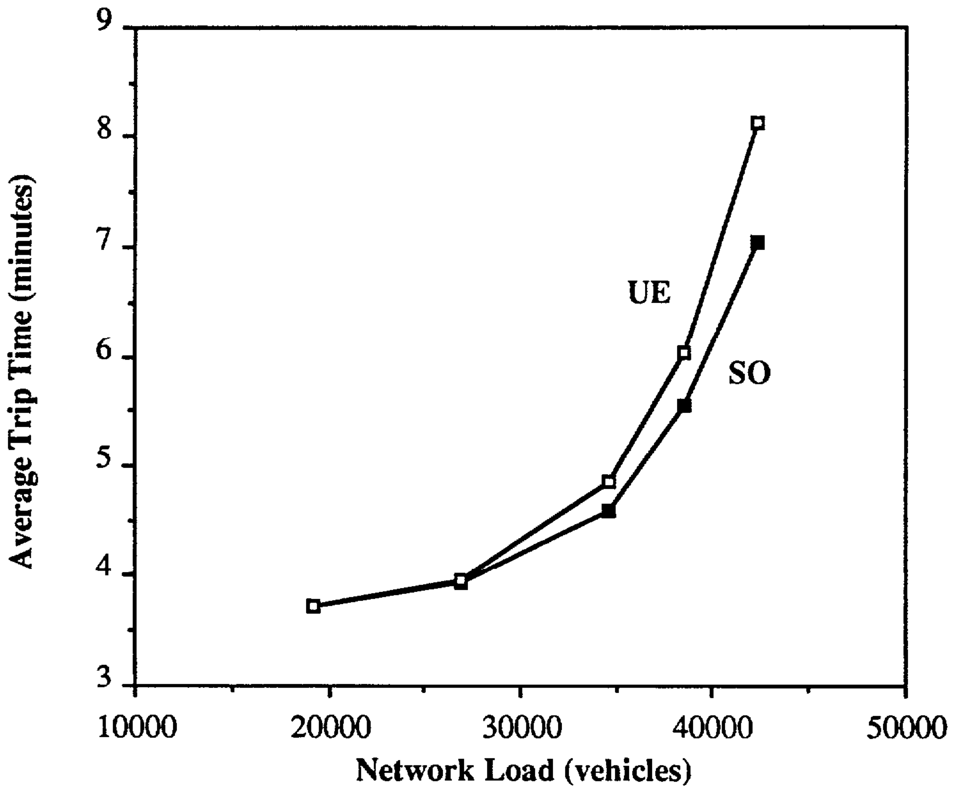


Figure 36. Comparison of Average Trip Times (minutes) of SO and UE Assignments for Network Loads under Loading Profile I

Analysis of Results

As seen in the first set of experiments, the average distance traveled under a UE assignment is lower than the corresponding value for a SO assignment, while the corresponding travel times are of course lower for SO. This indicates that lower system travel times are achieved through routing some vehicles over long (distance) paths. Figures 36 and 37 compare the average trip times under SO and UE for loading profiles I and II respectively. A comparison of average travel times indicates that the system performs better under the uniform loading profile I as opposed to the peaked loading profile II which impacts the network with large numbers of vehicles in a relatively short time. As expected, the marginal penalty of network loading on system travel time increases with network load.

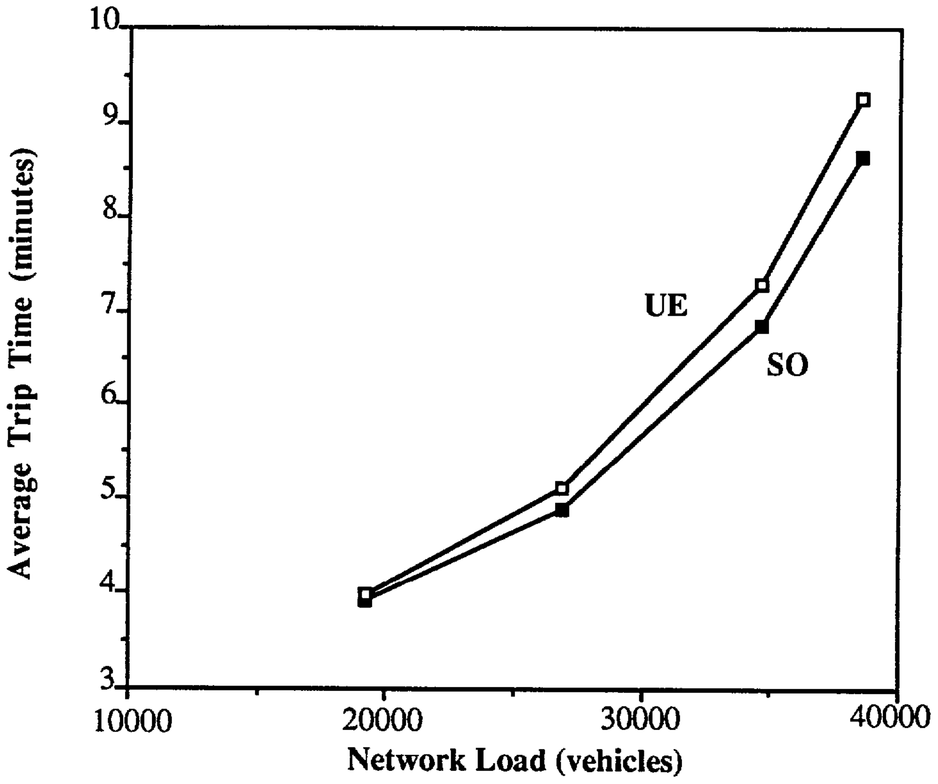
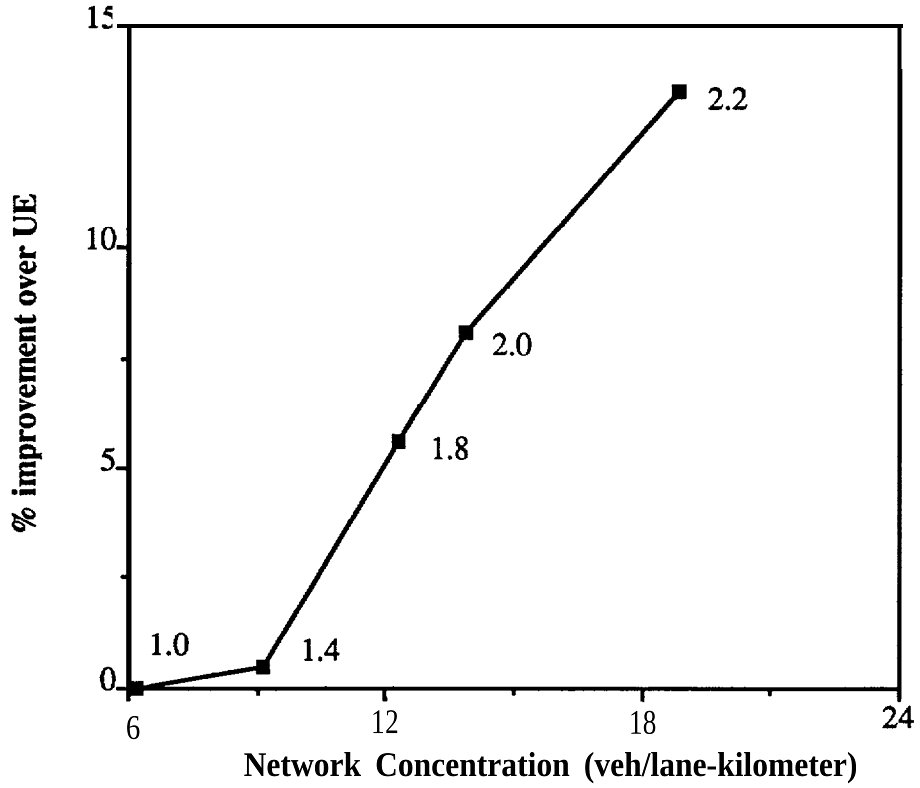


Figure 37. Comparison of Average Trip Times (minutes) of SO and UE Assignments for Network Loads under Loading Profile II

The results obtained here mirror those obtained in the first set, with increased differentiation between the solutions provided by the time-dependent SO and UE assignments at higher network loads, for the low to moderately high congestion levels generated by the experiments. This trend is illustrated in Figure 38 which shows the percentage improvement of SO over UE as a function of average network concentration under loading profile I, and emphasizes a clear-cut demarcation in the quality of solutions provided by SO and UE assignment rules. For example, SO gains 13.5% improvement over UE for moderately high network congestion when the average network concentration for the entire duration of interest is about 18 vehicles per lane-mile. Note that the decreasing trend at very high concentrations in Figure 24 for Experiment **Set** I is not observed here because these concentration levels were not reached in the second set.



NOTE: The number by each plotted point is the corresponding loading factor

Figure 38. Percentage Total Trip Time Savings of SO over UE as a Fraction of Total UE Trip Time Versus Average Network Concentration for Loading Profile I

Figure 39 shows the plots of average trip time versus network loading for SO and UE assignment rules for loading profile I (as in Figure 36), and in addition the average trip times for a strategy in which all vehicles are provided with real-time descriptive information in the network and make switching decisions according to the boundedly-rational user behavior rules (with a 0.2 indifference band and 1 minute threshold in the current case), given that their initial loading onto the network is based on the UE solution. This curve is between the SO and UE solutions, indicating that the system performance under a real-time descriptive in-vehicle information supply strategy superimposed over a time-dependent UE pattern, is better than the corresponding UE solution, though it is unclear what kind of a benchmark the UE pattern would represent in the first place. On the other hand, the SO solution represents the best system performance that can be theoretically obtained, and the fact that it outperforms the descriptive strategy emphasizes the need for coordinated information supply strategies for potentially meaningful enhancements over and above the solutions provided by uncoordinated descriptive information supply strategies. The current set of SO solutions are slightly sub-optimal due to the specification of local as opposed to global marginal travel times for the calculation of time-dependent marginal shortest paths in our methodology, especially at higher network loading levels.

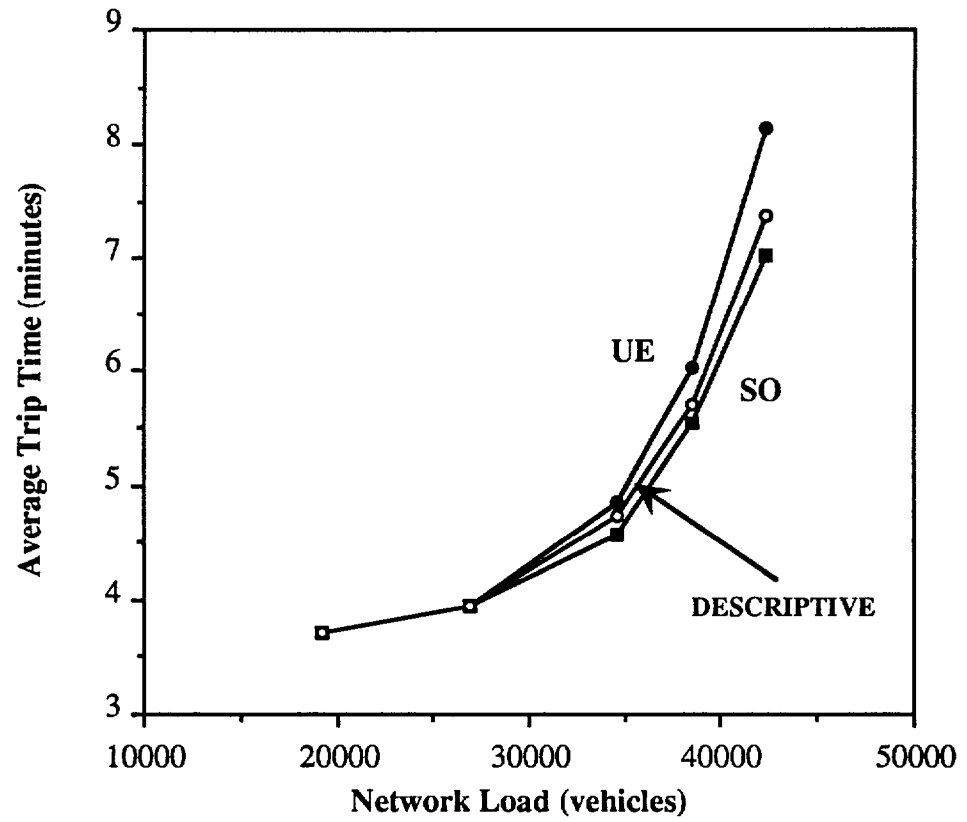


Figure 39. Comparison of Average Trip Tmes (minutes) of SO, UE and a Descriptive Assignment Strategy with UE Solution as the Starting Point, for Various Network Loads under Loading Profile I

The second angle from which results are investigated relates to traffic flow theory in the context of network level traffic descriptors. The various network-wide traffic descriptors used in the current analysis were defined previously. As noted earlier, the well-established trend of decreasing speed with increasing concentration for arterials was found to hold at a network level for both the steady-state conditions and dynamic loading scenario. This trend is further illustrated in Figure 40, which shows the network-level speed-concentration relationship for the UE assignment under loading profile II where each point represents a network loading level.

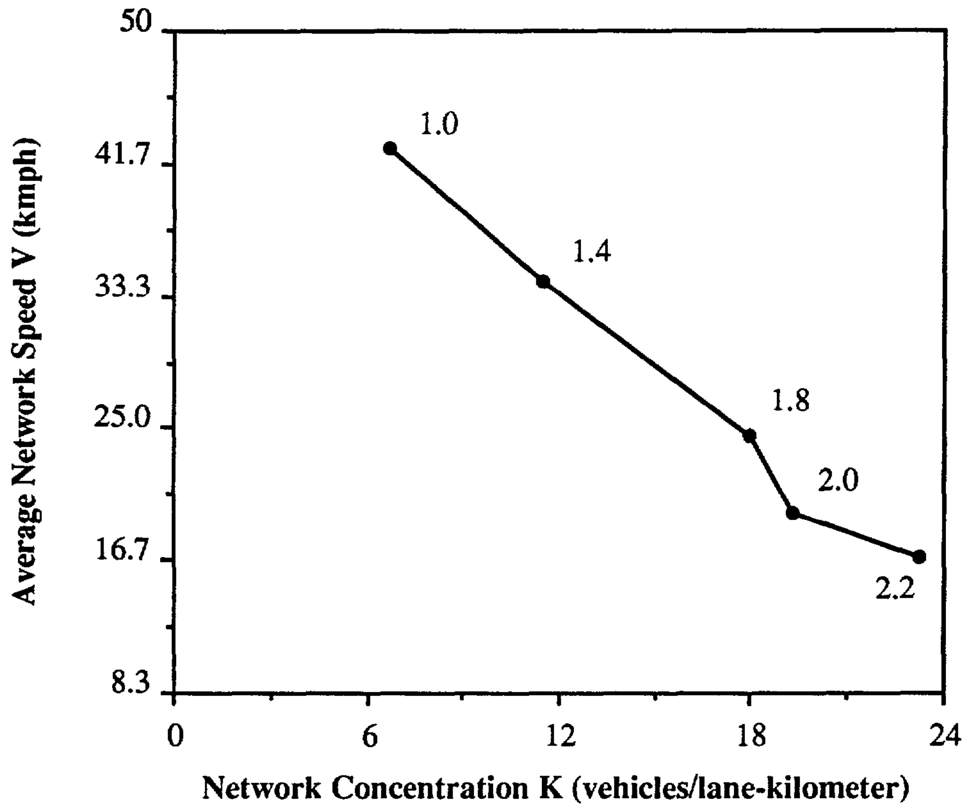


Figure 40. Average Network Speed V (kmph) as a Function of Average Network Concentration K (vehicles/lane-kilometer) for the UE Loading Profile II Case (averaging is done over space and time)

The principal new element examined here arises from the time-varying nature of the vehicular concentration in the network. Figure 41 plots the 5-minute averages of network speed as a function of the corresponding 5-min average network concentrations for the duration of interest, as obtained from the SO solution of loading profile II for loading factor 1.8, used here for illustrative purposes. The numbers 5, 10, 15 ....etc. in the figure represent the 5 minute period starting at those times, and correspond to the points on the plot. The same decreasing trend between speed and concentration is exhibited by the individual short intervals. However, the dynamic nature of the process appears to lead to two different phases: The upper layer of points reflect the first 25 minutes of the state when the network congestion is building up. After the network reaches a certain congested level, the speed-concentration relationship follows the lower layer (which represents the period of time from 25 minutes to 50 minutes). This two-phase phenomenon is worthy of additional theoretical and empirical investigation of the underlying traffic phenomena.

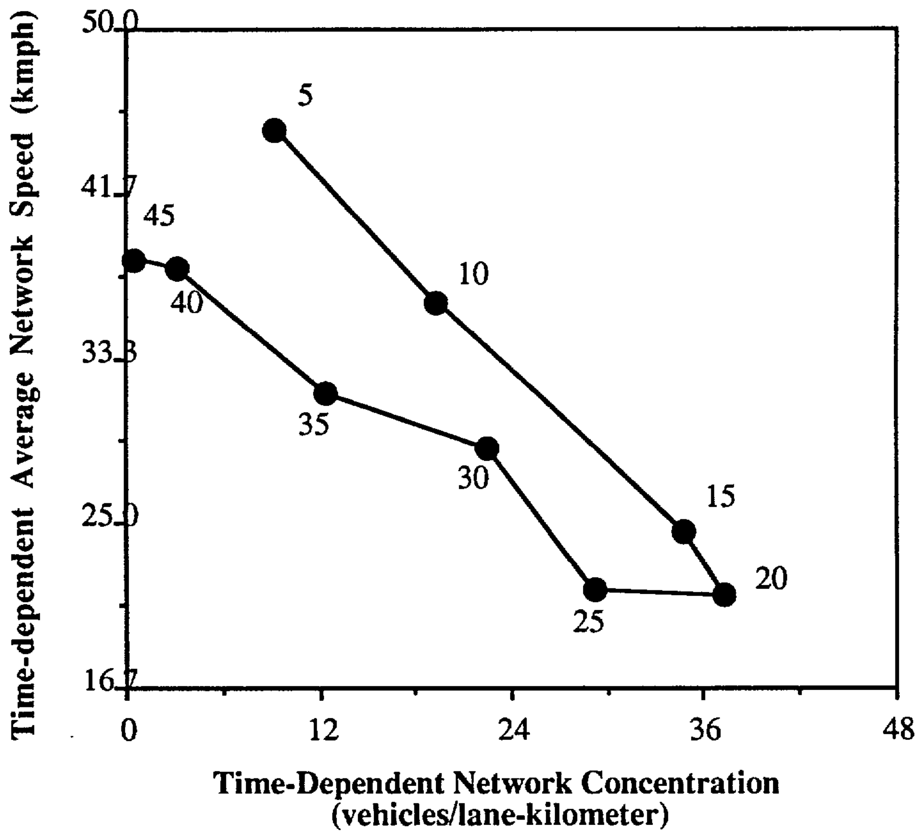


Figure 41. The Time-Dependent Average Network Speed as a Function of the Time-Dependent Average Concentration, Based on 5 minute Average Values for the SO Loading Profile II with LF = 1.8

CONCLUDING COMMENTS

The evaluation of network performance under real-time information systems for electronic route guidance has been confined to highly idealized network configurations and limiting assumptions about various aspects of the problem, particularly user behavior and traffic flow interactions. Theories of network performance under dynamically varying traffic loads and real-time information availability to users are still in their early stages of development, and methodologies to analyze performance of general networks under such conditions are not available. The complexity of the problem arises from the spatial and temporal interactions among individual tripmaker decisions, in response to the supplied information, taking place in the traffic network. This degree of complexity has precluded meaningful analytic treatment of the problem in general networks. Until advances in theory and computation succeed in resolving the formidable difficulties of the problem, computer simulation provides a powerful alternative to analyze the time-dependent performance of traffic networks under a variety of conditions and assumptions regarding user behavior, market penetration and other elements that are subject to external uncertainty or correspond to system design parameters. By providing the analyst with a high degree of experimental control, systematic investigation of network performance and its determinants can be undertaken under a wider range of scenarios than are practically available for observation.

The experiments performed using the simulation-based algorithm to solve both the SO and UE versions of the time-dependent traffic assignment problem have provided insights of critical importance to the design of ATIS information supply strategies and results of fundamental significance in the context of network assignment and network traffic flow theories. The experimental results proffer an illustration of the insights that can be obtained on the basic constitution of the problems being addressed while suggesting directions for future research. The first main conclusion is that the results suggest meaningful differences in overall system cost and performance between time-dependent system optimal and user equilibrium assignments. The second main conclusion is that traffic networks under time-dependent traffic assignment patterns continue to operate within the envelope of relatively simple network traffic flow relationships that exhibit strong similarities to the traffic models established for individual road sections.

If we take the UE assignment results as somehow indicative of the situation that might be attained over time in a system where drivers have access to real-time on-board descriptive information through ATIS, the results of our experiments suggest that there is considerable potential for system optimal, coordinated route guidance, especially in heavily congested (though not oversaturated) networks. These results appear to contradict unsupported claims that descriptive information would likely perform as well as normative SO route guidance because UE system costs were claimed to be very close to SO costs. Instead, they strengthen previous recommendations (e.g., in Mahmassani and Jayakrishnan, 1991) that coordinated information is necessary beyond a certain market penetration level.

The results further highlight the dynamic nature of the benefits accumulated by a SO assignment over UE. They suggest that SO is most effective when the traffic network is moderately to highly congested. In the context of peak period traffic, this implies that most savings through SO assignment would be achieved not at the beginning nor end of the peak period, but in a time range in between. When the network is lightly or very highly congested (oversaturated), an SO assignment does not perform significantly better than UE. For relatively uncongested traffic situations, SO and UE yield almost identical solutions.

The results indicate remarkable consistency with previous observational and simulation results on the relations among network-wide traffic flow variables. In particular, network-level averages of speed and concentration are related in the familiar pattern, though this relationship is dependent on the underlying assignment rule and time-dependent loading patterns. An interesting phenomenon in the dynamics of traffic network performance was illustrated when examining the time-dependent variation of speed with concentration, whereby two distinct phases were apparent reflecting the evolution of the loading process and congestion over the network. This phenomenon is worthy of additional theoretical as well as observational investigation.

With regard to the effectiveness of real-time information systems, the results highlighted the high degree of dependence of the potential benefits of uncoordinated descriptive information strategies on user behavior and the “initial conditions” under which the network is used (i.e. the flow pattern upon which the information supply system is superimposed). The meaningful differences between the SO solution and the UE solution highlighted the potential benefits of coordinated route guidance, particularly as the overall concentration increases in the network. However, it can be expected that as congestion approaches saturation conditions, the advantage of SO over UE would tend to disappear as opportunities for improvement become more limited.

Of course, to the extent that these results are based on simulations using essentially the same network topology, they must be treated as primarily illustrative and suggestive rather than definitive. Considerable additional numerical and observational work is required for this purpose. However, only now are the necessary methodological tools available to support such investigations.

Finally, the experiments serve to demonstrate the successful implementation of the solution concepts developed for dynamic traffic assignment in the ATIS/ATMS context. The descriptive simulation-assignment framework, DYNASMART, provides a very useful tool to evaluate traffic patterns under real-time information. The algorithms developed for SO and UE assignment are also operational, and produce results that successfully pass engineering judgment reasonability tests. They also demonstrate the potential of coordination in the provision of route guidance, and provide a practically tractable basis for the kind of normative traffic assignment capability required by the ATIS/ATMS controller.

CHAPTER 5
MULTIPLE USER CLASSES DYNAMIC TRAFFIC ASSIGNMENT AND REAL-TIME IMPLEMENTATION

The single user class dynamic traffic assignment problem was introduced in the previous chapter and solved under the ideal full information availability scenario (for the controller). This chapter extends the single user class problem to include the various user characteristics and capabilities encountered in the real-world, giving rise to the multiple user classes dynamic traffic assignment problem. In actual situations, the controller may not have complete information on origin-destination (O-D) trip desires for the entire duration of interest. As discussed in the previous chapter, this leads to a partial information availability scenario. This issue is addressed by using a rolling horizon approach to implement the multiple user classes algorithm in real-time to realistic networks.

The first part of this chapter addresses the multiple user classes (MUC) problem. It introduces the user classes of interest and discusses the problem formulation. The solution algorithm for this problem is obtained by extending the single user class solution procedure. The chapter further addresses the rolling horizon approach for quasi real-time implementation of the solution algorithm for the MUC problem. After discussion of real-time implementation issues, and description of the rolling horizon approach, the implementation of this approach to the MUC solution algorithm is illustrated.

THE MULTIPLE USER CLASSES DYNAMIC TRAFFIC ASSIGNMENT PROBLEM

Introduction

A number of factors influence actual system performance under ATIS/ATMS. The fraction of users with capability for one-way or two-way communication (market penetration) with a central controller, the various information supply strategies or assignment rules, and the user response behavior to supplied information are critical determinants of the particular dynamic assignment strategy in any realistic scenario for implementing ATIS/ATMS. As noted previously, an ideal scenario from the controller's perspective is one where all users are equipped, are provided route guidance instructions based on a system optimal strategy, and comply fully with the supplied information, thereby extracting the best possible performance from the system. However, real world conditions may differ significantly from this optimistic scenario, especially in terms of market penetration, the types of information users have access to, and actual user behavior. Hence, any methodology employed to address the problem in the real-world implementation context should account for the multiple user classes with varying characteristics and capabilities in the traffic system.

Even under the most optimistic scenarios for ATIS market penetration over the next decade, only a fraction of all vehicles in a network are expected to be equipped with in-vehicle route-guidance systems. Furthermore, equipped vehicles may possess different capabilities, or have access to different types of information. Drivers may also respond differently to the supplied information, with some drivers complying with the prescribed or suggested routes, others making their own decisions based on information on current or predicted conditions, and yet others behaving in a contrarian manner. Practical considerations such as these form the basis for classifying users into multiple groups, each of which has distinct characteristics in the context of implementing ATIS.

The multiple user classes (MUC) dynamic traffic assignment problem addressed considers the problem faced by a central controller seeking to optimize overall network performance through the provision of real-time routing information to equipped motorists, taking into account different user classes in terms of information availability, information supply strategy, and driver response behavior. In particular, four user classes are incorporated in the formulation: (1) equipped drivers who follow prescribed system optimal paths; (2) equipped drivers who follow user optimum routes; (3) equipped drivers who follow a boundedly-rational switching rule in response to descriptive information on prevailing conditions (e.g. similar to AUTOGUIDE); and (4) non-equipped drivers who follow externally specified paths, which may be historically known or solved for exogenously. Given the time-dependent O-D desires for users in each of these four classes, the formulation seeks a time-dependent traffic assignment which provides the number of vehicles of each class on the network links and paths satisfying system-wide objectives and respective conditions for each class.

Note that our framework also recognizes multiple user classes in terms of traffic performances characteristics, for example trucks versus passenger cars. However, such classes do not have direct implications on the solution algorithm, as associated differences are captured at the level of the traffic simulator. As explained in Chapter 2, DYNASMART recognizes several vehicle types, and correctly represents their respective interactions with other vehicle classes. However, such traffic performance characteristics have no direct implication on the problem formulation or solution procedure, unlike classes that differ in terms of information supply and response behavior. Also note that the four classes incorporated in the present formulation represent the four generic classes of users that have significant implications on the solution framework. In other words, while other classes could be envisioned, the manner in which they are represented in the formulation and their implication for the solution methodology will most likely fall into one of the above four categories or combinations thereof. For example, the third class of users, those who follow behavioral rules in response to descriptive information, could actually consist of several subclasses, each following a different set of behavioral rules or receiving different information. Such subclasses would be treated in essentially the same fashion as the current third class, with only minor differences in implementation details.

Formulation of the Problem

The problem formulated here represents the fully informed situation, where the central controller has complete a priori information about every tripmaker in terms of origin, destination, start time of the trip, and user class, and uses this information to develop an integrated scheme that assigns to each user a path to the destination so as to achieve system-wide objectives as well as the conditions corresponding to the behavioral characteristics of each user class.

Problem Statement

Consider a traffic network represented by a directed graph G(N, A) where N is the set of nodes and A the set of directed arcs. A node can represent a trip origin, a destination and/or a junction of physical links. We consider a network with multiple origins and destinations. The time experienced by a vehicle to traverse a given link depends on the interactions taking place among vehicles in the traffic stream along this arc. The analysis period of interest, taken here as the peak period, is discretized into small equal intervals $t = 1,..........., T$. Given a set of time-dependent O-D vehicle trip desires for the entire duration of the peak period, expressed as the number of vehicle trips r_{ij}^{tu} of user class u leaving node i for node j in time slice t, $\forall\ i, j \in N$, $t = 1,..........., T$, and $u = 1,...., U$, determine a time-dependent assignment of vehicles to network paths and corresponding arcs. In other words, find the number of vehicles $r_{ijk(u)}^{tu}$ of user class u that follow path $k(u) = 1,.........., K_{ij}^{u}$ between i and j at time t, $\forall\ i, j \in N$, $t = 1,..........., T$, and $u = 1,..., U$, as well as the associated numbers of vehicles on each arc $a \in A$ over time. Here, $u = 1,...., 4$ corresponds to the four user classes described in the previous section.

Definition of Variables and Notation

The following variables and notation are used in the formulation of the problem:

i = subscript for origin node

j = subscript for destination node

n = node in the network, $n \in N$

a = arc (or link) in the network, $a \in A$

u = subscript for user class, $u \in U$

k(u) = subscript for a path in the network for user class u

τ = subscript denoting the time interval in which assignment is made

t = subscript denoting current time interval

Δ = length of a time interval

T' = total duration (peak period) for which assignment is to be made

$r_{ij}^{\tau u}$ = number of vehicles of user class u who wish to depart from i to j in period τ

$r_{ijk(u)}^{\tau u}$ = number of vehicles of user class u who wish to depart from i to j in period τ assigned to path k(u)

$\delta_{ijk(u)}^{\tau tau}$ = dynamic arc-path incidence indicator, equal to 1 if vehicles of class u going from i to j assigned to path k(u) at time τ are on link a in period t, i.e.

[$\delta_{ijk(u)}^{\tau tau}$ = 1, if $r_{ijk(u)}^{\tau u}$ is on arc a during period t

= 0, if arc a does not belong to path k(u)

= 0, if $\tau > t$

= 0, if $r_{ijk(u)}^{\tau u}$ is not on arc a during period t]

$T_{ijk(u)}^{\tau u}$ = path travel time for vehicles of user class u going from i to j assigned to path k(u) at time τ

$x_{ijk(u)}^{\tau tau}$ = number of vehicles (i to j) of user class u assigned to path k(u) in period τ which are on link a at the beginning of period t

$d_{ijk(u)}^{\tau tau}$ = number of vehicles (i to j) of user class u assigned to path k(u) in period τ which enter arc a in period t

$m_{ijk(u)}^{\tau tau}$ = number of vehicles (i to j) of user class u assigned to path k(u) in period τ which exit link a in period t

x^{ta} = total number of vehicles on link a at the beginning of period t

d^{ta} = total number of vehicles which enter link a in period t

m^{ta} = total number of vehicles which exit link a in period t

$C(n)$ = set of links directed towards node n
$B(n)$ = set of links directed away from node n

Deterministic Full Information Scenario

This formulation extends the single class full information scenario, in which the central controller is assumed to have complete a priori information on O-D desires for the entire duration of interest. Again, this is mostly a conceptual formulation, where the mathematical expressions are introduced for conceptual clarity. It is not a complete formulation from a mathematical standpoint as it is not sufficient for the development of a complete solution algorithm.

Given:

$r_{ij}^{\tau u}$, $\forall$ i, j, $\tau = 1, \ldots\ldots, T'$, and $u = 1, \ldots\ldots, U$

Objective function:

Min. $\Sigma_\tau \Sigma_i \Sigma_j \Sigma_u \Sigma_{k(u)} (r_{ijk(u)}^{\tau u} . T_{ijk(u)}^{\tau u})$

or

Min. [$T(r_{ijk(u)}^{\tau u})$, $\forall$ i, j, τ, u, k(u)]

Subject to:

1. $r_{ij}^{\tau u} = \Sigma_{k(u)}\ r_{ijk(u)}^{\tau u}$, $\forall$ i, j, τ, u
2. $\Sigma_c m^{tc} = \Sigma_b d^{tb}$, $\forall$ t, $c \in C(n)$, $b \in B(n)$, $n \neq i$ or j
3. $x^{ta} = x^{t-1a} + d^{t-1a} - m^{t-1a}$, $\forall$ t, a
4. $x^{ta} = \Sigma_u \Sigma_{k(u)} \Sigma_\tau \Sigma_i \Sigma_j [r_{ijk(u)}^{\tau u} . \delta_{ijk(u)}^{\tau tau}]$, $\forall$ t, a
5. $T_{ijk(u)}^{\tau u} = \Sigma_t \Sigma_a [\delta_{ijk(u)}^{\tau tau} . \Delta]$, $\forall$ i, j, τ, u, k(u)
6. $\delta_{ijk(u)}^{\tau tau} = f(r_{ijk(u)}^{\tau u})$, $\forall$ i, j, τ, t, a, u, k(u)
7. $d^{ta} = \Sigma_k \Sigma_\tau \Sigma_i \Sigma_j\ d_{ijk(u)}^{\tau u}$, $\forall$ t, a
8. $m^{ta} = \Sigma_k \Sigma_\tau \Sigma_i \Sigma_j\ m_{ijk(u)}^{\tau u}$, $\forall$ t, a
9. $\tau \leq t$
10. $\delta_{ijk(u)}^{\tau tau} = 0$ or 1
11. $\{r_{ijk(2)}^{\tau 2}$, $\forall$ i, j, k, τ } satisfy user equilibrium conditions.
12. $\{r_{ijk(3)}^{\tau 3}$, $\forall$ i, j, k, τ } are consistent with the boundedly rational rules specified in DYNASMART

13. $\{r_{ijk(4)}^{\tau 4}\}$ are known a priori $\forall$ i, j, k, τ

14. All variables (other than $\delta_{ijk(u)}^{\tau tau}$) ≥ 0

There are two alternative forms for the objective function in the above formulation. The first states that the total travel time of the assigned vehicles in the system is aggregate of the product of the number of vehicles of each user class assigned to a particular path (from a given origin to a given destination at a particular time) and the corresponding path travel time. The assumption on identical travel experience is realistic when assignment intervals are reasonably small (in which case there are not more than two or three vehicles to a particular path from an origin to a destination). The nonlinearity of the objective function arises from the fact that the travel time on the path is itself a complicated non-explicit function of the number of vehicles assigned to the various paths of the network, via the dynamic link-path incidence.

The second form of the objective function simply states that the total travel time of all vehicles assigned to the various paths during the duration of ATIS application is some function of the assignment. This objective function can be evaluated by any available means. We do it through simulation.

Constraint (1) is a definitional constraint stating that O-D desires assigned to the various paths should sum up to the demand (conservation at the origin). Constraint (2) states that vehicles cannot be stored at intermediate nodes, that is, the number of vehicles exiting from all links incident on an intermediate node should equal the number of vehicles entering all links incident from that node at any given time. Constraint (3) represents the conservation of vehicles on a link and states that the total number of vehicles on any link at the end of the current time interval is the net algebraic sum of vehicles on that link at the end of the previous time period, vehicles entering that link during the current period and vehicles exiting that link during the current period.

Constraints (4), (5) and (6) represent the time-dependent link-path incidence relationships which fundamentally characterize the dynamic assignment problem. Constraint (4) represents the time-dependent relationship between the number of vehicles assigned to various paths and their aggregation on links. Constraint (5) illustrates the calculation of the path travel times using the dynamic link-path incidence variables. The number of time steps in which the dynamic incidence variable takes a value 1 implies the number of discrete time steps that a vehicle (or a group of vehicles) spent in the system, and multiplying with Δ gives the actual travel time in the system. One of the most commonly used indicators of system performance is the total time spent by vehicles in the system, and the path travel times conveniently allow evaluation of this indicator.

Constraint (6) states that the dynamic link-path incidence variables are a function of the assignment. As noted, this fundamental fact expresses the essence of the dynamic assignment problem. Constraints (7) and (8) are definitional constraints for the number of vehicles entering and exiting links in the various time intervals. Constraint (9) defines temporal correctness. Constraint (10) restricts the dynamic incidence variables to take values of 0 or 1. Constraints (1 l- 13) reflect the conditions characterizing the behavior of user classes 2, 3 and 4. While it is possible to express constraints (11) as variational inequalities, constraints (12) would be more cumbersome to write mathematically because they correspond to rules embedded in the DYNASMART simulator. The authors have provided elsewhere a mathematically more elaborate formulation but it does not contribute directly to the problem solution. Constraint (14) represents the non-negativity requirement.

Solution Algorithm

This section illustrates the simulation-based solution algorithm for the MUC dynamic assignment problem, obtained by extending the single user class solution algorithm described in the previous chapter. The DYNASMART simulation model is used to evaluate any particular assignment pattern and provide the relevant information necessary to guide the search to the solution satisfying the desired conditions. Figure 42 illustrates the solution algorithm for the MUC time-dependent assignment problem. Analogous to the algorithmic steps for the single user class assignment problem, the simulation results from the current iteration provide the basis for a direction finding mechanism for the search process, through the experienced vehicular trip times and the associated marginal trip times.

The algorithmic steps of the search process embedded in the algorithm are illustrated in Figure 43. It consists of an inner loop that incorporates a direction finding mechanism for the search process for the SO and UE user classes based on the simulation results of the current iteration. Convergence is sought by obtaining search directions for the SO (user class 1) and UE (user class 2) components of the solution for the next iteration. The class of (equipped) users that follow behavioral rules in response to descriptive information based on current traffic conditions (user class 3 or BR) is not directly involved in the direction finding mechanism of the search process. The paths of this class of users are obtained based on the traffic pattern that evolves in the network for the current assignment strategy (and the behavioral rules assumed), unlike classes 1 and 2 which obtain their paths based on search directions derived from the experience of previous iterations. Hence, from an algorithmic standpoint there is no direct guiding mechanism involved in obtaining the paths for user class 3, other than their being predicated on the assignment strategy for the SO and UE class vehicles. As illustrated in the figure, they form the outer loop of the iterative procedure. They may aid or impede, over different iterations, the progress towards convergence of the algorithm. As indicated, the unequipped users (user class 4 or PS) are exogenous to the iterative loop of the search process and represent constant background information (for each iteration) as their paths remain unchanged.

Figures 42 and 43 depict the solution algorithm for the MUC assignment problem. A brief summary of the approach is as follows:

1. Set the iteration counter I = 0. Obtain the time-dependent historical paths (paths obtained from database) for all equipped user classes for each assignment time step over the entire duration for which assignment is sought, as well as the paths to be assigned to the unequipped users.

2. Assign the O-D desires (which are known a priori for the entire peak period) for the entire duration to the given paths and simulate the traffic patterns that results from the assignment using DYNASMART. The path for each vehicle in user class 3 ($r_{ijk}^{\tau,BR,I,+1}$) is obtained from the current simulation experience.

3. Compute the marginal travel times ($mltt^{ta,I}$) on links using time-dependent experienced or estimated link travel times ($tt^{ta,I}$) and the number of vehicles ($x^{ta,I}$) on links obtained as post-simulation data (from step 2).

4. Using a special-purpose time-dependent multiple user classes least cost path algorithm, compute the least marginal time paths (based on the marginal travel times obtained in step 3) and the shortest travel time paths for each O-D pair for each assignment time step.

5. Perform an all-or-nothing assignment of the user class 1 (SO) O-D desires to the least marginal time paths computed in the previous step, and of user class 2 (UE) O-D desires to the shortest travel time paths. The result is a set of auxiliary path vehicle numbers for each O-D pair for each assignment time step t = 1,............, T, for the SO and UE ($y_{ijk}^{\tau,UE,I}$) user classes.

6. Update paths and the number of users assigned to those paths, for the SO ($r_{ijk}^{\tau, SO, I+1}$)

and UE ($r_{ijk}^{\tau, UE, I+1}$) user classes. Update of paths is done by checking if the path identified in step 4 already exists (i.e., has carried vehicles in at least one prior iteration) for that O-D pair and including it if it does not. The update of the number of vehicles (assignment of vehicles to the various paths currently defined between the O-D pair after the path update) is performed using the Method of Successive Averages (MSA), which takes a convex combination of the current path and corresponding auxiliary path numbers of vehicles, for each O-D pair and each time step. A detailed description of MSA is provided in Sheffi and Powell (1982). Note that other convex combination schemes could equally be used.

7. Check for convergence using an $\in$-convergence criterion (currently, we use the convergence of the path vehicle numbers for both SO and UE user classes as the criterion).

8. If convergence criterion is satisfied, stop the program. Otherwise, update the iteration counter I = I + 1 and go to step 2 with the updated data on paths and the number of vehicles assigned to each of those paths for the SO and UE user classes; and with the paths obtained from step 2 of the current iteration as the initial paths for vehicles of user class 3 for the next iteration. The paths for vehicles of user class 4 are constant and are carried over unchanged to the next iteration (as illustrated by their exogenous assignment to the iterative loop).

ROLLING HORIZON FRAMEWORK FOR REAL-TIME IMPLEMENTATION

Introduction

The solution algorithms discussed so far for "complete information" formulations are intended as the basic core methodologies to be implemented in a dynamic assignment framework for real-time operation. Rather than assume that O-D desires are known a priori for the entire planning horizon, a more realistic scenario is one where the controller is assumed to have information on O-D desires for only a "short" duration into the future with a high degree of confidence. This partial information availability scenario is formulated using a rolling horizon approach which is well-suited for on-line implementation of the solution algorithms developed for the single and multiple user classes problems.

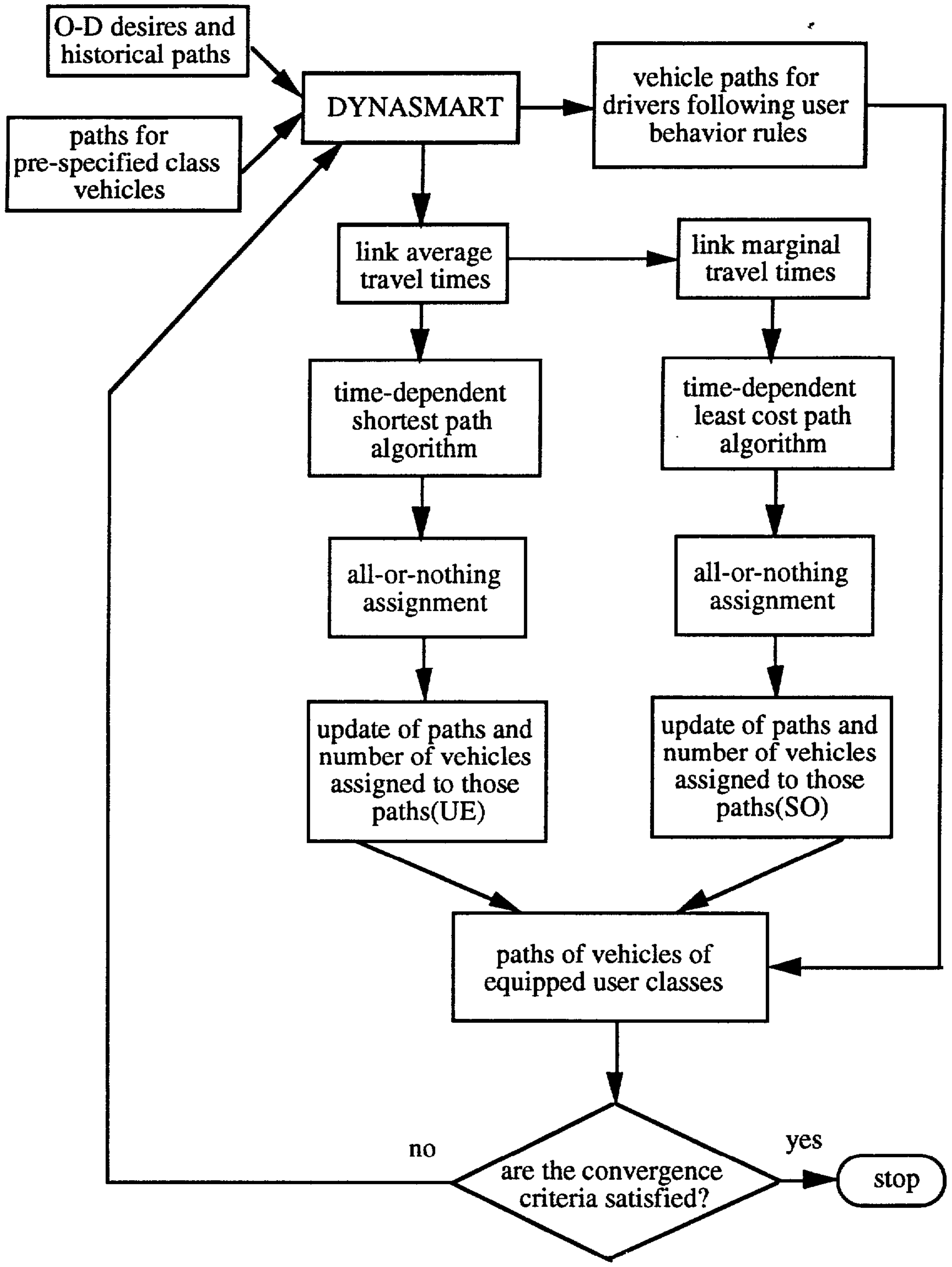

O-D desires and historical paths
paths for pre-specified class vehicles
DYNASMART
vehicle paths for drivers following user behavior rules
link average travel times
link marginal travel times
time-dependent shortest path algorithm
time-dependent least cost path algorithm
all-or-nothing assignment
all-or-nothing assignment
update of paths and number of vehicles assigned to those paths(UE)
update of paths and number of vehicles assigned to those paths(SO)
paths of vehicles of equipped user classes
no
are the convergence criteria satisfied?
yes
stop

Figure 42. Multiple User Classes Solution Algorithm

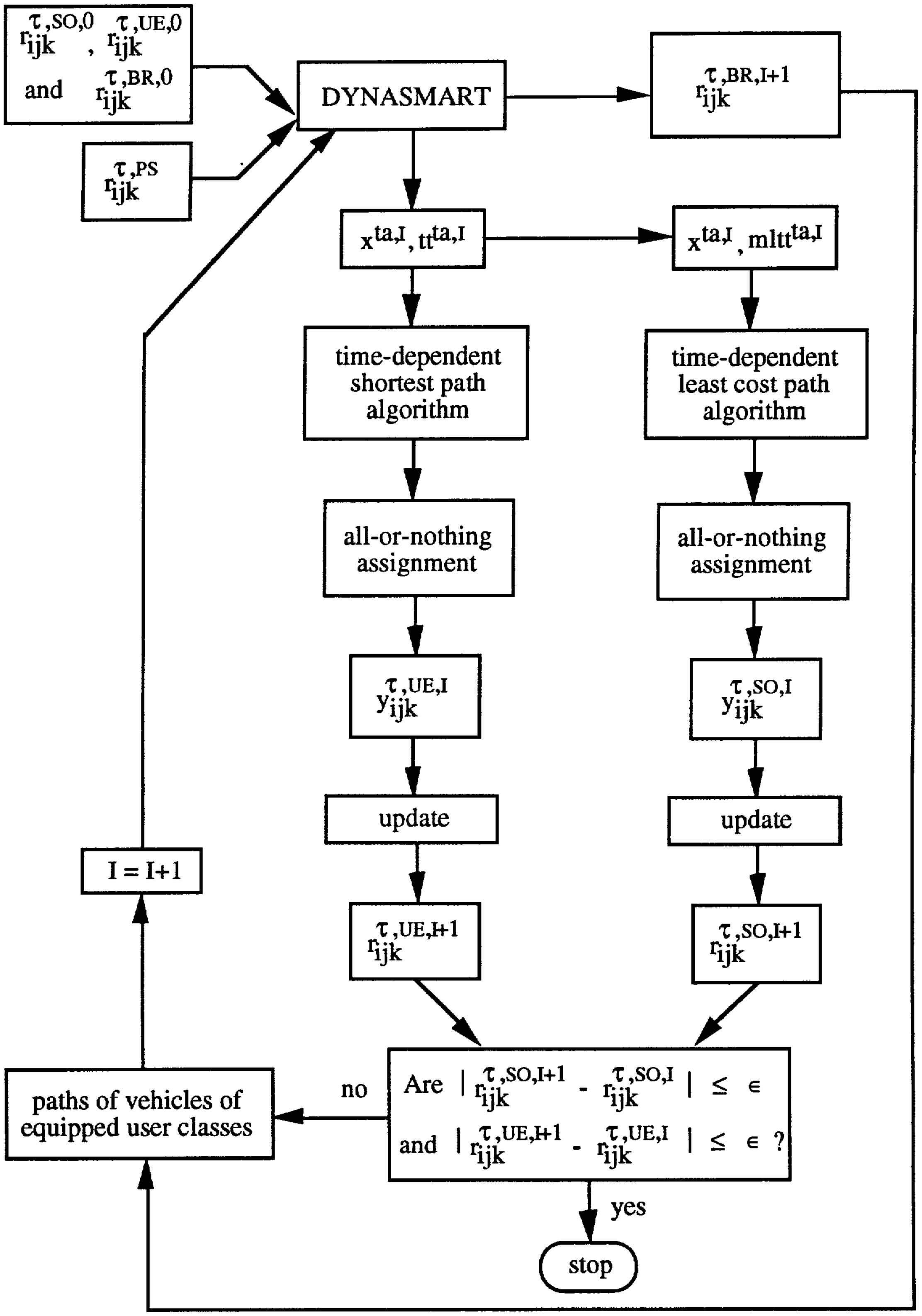
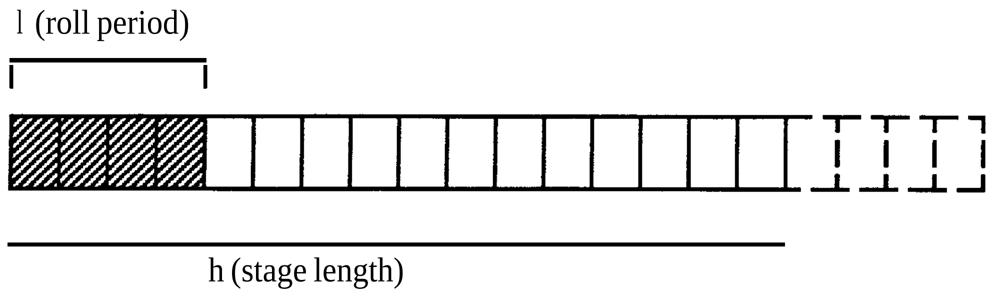

$r_{ijk}^{\tau,SO,0}$, $r_{ijk}^{\tau,UE,0}$ and $r_{ijk}^{\tau,BR,0}$
$r_{ijk}^{\tau,PS}$
DYNASMART
$r_{ijk}^{\tau,BR,I+1}$
$x^{ta,I}, tt^{ta,I}$
$x^{ta,I}, mltt^{ta,I}$
time-dependent shortest path algorithm
time-dependent least cost path algorithm
all-or-nothing assignment
all-or-nothing assignment
$y_{ijk}^{\tau,UE,I}$
$y_{ijk}^{\tau,SO,I}$
update
update
I = I+1
$r_{ijk}^{\tau,UE,I+1}$
$r_{ijk}^{\tau,SO,I+1}$
paths of vehicles of equipped user classes
no
Are $| r_{ijk}^{\tau,SO,I+1} - r_{ijk}^{\tau,SO,I} | \leq \epsilon$ and $| r_{ijk}^{\tau,UE,I+1} - r_{ijk}^{\tau,UE,I} | \leq \epsilon$?
yes
stop

Figure 43. Algorithmic Steps of the Solution Procedure

Rolling Horizon Approach

The basic idea behind the rolling horizon approach is that current events will not be influenced by events "far" into the future. In the context of the ATIS problem, this is analogous to stating that vehicles currently assigned will not be influenced by vehicles assigned "far" into the future as the currently assigned vehicles will probably be out of the system by that time. The stage length h in Figure 44 depicts that length of time (its value in actual problems is network specific). The roll period l represents the short duration into the future for which O-D desires are available with reasonable reliability. To make an assignment of vehicles to various paths for the current period, the controller requires knowledge of O-D desires for the rest of the stage length as these O-D desires are expected to influence current assignments. These O-D desires may be forecasted based on historical data and current information. The O-D desires beyond the stage length h are assumed to be zero. The situation is now analogous to the complete information availability scenario, albeit, only for the duration covered by the stage length h. The system is solved for optimality only for the duration of the stage length and O-D desires for the roll period (which are known with certainty) are assigned to the paths determined. The time frame is now "rolled" forward by a length equal to the roll period and the above process is repeated till the end of the duration for which ATIS is applied to the system. Hence, a series of optimizations are performed till the planning horizon is covered.

The path assignments in each stage are determined for the entire stage, but implemented for only the roll period (as only the demand for this period is available with certainty). A number of pertinent questions arise at this point. How far is "far"? What is the "optimal" stage length h? What is a good value for the roll period l? How accurate are the forecasted values for future O-D desires? Is there a need for feedback to check if the assumptions made were realistic? How robust is the solution vis-a-vis the predicted O-D desires. These questions need to be addressed while implementing the solution methodology for the rolling horizon framework. The values of the various parameters are expected to be problem specific; however, appropriate guidelines can be developed through numerical experiments and test applications. This approach also emphasizes the need for compatible O-D demand forecasting models.



l = The roll period(in number of time steps)

h = The stage length(in number of time steps)

nl = The current stage number

Figure 44. The Rolling Horizon Approach

Real-Time Implementation

Figure 45 illustrates the rolling horizon framework to implement the solution algorithm for the MUC dynamic assignment problem in real-time. The procedure is as follows:

1. Obtain the O-D desires for the first roll period, and the forecasted O-D desires for the rest of the first stage. Obtain the time-dependent historical paths (paths obtained from database) for equipped user classes for each assignment time step in the first stage, as well as the paths to be assigned to the unequipped users.
2. Perform a complete run of the MUC algorithm till convergence for the O-D desires of the current stage. Make an on-line assignment of the O-D desires for the current roll period to paths to their destinations. If the end of the time horizon for which an ATIS is desired is reached, stop. Otherwise, continue (go to step 3).
3. Shift the current stage by a length of time equal to the roll period to obtain the next stage. The next stage now becomes the current stage.
4. Update the origins and positions of all vehicles from the previous stage that did not reach their destinations by the end of that stage.
5. Update the O-D matrix for the current stage based on O-D desires for the current roll period, forecasts for the O-D desires for the rest of the current stage, and O-D desires from the previous stage who have not yet reached their final destinations. If re-routing of previously assigned vehicles is not desired, treat vehicles already in the network as members of class 4, who follow pre-specified paths (in this case obtained from the previous stage). If re-routing is desired, keep already assigned vehicles in their respective classes and update their origin to the current location (or appropriate downstream node).

6. Obtain the initial paths for the current stage based on historical data and the link travel time experience from the previous stage. Go to step 2.

In the on-line implementation of the procedure, there are three possible scenarios regarding the relative magnitudes of actual and forecast O-D trip desires. The ideal case is when the forecasts are accurate for the entire stage under consideration. The vehicle paths obtained in the solution for the roll period are then as good as one could expect them, unless changes take place in the network characteristics, such as a capacity-reducing accident (in which case the problem should be re-solved on-line to determine new paths for equipped classes in conjunction with applicable traffic management strategies).

However, forecasts may also either underestimate or overestimate actual demand, corresponding to the second and third scenarios respectively. In both cases, the quality of the paths determined for vehicles generated in the roll period may suffer, though this is inherent in any decision strategy operating under uncertain conditions. The robustness of the solution vis-a-vis the quality of the O-D forecasts, and the implications for the various parameters of the implementation scheme (particularly the lengths of the roll period solution stage and O-D prediction horizons), are undoubtedly the most important issues that need to be addressed in future numerical and observational research. From the standpoint of the actual implementation of the procedure, both underpredication and over-prediction scenarios can be accommodated. In the former (underprediction), there will be fewer actual vehicles than incorporated in the solution. This does not pose any implementation difficulty on-line as all actual O-D trip desires will have paths available to their respective destinations. In the latter scenario (over-prediction), the “additional” actual vehicles will be assigned to paths selected randomly from the optimal path set for their corresponding class. Since user class 3 vehicles make their own decisions in real-time, the rolling horizon procedure serves only the purpose of determining good “initial” paths for them. Finally, user class 4 vehicles do not receive paths from the central controller.

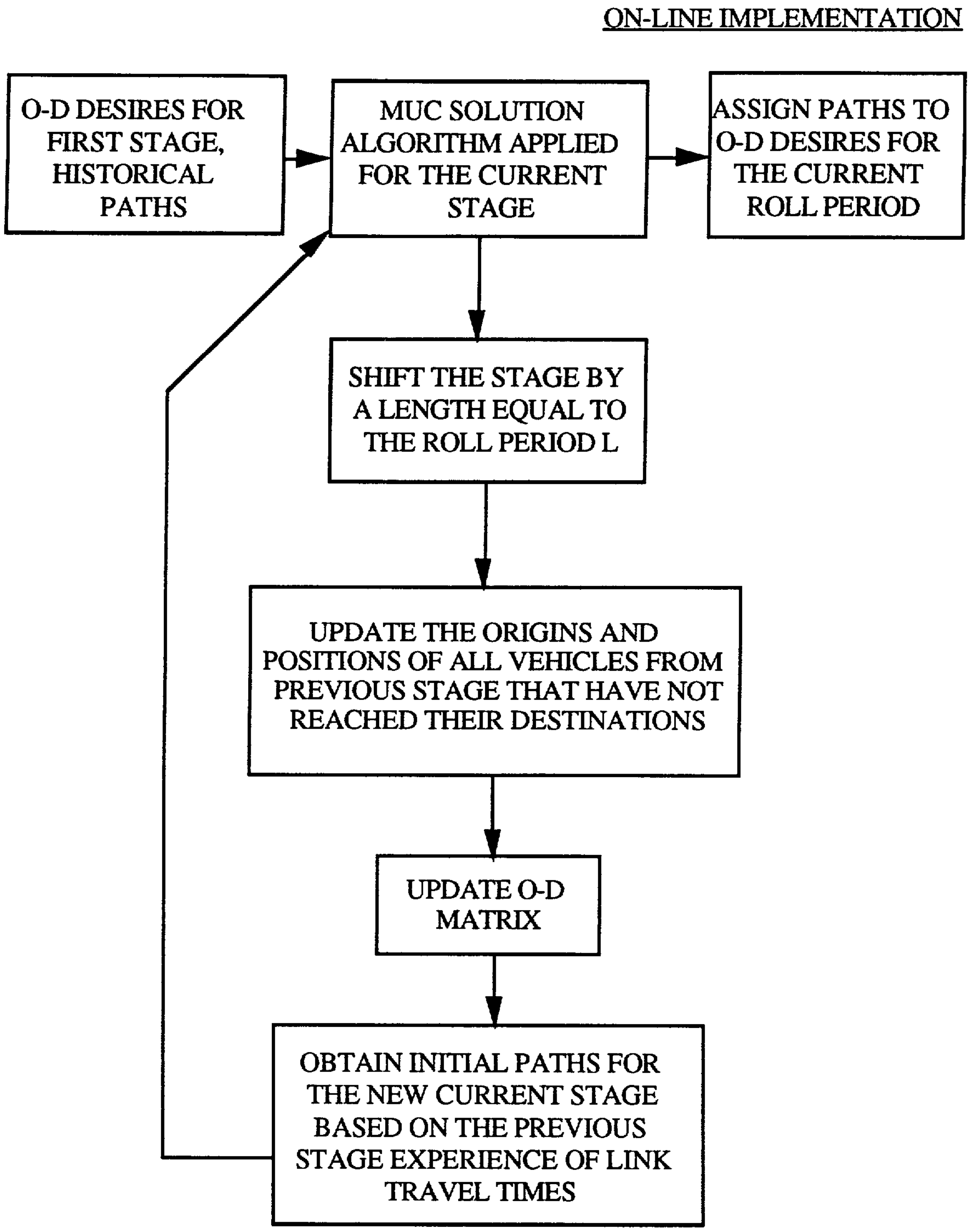

ON-LINE IMPLEMENTATION
O-D DESIRES FOR FIRST STAGE, HISTORICAL PATHS
MUC SOLUTION ALGORITHM APPLIED FOR THE CURRENT STAGE
ASSIGN PATHS TO O-D DESIRES FOR THE CURRENT ROLL PERIOD
SHIFT THE STAGE BY A LENGTH EQUAL TO THE ROLL PERIOD L
UPDATE THE ORIGINS AND POSITIONS OF ALL VEHICLES FROM PREVIOUS STAGE THAT HAVE NOT REACHED THEIR DESTINATIONS
UPDATE O-D MATRIX
OBTAIN INITIAL PATHS FOR THE NEW CURRENT STAGE BASED ON THE PREVIOUS STAGE EXPERIENCE OF LINK TRAVEL TIMES

Figure 45. Real-Time Implementation

CHAPTER 6
THE PATH PROCESSING COMPONENT OF DYNASMART

This chapter describes the algorithmic elements of the path processing component of DYNASMART, namely the k-shortest path computation and update procedures. The time-dependent least time and least cost algorithms used in conjunction with the time-dependent system optimal and user equilibrium assignment procedures described in Chapters 4 and 5 are discussed in the next chapter. This chapter is broadly divided in four sections. The first is an extensive introduction and literature survey section pertaining to k-shortest path algorithms and implementation issues. Next, the k-shortest path and the update path algorithms for a single user class, as implemented in DYNASMART, are analyzed; efficient implementations are discussed, and comparative results are presented for the adopted as well as for alternative designs. The third section extends the previous methodologies to the multiple user classes case. In the final section, “update” versus “calculate” strategies are assessed.

INTRODUCTION AND LITERATURE REVIEW

The need to calculate more than one shortest path (between a given origin and destination) in DYNASMART arises for several reasons including: (1) the need to model vehicle movement patterns under information, (2) the need to give alternative paths to drivers, (3) the capability to generate and represent a realistic choice set for drivers in connection with the user decisions component and finally (4) the need to improve the computational efficiency of the overall path computation component by combining update and calculate strategies.

An extensive literature review was carried out to identify the algorithms that could provide best performance in the DYNASMART environment. For this purpose three algorithms were designed, implemented and actually coded. The algorithms were tested on real street and random networks, and the best of them is included in DYNASMART. The computational results are discussed in the second section.

Terminology and Notation

This section contains formal definitions of the terms commonly used in the shortest path literature. Most of the notation and definitions follow Dial et al’s (Dial, 1978). A directed network or simply a network G(N,A) consists of a finite set of nodes N and a finite set of arcs A. An arc $a \in A$ may be also denoted as ordered pair (i,j), referring to the fact that an arc is conceived as beginning at node $i \in N$, and terminating at node $j \in N$.

A directed path is a finite sequence of nodes $P=\{n_1,n_2,...,n_r\}$, where n_1 is the origin node of the path and n_r is the destination node. A path P is called simple path, if $n_i \neq n_j$ $\forall i,j \in P$ and $i \neq j$. Let C_{ij} denote the nonnegative elements of a cost matrix C for any arc (i,j). We define the cost of the path P as:

$$\lambda_P = \sum_{i=1,j=i+1}^{r-1} C_{n_i n_j}$$

A path P between two nodes is called the shortest path if λ_P is the minimum among all paths between these two nodes. If the procedure that calculates this path is rooted at the origin node, then the cost of the path to node r is called label of the node r and is denoted as λ_r.

Literature Review

Algorithms

A number of methods have been proposed since the late 50's for the solution of the shortest path and k-shortest path problems, after the pioneering work of Moore (1959) and Dijkstra (1959). An excellent review of this early work was done by Dreyfus (1969) and Pollack (1961a). Much work has been also done on implementing the proposed algorithmic ideas, especially that of Dial et al for the single shortest path (Dial et al., 1979), and Shier for the k-shortest path case (Shier, 1974 and 1979).

Two versions of the k-shortest path problem have been investigated over the past thirty years. In the first version, no path is allowed to contain repeated nodes; algorithms for this type of problems have been investigated by several authors, especially Yen (1971a), Lawler (1972, 1976, 1977), Bellman et al (1960), Minieka and Shier (1973), Sakarovitch (1968), Fox (1978), Perk (1986) and possibly others. In the second version, paths are allowed to contain repeated nodes. This study is focused on the second category of k-shortest path algorithms. The loopless approach has been ruled out because it is very demanding computationally, and its computational performance is not expected to be under the acceptable time limits for a real-time system. Especially if repeated nodes do not commonly occur--as usually happens for large street networks--then it is not compelling to use a loopless path algorithm.

According to Dreyfus, the first efficient algorithm that calculates k-shortest paths allowing loops was developed by Hoffman and Pavley (1959). They defined as a deviation from the shortest path, any path that coincides with the shortest path from the origin up to some node j then deviates directly to some node k, then reaches the destination node via the shortest path from k. If all such deviations are computed between two nodes, then the second best path can be determined as the minimum among the set of deviation paths. The algorithm first calculates the shortest path tree from all nodes to a given destination. Next, the deviations from the second path are explored and the third best path is noted among the deviations of the second best path and those remaining in the first deviation set. If the average node has d outgoing links, and the average shortest path contains m links, an average problem requires md steps, beyond those for calculating the shortest path tree, to find the second shortest path. However, the subsequent paths require significantly more computation time to find than the first one.

Dreyfus suggested some modifications that appear to increase the efficiency of this algorithm, but no implementation has been reported so far in the literature. Fox (1973) studied this Dreyfus-Hoffman-Pavley method and proposed data structures for the lists the algorithm needs to keep. Specifically he proposed a heap structure for the list that keeps all the sub-optimal paths for every node. This structure is a binary tree that always keeps its least element in the first position, and requires log2N operations to insert and delete an element in a N-element list. This structure works better when a large number of paths is sought, k>10logN. Fox considers also a perturbation scheme that safely breaks ties. Specifically, he proposes picking a small number b and then generating uniform random numbers in the interval (O,b), and adding a distinct one to each arc cost. This indeed ensures that no ties or zero cost loops will result during the process.

Another early algorithm that is worth noting has been proposed by Pollack (1961). The algorithm can be applied when k is small, and appears especially competitive if only two paths are sought. His method can be described as follows:
Given the shortest path from node i to node j, the length (cost) of each link along this path is set, in turn, to infinity and the shortest path for the resulting network is found. The best of these paths then is the second best path. An interesting feature of this scheme is that all the shortest path calculations, each corresponding to a particular arc set to infinity, are completely independent of each other. This makes this approach a possible candidate for massively parallel computations. Also this procedure could be easily transformed so that time-dependent k-shortest paths can be calculated for a given OD pair.

Another significant contribution that has attracted considerable attention is a paper by Bellman and Kalaba (1960), who proposed an entirely different procedure. If hki is the length of the kth shortest path from node i to destination node N and if $\min_k(x_1,x_2,\ldots,x_n)$ is defined to be the kth minimum element of the set $\chi=(x_1, x_2,\ldots x_n)$ then $\lambda^k{}_i$ is characterized by the following relations:

$$\lambda_i^{k+1} = \min\begin{bmatrix} \min_{k+1}^{j\neq i}(C_{ij}+\lambda_j^k) \\ \min_k^{j\neq i}(C_{ij}+\lambda_j^{k+1}) \end{bmatrix} \qquad (i=1,\cdots,N-1) \quad (1)$$

$$\lambda_N^1 = \min_1^i(C_{iN}+\lambda_i^1)$$

The term $\min_{k+1}(C_{ij}+\lambda^k{}_j)$ determines the value of the kth best path originating at node **i** and deviating from the shortest path at that node **i**. The term $\min_k(C_{ij}+\lambda^{k+1}{}_j)$ evaluates the best path consisting of any first arc, plus the second best continuation. This approach performs better than the previous ones when more than two paths are sought, while Hoffman-Pavley's procedure outperforms for k=2.

In connection with these two algorithms, Dreyfus proposes a procedure that stems from the previous ideas, but seems to be more efficient computationally. However, it is presented only theoretically and no details on implementation are given. Lawler (1976) also discusses this approach and clarifies a few of its aspects; however, he does not present any formal implementation design for it.

The similarity in algebraic structure between the ordinary shortest path and the k-shortest path problems was recognized and studied extensively by Minieka and Shier (Minieka, 1974; Minieka and Shier, 1973; Shier, 1974, 1976 and 1979). They developed a whole class of solution methods for the k-shortest path problem utilizing strategies shown to be efficient for the ordinary shortest path case. In particular, the basic label setting and label correcting schemes for the single path problem were shown to be applicable to the k-path case. Several of these algorithms were implemented and tested for a wide range of network structures by Shier (1979), in what appears to be the most comprehensive implementation work for this class of problems. The main label setting and label correcting schemes used in the present study have been motivated by Shier's work.

There appears to be nothing reported in the literature on computing shortest paths simultaneously for multiple user classes; similarly, the literature is very slim for update path algorithms. A dual simplex algorithm that computes paths for one destination given that the paths to another close destination are known was presented by Florian et al.

(1981). This approach, however, was not found to be directly applicable in the context of this research.

Review of Implementation Aspects

Considerable progress has been made in the past three decades on applying computer science techniques in the design of algorithms, especially in the Operations Research community. Shortest path algorithms are no exception in this respect. Most of the attention has been given to designing data structures that manipulate efficiently the SE list, although network representation, and efficient storage of the shortest path trees have also been studied extensively. Such efforts, in combination with computer hardware evolution, have dramatically improved the computational times for most shortest path approaches (Dial et al., 1979; Fox, 1978; Bertsekas, 1992). In this section, we review the data structures and implementation schemes that are applicable to the k-shortest path case.

An early implementation of a label setting scheme has been proposed by Dial (1969). Its chief feature is the data structure of the SE list. It uses an "address calculation scheme", which is also known as a bucket structure (Figure 46).

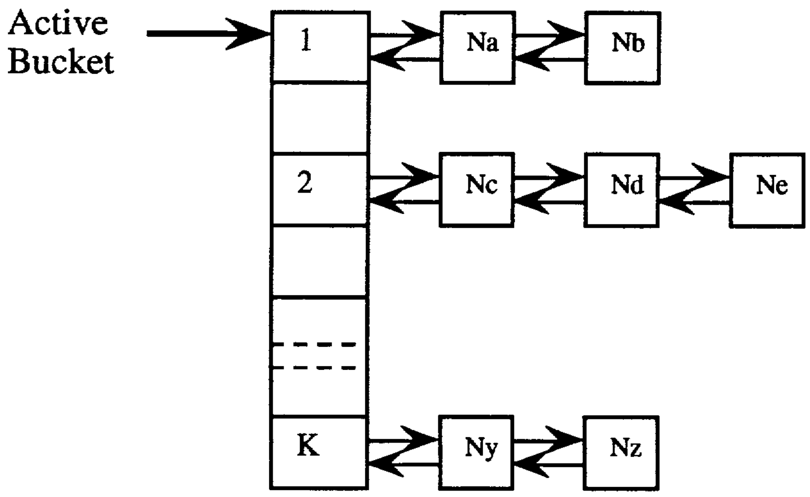


Figure 46. Bucket Structure

Nodes are arranged into the buckets according to the value of their label. Inside a bucket, the nodes are connected by a double linked list and sorted according to their label sizes. If the label of a node is updated, this node is moved to another position in the list, or into another bucket. This can be done in a two-step operation because the double linked list holds the exact position of its elements in the structure. Moreover, this structure always holds the least label node in the first position of the top bucket, which is advantageous for label setting algorithms. The most costly operation for this structure is the node insertion, and therefore its efficiency depends on the number of operations associated with the node insertion. This is in turn depends on the bucket size, the number of buckets, and the variance of the arc cost values. Extensive description of this data structure can be found in Aho et al. (1983). This implementation can be coded using one-dimension array pointers, each pointing to the first node of a linked list. The linked list can be represented by an Nx2 array. This scheme has been implemented by Shier for the k-shortest path case with very encouraging results.

Gilsinn and Witzall (1973) were among the first to test the above scheme for a wide range of networks, and compare it with label correcting schemes. They implemented and coded three label correcting and five label setting algorithms, and performed a number of comparison tests on random networks. They concluded that Dial's label setting approach was performing better than their label correcting designs for most of the networks considered. However, none of the label correcting designs that they coded would be considered efficient today.

Van Vliet (1978) also implemented Dial's algorithm along with four other designs, including one label setting algorithm with a heap data structure for the SE list, and a label correcting with Pape-D'Esopo's double ended queue. Both of these structures are employed in this study for the k-shortest path case. It is worth noting his conclusions about the four factors that affect the efficiency of such structures, namely: network size, mean link cost, network shape and ratio of number of links to number of nodes.

Efficient data structure designs for label correcting SE list has been the subject of extensive research after the introduction of the algorithm by Moore. Pape (1974), Golden (1976), Pallotino (1984), Pallotino and Gal10 (1986) and Dial et al. (1978) among others investigated a number of designs, from simple linked queues, circular lists, stacks, double linked lists, up to double two ended queues and double ended queues. Almost all the mentioned studies performed comparison tests among a number of different structures. Most of these results indicated that the double ended queue (deque) tends to perform better for large sparse networks. This structure is discussed extensively in the implementation section of this chapter.

Finally, the newest family of shortest path algorithms, the PSP algorithms, introduced by Glover et al are implemented in Glover (1985). For the single path case, two simple lists are used without any internal arrangement. The algorithm examines the nodes from the first list in a FIFO fashion. When this list is empty the second list is scanned sequentially from the beginning to the end and the nodes with labels less than a threshold value are inserted at the end of the first list. This simple structure has been proposed and implemented by Glover et al (1985). Transformation of this structure for the k-shortest path case is analyzed and implemented in this study for the first time.

Extensive implementations of k-shortest path schemes, both label setting and correcting, were studied and proposed by Shier (1979). The basic scheme for all of them was described in the algorithms review section. A k-vector label structure was used for every node. Five designs were implemented and tested. The first is a basic label correcting algorithm (BLC), a scheme that scans every node at every iteration; this scheme proved to be very inefficient. Next is an alteration flag (AF) algorithm, a simple array structure, called flag, that distinguishes nodes as eligible or not eligible for scanning. This scheme performed satisfactorily for small sparse networks, but not so well for larger or denser networks. The third approach tested is a variant of the BLC algorithm, a double sweep scheme that alternates forward and backward iterations until no improvement in any of the k-vectors is possible. The last label correcting algorithm tested uses a simple FIFO list for the SE list, implemented as a circular list with two pointers, one to the top and the other to the rail of the list. Nodes are inserted at the tail and deleted from the top of the list. A one-dimension array of size N, similar to that used in the AF algorithm is employed to detect whether a node is already in the list. These four label correcting schemes were compared to a rather efficient label setting design based on Gilsinn and Witzgall's (1973) implementation of Dial's algorithm. In Shier's modification, a k-vector label is kept for every node. Recognizing that the components of each vector arc kept in strictly increasing order by themselves, since the already scanned (permanent) labels will always precede the unscanned (temporary), an identifier array, similar to that of SL method, is used to detect if a node is already in the bucket structure. However, the author did not describe the implementation details of the double linked list of each bucket. The comparative tests performed showed superiority of the last label setting implementation. This should have been expected because the particular implementation scheme of the label setting procedure is highly efficient and advanced, unlike the label correcting implementations tested. Designs using deque data structures would have given better results.

The implementations developed for DYNASMART and the tests performed to evaluate alternative designs have benefited from Shier's work. All three schemes implemented here are expansions of the simple shortest path case. However, unlike previous implementations, this study has gone beyond simple modifications of single path cases to take advantage of the special structure of the k-path case, resulting in more efficient designs. The specific details of these procedures are given in the next two sections.

SINGLE USER CLASS

Analysis and Design of the k-Shortest Path Algorithm

In this research we compare three basic k-shortest path schemes: a label setting, a label correcting and a hybrid scheme. The label setting scheme employs a heap structure for the scan eligible (SE) list. This scheme had been implemented previously in an earlier version of DYNASMART (Jaykrishnan, 1992). The structure tested here differs slightly from Jaykrishnan's implementation in some details that lead to better execution times for some classes of networks. In addition, the heap structure has been regenerated to produce programs coded by the same programmer, so that objective comparisons can be made. Moreover, the heap structure is one of the most efficient designs for label setting algorithms, and therefore constitutes the best choice for testing this class of algorithms. The second approach tested is a label correcting algorithm with double ended queue design for the scan eligible list. Different possible strategies that take advantage of the k-vector structure of the labels are evaluated. In depth analysis of the mechanisms of the label correcting approach was also performed. Third, a modification of Glover's original PSP algorithm for the k-shortest path problem was designed and implemented.

In this report, only the label correcting algorithm is extensively discussed, because its performance was found to be superior to the other two, and is as such included in the current version of DYNASMART. In what follows, it is assumed (with no loss of generality) that node 1 is the given origin node and that the k-shortest path lengths (times, costs) to all nodes of the network are sought. The basic steps used by a label correcting algorithm are outlined below. Explanation of the basic terminology used here can be found in the first section. No label is permanently set' by this approach before the algorithm terminates.

Label Correcting K-Shortest Path Algorithm

Step 1: Initialize the labels of all the nodes in the network with an initial upper bound. That is, assign a k-vector hj=(hJ ,LJ-2,. . .,hjk) to every node **j,** where the components of the vector λ_j are listed in increasing order. While several approximations could be used to begin the process, it is convenient to use the initial guess:

$$\lambda_1 = (0,\infty,\ldots,\infty)$$
$$\lambda_j = (\infty,\infty,\ldots,\infty\ldots) \quad \forall j, j \neq 1$$

Step 2: Create a list that will contain all the nodes that are 'eligible to be scanned', (SE) list, and insert the node **1** into it. By 'eligible to be scanned', it is meant that these nodes have the potential to reduce at least one of the labels of their succeeding nodes. The meaning of scanning is explained later.

Step 3: Select the top node from the SE list. If there are not any other unscanned components of the k-vector label of this node, delete the node from the SE list and scan it. If there is an unscanned label associated with this node, leave the node in the SE list. Let **i** denote the chosen node, and **j** any node reachable from **i**. Scanning has the same definition as in the label setting case. The scanning procedure can be summarized again as follows:

If $\lambda im+Cij < \lambda jk$ then replace λjk by $\lambda im+Cij$, and
insert node j in the SE list (if it is not already there). Otherwise do nothing.

This procedure is repeated for every m=1,...,k and for every $\mathbf{j} \in \Gamma(\mathbf{i})$. If the SE list is empty, go to step 5.

Step 4: Repeat Step 3.

Step 5: Terminate the procedure. The k-vector label λ_i for every node **i**, contains the k-shortest labels from the origin node to node **i**.

This algorithm will work even if some of the network arcs have negative costs, as long as no negative cycle exists in the network. Proof of this algorithm can be found in Tarjan (1988). The fact that the node chosen to be scanned from the SE list is not constrained to be the minimum label node (as in the label setting procedure), makes Step 3 of this algorithm less expensive computationally than label setting. The advantage of this approach is that the search, deletion and insertion in the SE list require just a few operations; the disadvantage is that a node may enter the SE list multiple times. Consequently the computational time required by Step 3 (denoted by σ) is inherently very small, while the number of iterations is larger than **kN**. Considerable effort has been directed at reducing this number of iterations. Several data structures have been proposed in the literature for this purpose, especially for the ordinary shortest path case. Some extensions to the multiple path case have been proposed by Shier, as discussed in the review section. However, no structure has been proposed specifically for the k-shortest path case. This is one of the objectives of this study.

In the next section, a method is proposed to dramatically improve the effectiveness of any label correcting approach for k-shortest path problems. This method, which can be applied with any SE list structure, takes advantage of the fact that k-vector labels are needed instead of just one label. The SE list in this research is modeled as a double ended queue (deque), although other structures are also investigated. The implementation details of this structure are given in the next section. Some theoretical considerations are discussed in this section.

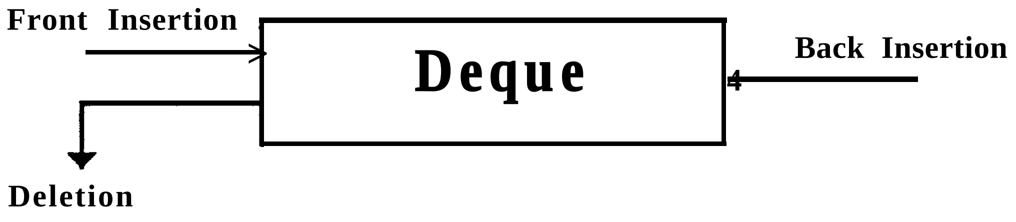


Figure 47. Double Ended Queue (Deque)

As shown in Figure 47, the deque is a simple linked list where the insertion can be done at both ends, while deletion takes place only from the front end. The deque design has been reported to outperform others for label correcting algorithms for ordinary (single) shortest path problems. However, no implementation for the k-shortest path case has been reported.

Implementation of the K-Shortest Path Algorithm

Network and Path Representation

A network may be represented in a computer in several ways, and the manner in which it is represented directly affects the performance of algorithms applied to the network. The most popular way of storing a network is to use a linked list structure. In this method, all the arcs that begin at the same node are stored together and each is represented by recording only its ending node and length. A pointer is then kept for each node (heading) to indicate the block of computer memory locations for the arcs beginning at this node. The set of arcs emanating from node **i** is called the forward star of node **i.** If the nodes are denoted sequentially from 1 to N and the arcs are stored consecutively in memory such that the arcs in the forward star of node **i** appear immediately after the arcs in the forward star of node **i-1, then** this method, called the forward star form, requires only N+2A units of memory.

Throughout this section we assume that the network is represented in forward star form. The forward star structure for the example in Chapter 2 is shown in Figure 48.

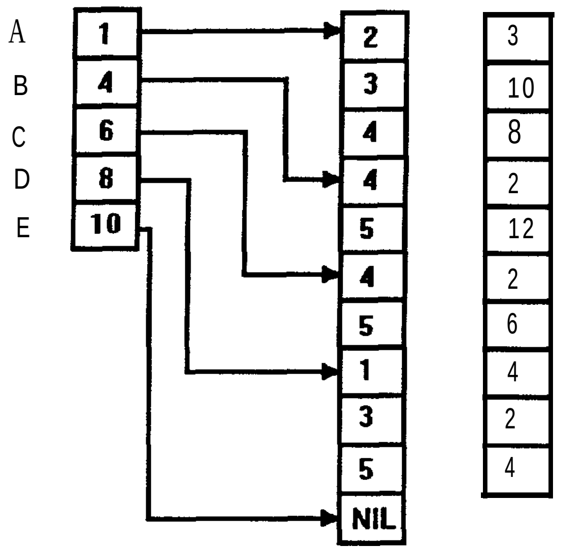


Figure 48. Forward Star Network Representation

A similar structure that may sometimes be more appropriate is a backward star form. It has an analogous structure, with the nodes that precede a node recorded in its backward star. This form is particularly useful if the shortest path tree is rooted at the destination.

Another related representation issue is the storage of the resulting k-shortest paths. An efficient way is to use pointers to the previous node (along the path). This is used for the ordinary shortest path case and can be extended to the k-path case. For each element of the k-vector label a two-dimension pointer array is defined. The first dimension holds the previous node in the path and the second dimension the rank (k) of the label of this node. This is shown diagrammatically in Figure 49.

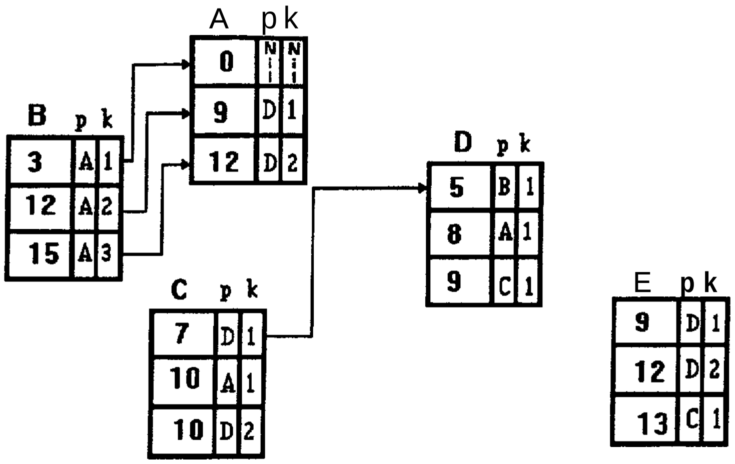


Figure 49. Path Storage Example

Implementation of the Algorithm

In a previous section we introduced the basic scheme of a label correcting algorithm and proposed a double ended queue (deque) structure for the scan eligible (SE) list. The objective here is to minimize the number of reentries of a node in the list.

The deque structure has been implemented for the ordinary path case with very good results for a broad class of networks, including transportation networks. It is easy to implement in any general purpose high level language, and does not require complicated structures or pointers. In order to implement it, one needs to specify, for every node updated by the algorithm, if the node is currently in the SE list, or if not, whether it has been in the list in the past. Depending on the answer, the node is inserted from a different end in the deque.

Furthermore, some way of representation is needed for the internal arrangement of the nodes in the deque so that the basic insertion and deletion operations can be accomplished. Pallotino and Gal10 (Pallotino, 1984; Gal10 and Pallotino, 1988) have extensively studied different SE list representations, including the deque, and proposed a one-dimension array of size N that takes the following values:

$$\text{Deque}(i) = \begin{cases} 0 & \text{if the node is not currently in the Deque} \\ & \text{and it has never been in there before.} \\ -1 & \text{if the node is not currently in the Deque but} \\ & \text{it has been in there before.} \\ j & \text{where } j \text{ is the next node in the Deque, if it is} \\ & \text{currently in the deque and it is not the last node in the} \\ & \text{structure.} \\ \infty & \text{if it is the last node in the Deque.} \end{cases}$$

In addition to these, two pointers must be used for the deque: one pointing to the first element of the deque (FirstDeque) and one at the last (LastDeque).

Next we define the following basic operations of the deque: creation, insertion, deletion. The *creation* is an initialization procedure that must take place once at the beginning of the program. It assigns the values: $\text{Deque}(i)=0 \ \forall i \in N-\{1\}$ and $\text{Deque}(1)=9999999$ The *deletion* of a node from the deque consists of identifying the first element of the deque, and then deleting it. This is accomplished using the FirstDeque pointer to pick the first element and then to move this pointer to the next node by assigning:

FirstDeque=Deque(FirstDeque).

Finally, the *insertion* operation inputs a node in the deque. If the deque is empty, then the inserted node is named first node and last node at the same time and its Deque(Node) points to infinity. If the list is not empty, then the input node is inserted at the beginning or at the end of the structure depending on its Deque(.) label value. Both operations involve two steps:

```
If Deque(Node)=0 then
        Deque(Node)=FirstDeque
        FirstDeque=Node
Else If Deque(Node)=-1 then
        Deque(LastDeque)=Node
        LastDeque=Node
```

A node is inserted in the deque because its k-vector label has been improved. When we delete this node from the deque to scan it, we need to know the specific label component that has been improved from this node's label vector, so that it can be used to update subsequent nodes. Therefore, when a node is inserted in the deque, it must carry its label value with it. This is accomplished in the label setting case with an ID pointer to this specific label in the node's label vector. A similar concept could also be used in the label correcting case. However, it is not important to use just one variable as an ID for the deque, because no internal arrangements are taking place inside it. More than one identification variable (or even part of the k-vector label) can be assigned to the node entering the deque. Another difference with the heap (in the label setting case) is that the latter is intended to support a scheme that must always scan the minimum label node. Therefore the elements in the heap are arranged according to their corresponding label-components. The same node may occupy several slots in the heap if more than one of its labels have been improved. Each instance (of the same node) is scanned separately according to the corresponding label. This can be avoided in the label correcting case. Each node in the SE list can have more than one label associated with it, instead of multiple instances of the same node with different labels. When a node is deleted and scanned, all its labels can be scanned simultaneously. This leads to substantial savings in the execution time of the algorithm. This design is shown diagrammatically in Figure 50.

This design is not a simple extension of the one for the ordinary shortest path. It takes advantage of the fact that k paths are sought and simultaneously calculates as many of these paths as possible. It does not depend on the specific SE list structure. The same scheme was also applied in conjunction with Glover's PSP algorithm.

Another implementation aspect of this approach is the searching-sorting of the k-vector label. Every time a new label is calculated for a node, it must be compared to the maximum component of the k-vector label for this node. If greater than this maximum value, the label is rejected; otherwise it must be included in the label structure and properly placed depending on its value. In the label setting approach this is not a problem since the label setting property guarantees that a scanned label is always the minimum. But for a label correcting implementation this is an important and time consuming task. There are several ways to sort the incoming elements of the label structure: from simple naive sequential comparisons, to binary trees and strategies based on Fibonacci numbers. The fact that the number of desired labels is in the order of at most 10 led us to reject "advanced" structures with high associated overhead cost (Aho et al., 1983), and to use a simple linked list structure. This structure was implemented using internal array pointers to the next smaller label in the structure. An example of such a structure is shown in Figure 51.

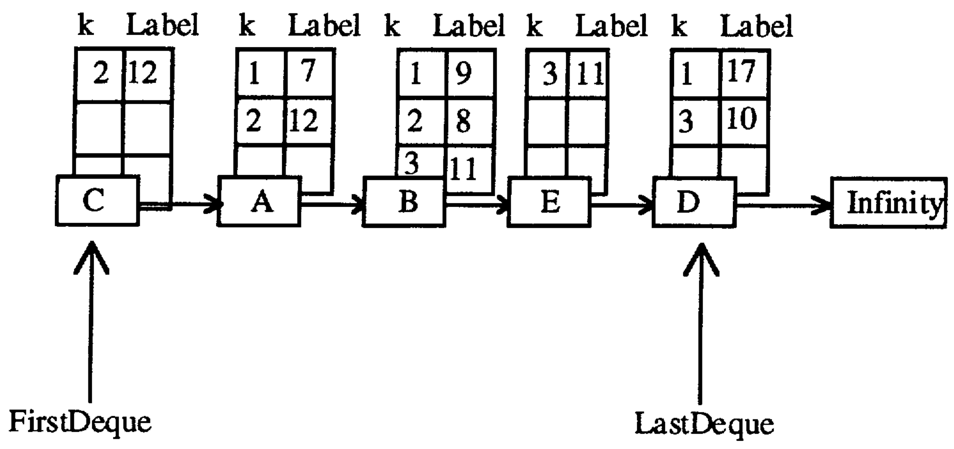


Figure 50. The Structure of the Deque with More than One Labels Per Node

		Label	Pointer
	1	56	8
	2	34	4
	3	8	NIL
	4	23	10
	5	11	3
First Label Pointer --->	6	89	1
	7	43	2
	8	45	7
	9	17	5
	10	22	9

Figure 51. A Typical k-Vector Label Structure for k=10

A pointer (First Label Pointer) is used to indicate the largest element in the structure. Every time a new label is obtained from the algorithm for this node, it is first compared to the label corresponding to this pointer. If it is smaller and must enter the structure, it takes the position of the largest label. Then the internal pointer must be arranged so the new label is placed properly. This is done sequentially by comparing the incoming element to the existing labels from the largest to the smallest.

This procedure was selected among several other alternatives. We decided to have the labels in descending order after noting that there is a higher probability for a smaller label to be produced earlier, because in general paths with more arcs tend to be longer.

Next, we summarize the entire algorithm using the implementation approaches discussed:

Step I: Initialize all the label vectors to 999999, except the first label of the origin node, which takes value 0. Initialize the first label pointer of the origin node to 1.

Step 2: Create the deque, and insert node 1 into it, with associated unscanned label 0. Initialize all the Deque(i) pointers to value 0 except for the origin node; its Deque(.) pointer takes the value 999999.

Step 3: Delete from the deque the first node and name it CurrentNode. If the FirstDeque points to 999999, then go to step 5. For every label associated with this node scan all neighbor nodes and update their labels as follows:

Calculate the sum hmi+Cij , where i is the CurrentNode, j is a neighbor node, and hmi is an unscanned label of node i. If this sum is less than the largest label of node **j,** then include this label into the label structure of **j.**

If node j is already in the deque, add this new label into its label structure as an unscanned one. Otherwise, *insert the* node in the *deque* with the new *label unscanned.*

Step 4: Go to step 3.

Step 5: Terminate the process. The k-vector label of every node contains the k-shortest paths from the origin node to this node.

Computational Testing and Results

The results of computational tests to evaluate and compare the methods implemented are discussed in this section. A number of randomly generated shortest path problems were solved, along with a real transportation network, using the same computer (CRAY Y-MP/8 supercomputer) and the same compiler (UNICOS CFI77 FORTRAN). All the codes were implemented by the same programmer and no attempt was made to take advantage of any of the advanced hardware features of CRAY at this stage. Each test problem was run 50 times using different origin nodes randomly selected but consistently used with every code executed on that network. The average execution time for the 50 runs is reported in each case. Every effort was made towards efficient code design. Subroutines were unified into the main program for the test runs so that no time is spent for subroutine calls.

The test networks consist of three sets of random networks and one real large urban street network, that of the core area of Austin, TX, consisting of 625 nodes and 1742 arcs. The random networks were generated using a special purpose random network generator (Ziliaskopoulos, 1992).

The following two tables present execution times (in CPU milliseconds) for a series of computational experiments designed to study the relative efficiencies of the three tested schemes: LC for label correcting with deque list structure, LS for label setting with heap structure, and PSP for modified version of Glover's partioning algorithm with deque structure.

Table 20. Computational Results in CPU Milliseconds for k=2

	100 N 250 A	500 N 1250 A	1000 N 2550 A	1500 N 4500 A	2500 N 7500 A	625 N 1742 A
LC	.60	4.08	10.99	18.29	31.66	4.78
LS	1.91	14.05	33.97	56.80	104.45	14.62
PSP	1.39	5.87	11.48	18.59	25.38	6.17

Table 21. Corn utational Results in CPU Milliseconds for k=5

	100N 250 A	500N 1250 A	1000N 2550 A	1500N 4500 A	2500 N 7500 A	625 N 1742 A
Lc	1.31	9.88	30.23	45.85	83.18	10.15
LS	5.67	40.80	94.7 1	160.46	294.00	42.06
PSP	2.07	10.76	31.43	38.3 1	57.91	10.95

Table 22. Computational Results in CPU Milliseconds for k=10

	100N 250 A	500N 1250 A	1000N 2550 A	1500 N 4500 A	2500 N 7500 A	625 N 1742 A
Lc	3.66	25.13	84.60	120.78	224.3 1	23.82
LS	12.7	88.33	2 10.72	345.04	642.5	92.55
PSP	3.11	29.41	83.72	94.77	182.15	23.77

Table 23. Computational Results in CPU Milliseconds for k=15

	100N 250 A	500N 1250 A	1000N 2550 A	1500 N 4500 A	2500 N 7500 A	625 N 1742 A
LC	7.26	47.72	167.89	231.98	443.56	44.85
LS	21.05	176.26	435.91	752.80	1196.58	157.45
PSP	8.13	46.95	100.32	117.82	367.46	48.98

Table 24. Means and Standard Deviations for k=5

	100N 250 A (mean std)	500N 1250 A (mean std)	1000N 2550 A (mean std)	1500 N 4500 A (mean std)	2500 N 7500 A (mean std)	625 N 1742 A (mean std)
LC	1.31 0.2033	9.88 1.3742	30.23 3.1011	45.85 7.0487	83.18 8.3540	10.15 0.3867
LS	5.67 0.032 1	40.80 0.0985	94.7 1 0.2569	160.46 0.9856	294.00 2.0263	42.06 0.1046
PSP	2.07 0.5213	10.76 2.0112	31.43 3.8834	38.31 6.3464	57.91 11.453 1	10.95 0.6677

In Table 24, the standard deviations are given for the same test set and for k=5. These standard deviations have been calculated from 50 different runs using the same destinations as the data in Table 21. In this case, the individual execution times from every origin have been recorded so that the standard deviations can be calculated.

The first and most important conclusion from the previous analysis is that, in general, the label correcting algorithm performs better than all the other implementations for medium and small size networks. However, the partitioning shortest path algorithm is superior for larger networks. This can be attributed to the fact that the PSP algorithm performs less efficient scanning of the list structure than the LC algorithm for small networks.

Another observation is that the execution times for the PSP algorithm exhibit higher variance than the others, as seen in Table 24. The label setting algorithm has the lowest variance in all cases. This can be explained by the fact that the number of iterations for the LS implementation is almost constant and independent of the origin node (kN iterations), and iterations do not differ greatly from each other, unlike the other two codes for which the composition of the SE list is heavily dependent on the origin node.

Update Path Algorithm

The path processing component is by far the most computationally expensive part of DYNASMART. This is the primary motive for studying alternative schemes that can accurately compute paths faster. The scheme presented in this section was introduced in Chapter 2. It consists of updating the already computed paths every simulation step (or small number of steps),while the k-shortest path algorithm computes the paths “from scratch” at specified intervals. This approach might miss a path that has become a k-best path at some point between two consecutive computations of the paths; however, it is unlikely to miss the best two or three paths. The trade-offs between updating versus calculating paths are discussed in a later section. In this section, an efficient scheme to update an existing k-shortest path tree with new travel times is presented. Two alternative update path schemes were investigated for this study. The first was based on a list update design that can vectorize readily but includes many computationally redundant parts. The second approach uses a compact tree update scheme that does not vectorize well because of its highly sequential nature; however, it avoids redundant computations and ultimately outperforms the first approach. The tree based approach is discussed in the next section.

This algorithm is based on a rather straightforward tree traversal procedure that sequentially updates the nodes with new labels moving in a depth-first fashion down the k-shortest path tree. The tree traversal procedure does not actually matter as long as a node-path is updated after its predecessor node-path has been updated. This is guaranteed by assigning a serial number *(PriorityNo)* to each node-path so that if a node on path m of node *i* points to node j and path *1,* then *PriorityNo(i,m)>PriorityNo(j,l)*. In order to achieve this, a procedure was built *(BuildPriorityTree())* to sequentially trace, for every node-path, its path to the destination node or to an already traced node-path. The steps of *the BuildPriorityTree()* procedure follow:

BuildPriorityTree() Procedure

Step I. Assign priority 1 for the first path (k=l) of the destination node.

Step 2. For every node and path *(i,m)* in the network other than the first path of the destination node run Step 3. If all the node-paths have been scanned, then go to Step 5.

Step 3. Check if node-path *(i,m)* has already been assigned a serial number. If assigned, then go to step 4 and move to the next node-path; otherwise, trace the next node-path *(j,l)* that follows this node-path along its path to the destination node, save (i,m) in sequential array *PriorityPointer(),* name *(j,l)* current node-path and go to Step 3.

Step 4. If the PriorityPointer() array is empty go to Step 3; otherwise sequentially assign to every node-path *in the PriorityPointer()* a serial number increasing from the last entry to the **first** and go to Step 2.

Step 5. Terminate.

When the procedure terminates, a unique serial number has been assigned to every node-path.

The node-paths are updated iteratively starting from the one with priority number two up to the node-path with priority number Nk. *The BuildPriorityTree()* procedure guarantees that when the label for the node-path *(i,m)* is computed by adding to the label of its predecessor node-path *(j,l)* the new travel time of arc *(i,j), the* label of *(j,l)* has been updated. The final step of the update path algorithm is a sorting subroutine that sorts the updated path in increasing order of label size. Next the overall update path algorithm is summarized.

Update Path Algorithm

Step 1. Call the procedure BuildPriorityTree()

Step 2. For Priority Number ip=2 to Nk, repeat the following operation:

Set the label of the node-path with priority number ip equal to the sum of the label of the predecessor node-path and the new travel time of the arc that connects the two labels.

Step 3. For every node sort the labels of the paths.

Step 4. Terminate the algorithm.

Implementation of the algorithm is straightforward. The computational performance of the proposed update path algorithm is reported for different size random networks and the core street network of Austin, TX (625N, 1742A) in Table 25.

Table 25. Computational Results in CPU Milliseconds for k=10

100 N 1250 A	500N 1250 A	1000N 2550 A	1500 N 4500 A	2500 N 7500 A	625 N 1742 A
1.98	7.89	23.81	34.42	63.17	7.03

The results in Table 25 demonstrate the considerably lower computational requirements of the update path scheme compared to all the k-shortest path algorithms. The update approach performs better for larger networks because the effect of an initial start-up cost is reduced for larger networks. Designs that combine update and calculate algorithms are investigated in this chapter's last section.

Parallel Design of Path Processing Algorithms

Application of parallel programming concepts to large network problems has attracted considerable attention (Mahmassani, 1990; Zenios,1991) mainly because of its potential to improve computational times for large problems. In this section, an assessment of the capabilities of parallel computers to speed-up path processing computations is made and different possible alternative designs are evaluated. A brief description of relevant parallel programming concepts is included first.

Some Concepts of Parallel Programming

The schemes examined here are designed with a multiple instructions multiple data (MIMD) machine in mind. CRAY Y-MP/8 belong to this category, with 8 CPU's and shared memory communication environment (CRAY, 1989). A number of parallel processing capabilities are available on this machine such as macrotasking, microtasking, autotasking, vectorization, and I/O subsystem parallelization.

Only macrotasking is studied in this application because it is a feature commonly available on every parallel machine. Vectorized designs are discussed in Chapter 7 in conjunction with the implementation of the time-dependent shortest path algorithm. Macrotasking is a form of multitasking that uses multiple processors in a FORTRAN program at the subroutine level. The whole operation is controlled by the programmer who is responsible to explicitly partition the program into tasks, each of which is eligible to run on a CPU. Typically, these tasks may take the form of different subroutines that can be executed concurrently, or they can involve separate invocations of the same subroutine. A task is a piece of code and data that can be scheduled for execution on a CPU.

The basic problem that arises in parallel programming is computational and storage dependence among the tasks. Computational dependence includes data dependence and control dependence. Data dependence is an ordering relationship between statements that use or produce the same data, while control dependence refers to the situation in which the order of execution of statements cannot be determined a priori. This happens when conditional statements (IF commands) are encountered in the program. Finally, storage dependence has to do with the independence of the workspace. Each parallel computational task has access to variables, and the fetching and storing of all the variables in one task must not interfere with that in another task

Parallel k-Shortest Path and Update Path Algorithm

Parallelism was exploited in this study at two levels: tree rooted at single destination (or origin), and multiple destinations (origins). At the single destination level, the algorithm is executed jointly on several CPU's; at the multiple destination level, each destination root node is assigned to one processor that executes independently a copy of the algorithm. The first approach did not perform satisfactorily, primarily because of the characteristics of the CRAY Y-MP/8 which entail considerable overhead for several processors to collaborate on finding the shortest path tree for a single root node. Some design details are briefly discussed. The second design performed extremely well. Its design and implementation details are discussed in this section and some computational results are given.

Initially, the network structure and the travel time data are read sequentially by a single processor and stored in the common memory. Next the variables and parameters are initialized by a single processor, although they could be initialized in parallel, because this step takes only a very small portion of the total computation time. A copy of the path computation iterative procedure is called once by each processor and runs completely independently. Every copy maintains its own SE list and k-shortest tree which are stored in the private memory space of each processor. The only time the shared memory is accessed is when travel time data is needed. In theory, this step does not interfere with the other processors because the data are accessed in read-only fashion. However, in practice, this can create memory contention, especially if more than one processor is trying to access

the same piece of data simultaneously. The steps of the above scheme are described next. Two library functions of CRAY are employed to implement this approach:
TskStart(): assigns a subroutine S to a processor P,
TskWait(): waits for a processor P to become idle.

Parallel k-Shortest Path Algorithm for Multiple Destinations

```
Step 1.  Read network structure and travel time data and initialize the parameters and
variables.
Step 2.  Do Step 3 for ip=l to NumberOfProcessors- 1.
Step 3.  Let Destination(ip) to be the current destination;
Call TskStart to assign the k-shortest path subroutine to processor ip.
Step 4.  Call TskStart to assign the k-shortest path subroutine to processor
Num berOfP rocessors.
Step 5.  Call TskWait and wait for all NumberOfProcessors processors; Terminate.
```

The k-shortest path subroutine has the same structure as the sequential scheme described in a previous section. An identical design is used for the update path algorithm. The only difference is that the update path procedure is called instead of the k-shortest path procedure.

The above scheme was coded and tested on street-like networks of varying sizes. The results were extremely stable and insensitive to the size of the network for the networks of Table 22. Four processors were utilized in a dedicated environment yielding speed-up of approximately 3.6 in almost every test. Table 26 shows the expected speed-up of a similar architecture with varying number of processors according to Amdahl’s Law (CRAY, 1989).

Table 26. Theoretical Speedup for the Parallel k-Shortest Path Algorithm

Processors	1	4	8	16	64	Infinity
Speedup	1	3.6	6.5	10.5	19	27

In contrast to the proposed design, parallelization at the level of the individual algorithm (for a single destination root node) did not perform well. The parallelization was applied on the deque operations. Specifically, each processor reaches the deque and deletes the first node of the deque. Next, it scans the deleted node by reaching its neighbor nodes. If the labels of any of the reached nodes is improved, then these nodes are inserted in the deque. In order for this scheme to work, however, the deque must be protected from simultaneous access by two or more processors that could destroy its structure and result in wrong results. Protecting the deque turns out to be very expensive computationally because it is activated and de-activated every single scanning iteration, thereby incurring significant overhead. Nevertheless, had the library of parallel functions of CRAY been cheaper, this scheme would perform fairly. Moreover, it must be noted that in the above scheme we did not consider intersection movements and time-dependency. If these factors are considered the size of the task (grain) is expected to increase so that parallelization at the level of a single destination would be expected to perform satisfactorily.

MULTIPLE USER CLASSES

K-Shortest Path Algorithm : Analysis and Implementation.

Two alternative schemes are considered for computing paths on networks with multiple user classes. The first approach is a simple extension of the single user class design. It successively computes the paths using multiple realizations of the network (one for each user class) with different arc costs (for the corresponding class) for every realization. The computational complexity and memory requirements of this scheme are direct multiples of those for a single user class. This design was challenged by another more advanced scheme that is more efficient in terms or memory and computational time. This scheme combines the k-shortest path and the update path algorithms presented in an earlier section in an intelligent way to create upper bounds for one user class using the paths computed for the previous class. Moreover, computing k paths for every user class will result in mk total paths for every node. Most of these paths are expected to be the same or overlap to a great extent, i.e., the mth best path for one user could be the 1th best path for another user for the same origin-destination pair. This redundancy is avoided in the design proposed in this section. In order to avoid the computation and storage of redundant paths we combined a calculate/update scheme that is first described in words and then outlined formally in algorithmic steps.

Using the k-shortest path algorithm presented in the previous section (for a single user class), we compute kinitial paths for the first user class without using upper bounds. Next *we use the Update Path0* algorithm presented previously and update the stored paths for the first class using the arc costs of the second user class. These paths are upper bounds to the actual shortest paths for the second user class; by using them in computing the shortest paths for the second user class, the computation time can be dramatically improved. The improvement results not only from the paths of the previous user class being sharp upper bounds, but also because some of the paths actually do not change at all. They may have a different rank for the new user class but they are physically the same. If a path remains the same, then its shortest path label equals the upper bound computed in the *Update Path* subroutine and can therefore be easily detected. Suppose k2 paths are needed for the second user class, and kl for the first user class. The kl least paths from the kinitial paths that we have computed are marked as permanent and will not be deleted from the structure even if they are not among the k2 paths for the second user. If any of these paths happen to be among the k2 paths for the second user class, then they are marked as such. If the non-similar paths are greater than (kinitial-kl) then we augment the path structure. The above reasoning applies to the remaining user classes and is summarized in the following algorithm:

K-Shortest Path Agorithm for Multiple User Classes

Step 1. Compute kinitial paths for the first user class using the k-shortest path algorithm for a single user class.

Step 2. Repeat Steps 3 to 5 for CurrentUser=2 to Number of Users.

Step 3. Call the Update Path algorithm using as new costs the arc costs for the CurrentUser class.

Step 4. Compute kCurrentUser paths using the k-shortest path algorithm with upper bounds, stated next.

Step 5. Call the BuildPriorityTree procedure.

Step 6. Update the common k-path structure for every user class and destination.

Step 7. Terminate

Label Correcting K-Shortest Path Algorithm with Upper Bounds

Step 1. Initialize the labels of all the nodes in the network with an initial upper bound. That is, assign a k-vector $\lambda_j=(\lambda_j^1,\lambda_j^2,\ldots,\lambda_j^k)$ to every node **j**, where the components of the vector λ_j are listed in increasing order. These values are upper bounds to the shortest values and are computed in the Update Path Algorithm.

Step 2. Create a list that will contain all the nodes that are 'eligible to be scanned', (SE) list, and insert node 1 in it. A node is 'eligible to be scanned' if it has the potential to reduce at least one of the labels of its adjacent nodes.

Step 3. If all the nodes have been scanned at least once and the SE list is empty, then go to Step 6; otherwise go to Step 4.

Step 4. Select the top node from the SE list. If there are no other unscanned components of the k-vector label of this node, delete the node from the SE list and scan it. If there is an unscanned label associated with this node, leave the node in the SE list. Let **i** denote the chosen node, and **j** any node reachable from **i**. The scanning procedure can be summarized again as follows:

If $\lambda_i^m+C_{ij} < \lambda_j^k$ *then replace* λ_j^k *with* $\lambda_i^m+C_{ij}$, *and insert node j in the SE list (if it is not already there). Otherwise do nothing.*

This procedure is repeated for every m=1,...,k and for every $\mathbf{j}\in\Gamma(\mathbf{i})$. If the SE list is empty, go to Step 3.

Step 5. Repeat Step 3.

Step 6. Terminate the procedure. The k-vector label λ_j for every node **i**, contains the k-shortest labels from the origin node to node **i**.

When the overall algorithm terminates, there is a single structure with K paths that contain at least k^1 best paths from the first user, k^2 best paths of the second user and so on. However, because the paths do not differ significantly for the different user classes, the total number of stored paths K is significantly less than $\sum_{i=1}^{\text{no of users}} k^i$. Moreover, some of the K-k^i paths for the ith user class are among (or are very close in length to) the K best paths of this user class. Note that the proposed algorithm computes the paths for multiple user classes but only for one destination node (from all origins).

The implementation of the k-shortest path algorithm for multiple user classes does not differ conceptually from the ordinary algorithm. The individual pieces of the algorithm follow the same guidelines described previously.

Update Path Algorithm for Multiple User Classes

The update path algorithm for the multiple user classes case is exactly the same as that for a single class. Moreover, it is used internally by the k-shortest path algorithm for multiple user classes as explained in the previous section. In addition, it **is** used to update the k-shortest paths every simulation step of DYNASMART in the same way as in the single user case. The only difference is that more paths are typically updated (K-paths) in the multiple class case than in the single class case.

UPDATE/CALCULATE COMBINATION APPROACHES

In this section, we evaluate the strategy of combining calculate and update approaches to speed up the path processing computations in DYNASMART. The objective is to specify the "optimum" number of time intervals that the paths are updated between two successive computations. In Table 27, the average computation time for one run for the k-shortest path (label correcting) and update algorithms are given together for comparison. In Figure 52 the computational times are sketched for various scenarios with increasing number of update time intervals, i.e. less frequent computations.

Table 27. Computational Results in CPU Milliseconds for k=10 for Calculate and Update Algorithms

	100N 250 A	500N 1250 A	1000N 2550 A	1500 N 4500 A	2500 N 7500 A	625 N 1742 A
Calculate	3.66	25.13	84.60	120.78	224.3 1	23.82
Update	1.98	7.89	23.8 1	34.42	63.17	7.03

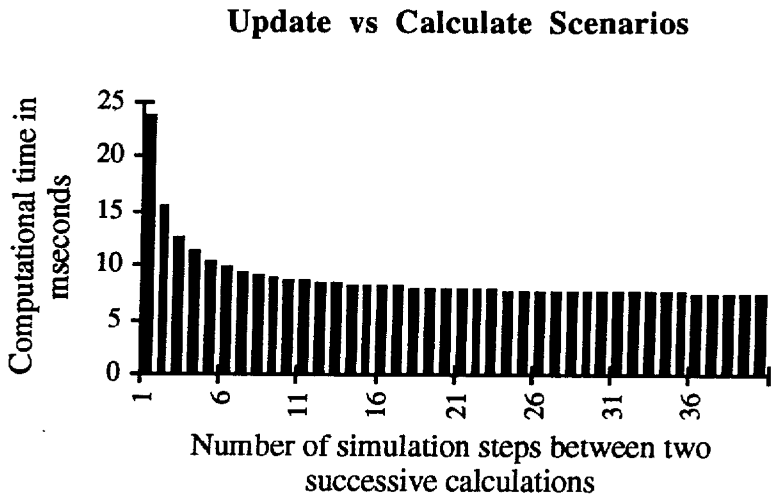


Figure 52. Computation Time for Increasing Number of Simulation Steps Between Two Successive Calculations

It is clear from Figure 52 that beyond 15 update steps the benefit of combining update and calculate schemes is leveling off. This justifies the fact that 20 simulation steps are interposed between any two consecutive computations currently used in DYNASMART. Another design that should be further investigated is the use of variable number of interposed simulation steps. When the conditions on the network are fairly stable (off-peak period) many steps are interposed, while as conditions become more variable and dynamic (during the peak period) the number of steps is decreased.

CHAPTER 7
MINIMUM PATH ALGORITHMS FOR NETWORKS WITH TIME-DEPENDENT ARC COSTS

In this chapter, we introduce two new minimum path algorithms for networks with time-dependent arc costs: a least-time path and a least-cost path algorithm. The least-time path algorithm calculates the minimum travel time paths on time-dependent networks with arbitrary travel time functions. The least cost-path algorithm is a general approach that can calculate minimum travel cost paths on networks where the travel cost is not the travel time itself. Unlike other approaches, these algorithms are not restricted only to networks with first-in-first-out links. They are based on the general Bellman's principle of optimality. They discretize the horizon of interest into small time intervals and by starting from the destination node they calculate the minimum paths operating backwards. The least-time path algorithm is used by the algorithm described in chapter 4 to compute a time-dependent user equilibrium traffic assignment in a network, while the least cost path is used by the system optimal time-dependent algorithm. Both algorithms are incorporated in the multiple user class assignment procedure.

This chapter is organized as follows: in the first section, a brief background review is presented. In the second section, the least-time path algorithm is analyzed. In the third section, the least-cost path algorithm is presented. Finally, in the last section an efficient implementation scheme for the least-time path algorithm is presented in detail and computational results are reported.

INTRODUCTION AND LITERATURE REVIEW

Introduction and Research Objectives

This study introduces two new algorithms: a time-dependent least-time path (LTP) and a time-dependent least-cost path (LCP) algorithm. The LTP algorithm computes the least-time paths on networks with time-dependent travel times on the arcs, while the LCP algorithm computes optimum paths on time-dependent networks where the travel cost is not the travel time itself. The LTP algorithm is required in the time-dependent UE assignment solution procedure, the LCP algorithm is used in the SO assignment algorithm. In the latter, the arc costs consist of the marginal travel times, and the least marginal travel time path is sought.

Both algorithms use discrete travel time and travel cost functions over a time period of interest (e.g. peak period) discretized into very small time intervals, A vector of labels, one for each time interval, is maintained for every node and is updated in a label correcting fashion. The scan eligible (SE) list can have any structure applicable to the static label correcting shortest path case. In this study, the algorithms are shown to have polynomial computational complexity, if the SE list has a simple queue structure; however, best performance has been obtained with a double ended queue structure for the SE list. Both algorithms proceed backwards, starting from the destination node, as they calculate the optimum paths from all nodes and for every discrete time interval to the destination node. An implementation scheme is presented for the new algorithms, to efficiently calculate paths for large street networks on commercially available computers. Both algorithms have been implemented and coded on a CRAY Y-MB/8 supercomputer, and tested on real street as well as large random networks. This scheme performs extremely well, much better than the theoretically computed upper bound, because it takes advantage of the fact that although the travel times or costs may change every discrete time interval, the best paths do not necessarily change as frequently. In fact, only few paths become best paths for a given origin-destination pair, even for long time periods and highly dynamic networks.

Literature Review

The first paper dealing with the time-dependent shortest path algorithms appears to be by Cooke and Hasley (1966). They developed an iterative function, which is an extension of Bellman's principle of optimality (Bellman, 1957), that gives the time-dependent shortest paths from every node in the network to one destination node, for a set of discrete departure time steps. The travel times on the arcs are defined in positive integer time units, for every time step of the discrete scale $S_M = (t_0, t_0+1, t_0+2, \ldots t_0+M)$ The integer number M is chosen so that the travel times are defined for any $t \in S_M$. The travel times for $t > t_0+M$ are assumed to be infinite. This assumption eliminates all paths with arrival time to a destination node beyond t_0+M, leading possibly to undetermined paths for some nodes and time steps. This algorithm has theoretical computational complexity $O(|N|^3M)$, where $|N|$ is the number of nodes in the network. However, no implementation scheme for this approach has been reported and hence no computational results are available to determine its actual performance.

Dreyfus (1969) proposed a label setting approach that generalizes Dijkstra's static shortest path algorithm (Dijkstra, 1959). This approach calculates the time-dependent shortest path between two nodes for one departure time step with the same computational effort as for the static case (O(|N|2)). However, if the paths from all the nodes to a destination node are sought, and for every time step, this approach has the same complexity as Cooke and Hasley's algorithm.

An implicit assumption in Dreyfus' approach is that the FIFO *(first-in-first-out)* property holds on the network links. If this assumption does not hold, then Dreyfus' algorithm fails to detect the shortest paths. This has been stated in one form or another by several authors, including Halpem (1974), Malandraki (1989), Orda and Rom (1990), and Kaufman and Smith (1993). Orda and Rom (1990), recently, proposed an approach that is not restricted to FIFO links only. This approach can identify optimum waiting times **on the** visited nodes when such waiting is allowed, or the optimum waiting time at the source node if waiting is not allowed anywhere else. However, their approach fails to find efficiently the best path if waiting is not allowed everywhere along the path. It must be noted also that Orda and Rom's approach does not apply to networks where the cost on the links is not the travel time itself, but another time-dependent arbitrary cost function (such as the marginal travel time encountered in the SO time-dependent assignment algorithm).

LEAST-TIME PATH TIME-DEPENDENT ALGORITHM

Formulation of the Problem

Let G=(N,A) be a finite directed graph with |A| directed arcs connecting the nodes. Let dij(t) be the non-negative time required to travel from node i to node j when departure time from node i is t; dij(t) is a real-valued function defined for every t E S in such a way that t+dij(t)E S, where S={ t0, t0+6, t0+26, . . . t0+MG}, t0 is the earliest possible departure time from any origin node in the network, 6 is a small time interval during which some perceptible change in traffic conditions may occur, and M is a large integer number such that, the interval from t0 to t0+M6 is the time period of interest (or planning horizon).

We assume that dij(t) for t>t0+M6 is constant and equal to dij(t0+M6). This is a reasonable assumption for urban transportation networks where after the peak period, somewhat stable travel times can be assumed. Nevertheless, it is not a restrictive assumption since M is user defined and can always be increased to include time periods with variable travel times on some arcs. We denote by node D the destination node of interest in the network. The algorithm proposed in this section calculates the time-dependent least-cost paths from every node i in the network and at every time step t to the destination node D.

At each step of the computation, denote by hi(t) the total travel time of the current least-time path from node i to node D at time t. Let $\Lambda_i = [\lambda_i(t_0), \lambda_i(t_0+\delta), ..., \lambda_i(t_0+M\delta)]$ be an M-vector label that contains all the labels $\lambda_i(t)$ for every time step $t \in S$ for node i. Every finite label $\lambda_i(t)$ from node i to node D is identified by the ordered set of nodes $P_i=\{i=n_1, n_2, ..., n_m=D\}$.

According to Cooke and Hasley, $\lambda_i(t)$ is defined by the following functional equation:

$$\lambda_i(t) = \begin{cases} \min_{j \neq i}\left\{d_{ij}(t) + \lambda_j(t + d_{ij}(t))\right\} & \text{for } i \in N|D;\ t \in S \\ 0 & \text{for } i = D;\ t \in S \end{cases}$$

A modified version of this equation is the building block of our approach. Instead of scanning all the nodes at every iteration, a list of scan eligible nodes is maintained, containing the nodes with some potential to improve the labels of at least one other node. The proposed algorithm operates in a label correcting fashion; therefore, the label vectors are upper bounds to the least-time paths until the algorithm terminates.

The LTP Algorithm

Initially, the SE list contains only the destination node D. The labels of node D are set equal to zero and the labels of the remaining nodes to infinity. In the first iteration, all nodes that can directly reach D are updated according to equation (2), and inserted in the SE list.

$$\lambda_i(t) = d_{iD}(t) + \lambda_D(t+d_{iD}(t)) \qquad i \in \Gamma^{-1}\{D\}$$

where $\Gamma^{-1}\{D\}$ is the set of nodes that can directly reach D.

Next, the first node i of the SE list is scanned according to the following equation :

$$\lambda_j(t) = \min\{\lambda_j(t), d_{ji}(t) + \lambda_i(t+d_{ji}(t))\} \qquad j \in \Gamma^{-1}\{i\}$$

for every time step $t \in S$. If at least one of the components of Λ_j is modified, node j is inserted in the SE list. This scheme is repeated until the SE list is empty, and the algorithm terminates. Relations (2) and (3) are modifications of equation (1) which in turn is an extension of Bellman's principle of optimality.

The steps of the LTP algorithm are as follows:

Step 1. Create the SE list and initialize it by inserting into it the destination node D. Initialize the label vectors at the following values:

$\Lambda_D=(0, 0, ..., 0)$

$\Lambda_i=(\infty,\infty,...,\infty)$ for $i \in N\backslash D$.

Step 2. Select the first node i from the SE list, name it "Current Node" and delete it from the list. If the SE list is empty, go to step 4.

Scan the current node i according to relation (3), by examining each node j, $j \in \Gamma^{-1}\{i\}$. Specifically, for every time step $t \in S$ do the following :

Check if $\lambda_j(t)$ is greater than $d_{ji}(t)+\lambda_i(t+d_{ji}(t))$. If it is, replace $\lambda_j(t)$ in the label vector Λ_j at position t with the new value.

If at least one of the M labels of node j has been improved, insert node j in the SE list. The details of the structure of the SE list, and the associated operations of creation, insertion, and deletion are discussed in the last section.

Step 3. Repeat step 2.

Step 4. Terminate the algorithm. The M-dimensional vectors Λ_i for every node i in the network contain the travel times of the time-dependent least-time paths from every node i to the destination node D for each time step $t \in S$.

•

Next we state without proving the following Theorem:

Theorem 1. *Upon termination of the algorithm, every element of the vector label is either an infinite number, meaning that no path exists from this node to the destination node at the corresponding time step, or a finite number that represents the shortest path from this node and time step to the destination node.*

The proof of Theorem 1 can be found in Ziliaskopoulos and Mahmassani (1992).

The structure of the SE list affects the computational complexity of the algorithm. The complexity of the LTP algorithm is $O(M^2|N|^3)$, if a simple queue is used as a structure for the SE list. This structure actually performs worse than several other designs, but is convenient for establishing a theoretical upper bound on the performance of the LTP

Networks with Non-FIFO Link Travel Times

The proposed LTP algorithm was documented without assuming FIFO (first-in-first-out) links and is therefore not restricted to such networks. It produces uninterrupted paths that reach the destination node in the least possible time, without any waiting at any node. However, if waiting on some or all nodes were allowed for a limited or unlimited time, the LTP algorithm could easily be modified to calculate the least-time paths considering waiting on the nodes, too.

Networks with non-FIFO links may arise in modelling cases where the kinds of non linear travel time functions properly or improperly used by traffic modellers to relate average link trip time to the prevailing flow or density on a link may not necessarily generate trip times that satisfy the FIFO assumption (as exemplified in Smeed, 1967). Moreover, they arise in urban street networks with multimodal transportation opportunities, where the fastest "park and ride" or "transfer" option is sought, in which case waiting at a node may be meaningful since some waiting in the transfer or parking areas may be possible.

An example of a network with a non-FIFO link is shown in Figure 53, which is borrowed from Orda and Rom (1990). The continuous function of travel time $1+(5-t)^2$ for arc (3,4) was discretized as shown in Figure 53. The paths and the pointers to the subsequent nodes and time intervals are shown as vectors of labels on the nodes. So the first label of node 2, [0 8 3,2] means that the best path from node 2 to node 4 at time 0 points to node 3, at time interval 2.

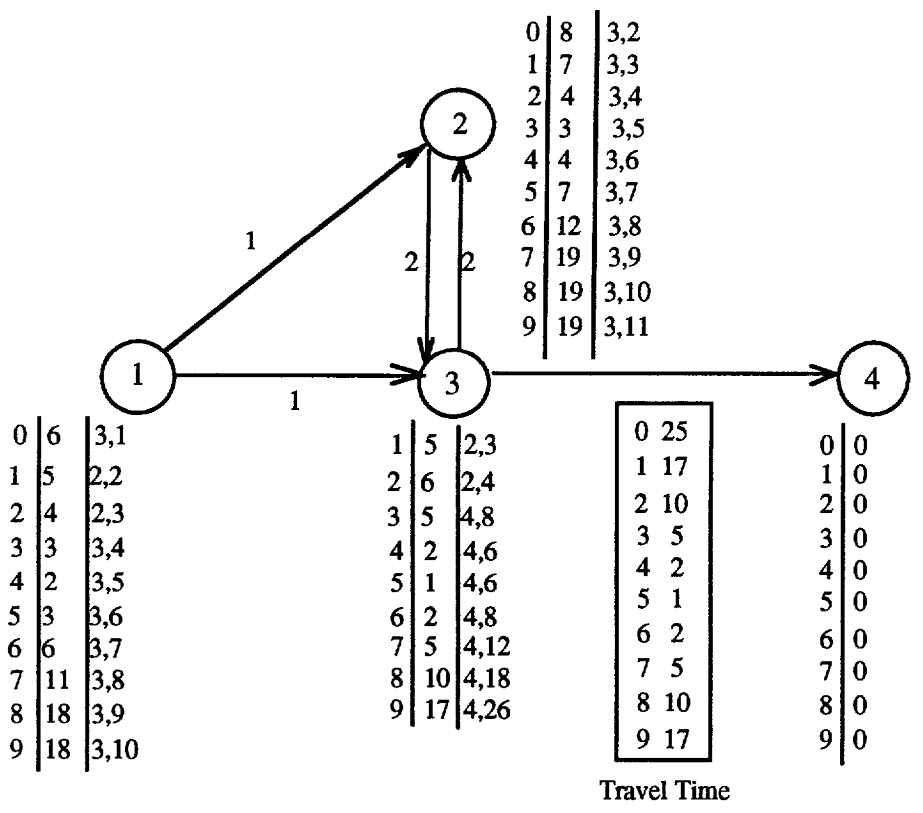


Figure 53. A Network Example with a Non-FIFO Link

The fastest path from node 1 to node 4 for departure time t=0 is $P_1=\{1,3,2,3,4\}$. This path contains a loop. *In general, if the FIFO property does not hold for all the links of a network, some best paths may contain loops.*

The FP algorithm efficiently solves this problem by discretizing the travel times. Bellman's principle of optimality is valid because the algorithm implicitly expands the network for every discrete time interval. Therefore, the subsolutions for every optimum solution are optimum, too; the best path to node 3 at time interval t=1 is $\{3,2,3,4\}$. Note here that the same example with continuous travel time function was identified as NP-complete by Rom and Orda (1990).

Another case for which the FIFO assumption does not hold in transportation networks is where the travel cost is not the travel time itself, but another arbitrary cost function of time. In this case, the links are traversed in a time according to a travel time function, but the cost incurred is given by some arbitrary function. This type of least cost path is essential in certain traffic modelling problems, especially in solving the time-dependent system optimal assignment problem described in Chapter 4. In the next section, a modification of the LTP algorithm is presented for such problems.

THE TIME-DEPENDENT LEAST COST PATH ALGORITHM

Problem Formulation

On the graph defined in the LTP algorithm formulation in the previous section define $c_{ij}(t)$ to be a real-valued cost function of time associated with arc (i,j). The $c_{ij}(t)$ is defined for every $t \in S$ and denotes the cost to traverse the arc (i,j). An example of such a cost in the dynamic traffic assignment problem the time-dependent marginal travel time, imposed on the whole system by an additional user on arc (i,j) at time t. Other examples of such travel costs are toll fees, fuel consumption, and pollutant emissions. We assume that $c_{ij}(t)$ is a non-negative number, although negative values that do not create negative cycles would be acceptable.

The LCP algorithm presented in this section calculates the time-dependent least-cost paths from every node i in the network and at every time interval t to the destination node D. At each step of the computation, let $\lambda_i(t)$ be the total travel time and $\eta_i(t)$ be the total travel cost of the current least-cost path from node i to node D at time t. Let $\Lambda_i = [\lambda_i(t_0), \lambda_i(t_0+\delta),..., \lambda_i(t_0+M\delta)]$ and $H_i = [\eta_i(t_0), \eta_i(t_0+\delta),..., \eta_i(t_0+M\delta)]$ be the M label vectors that contain all the labels $\lambda_i(t)$ and $\eta_i(t)$, respectively for every time step $t \in S$ for node i. Every finite label $\eta_i(t)$ corresponds to a path from node i to node D, which is identified by the ordered set of nodes $P_i=\{ i=n_1, n_2, ..., n_m=D \}$.

$\lambda_i(t)$ and $\eta_i(t)$ are defined according to the following functional equation:

$$\eta_j(t) = \min \{\eta_j(t), c_{ji}(t) + \eta_i(t+d_{ji}(t))\} \qquad j \in \Gamma^{-1}\{i\}$$

and

$$\lambda_j(t) = \begin{cases} d_{ji}(t) + \lambda_i(t+d_{ji}(t)) & \text{if } \eta_j(t) < c_{ji}(t) + \eta_i(t+d_{ji}(t)) \\ \lambda_j(t) & \text{otherwise} \end{cases} \qquad j \in \Gamma^{-1}\{i\}$$

The LCP algorithm is based on Equation (9). It operates in a label correcting fashion maintaining a SE list of nodes with some potential to improve the labels of at least one other node in a way similar to the LTP algorithm.

The Algorithm

Initially the SE list contains only the destination node D. The H- and Λ– label vectors of node D are set equal to zero, and for the rest of the nodes to infinity. In the first iteration all the nodes that can directly reach D are updated according to Equation 10, and inserted in the SE list.

$$\eta_i(t) = c_{iD}(t) + \eta_D(t+d_{iD}(t)) \qquad i \in \Gamma^{-1}\{D\}$$

and

$$\lambda_i(t) = d_{iD}(t) + \lambda_D(t+d_{iD}(t)) \qquad i \in \Gamma^{-1}\{D\}$$

where $\Gamma^{-1}\{D\}$ is the set of nodes that can directly reach D. Next, the first node i of the SE list is scanned according to Equation 9 for every time step $t \in S$. If at least one of the components of H_j is modified, node j is inserted in the SE list. This scheme is repeated until the SE is empty, and the algorithm terminates.

The steps of the LCP algorithm are summarized as follows:

Step 1. Create the SE list and initialize it by inserting into it the destination node D. Initialize the label vectors at the following values:

$\Lambda_D=(0, 0, ..., 0)$,

$H_D=(0, 0, ..., 0)$,

$\Lambda_i=(\infty, \infty, ..., \infty)$ and

$H_i=(\infty, \infty, ..., \infty)$ for $i \in V\backslash D$.

Step 2. Select the first node i from the SE list, name it "Current Node" and delete it from the list. If the SE list is empty, go to step 4.

Scan the current node i according to Equation 9, by examining each node j, $j \in \Gamma^{-1}\{i\}$. Specifically, for every time step t S do the following :

Check if $\eta_j(t)$ is greater than $c_{ji}(t)+\eta_i(t+d_{ji}(t))$. If it is, replace $\eta_j(t)$ in the label vector H_j at position t by $c_{ji}(t)+\eta_i(t+d_{ji}(t))$, and $\lambda_j(t)$ in the label vector Λ_j at position t by $d_{ji}(t)+\lambda_i(t+d_{ji}(t))$.

If at least one of the M H-labels of node j has been improved, insert node j in the SE list. The structure of the SE list and the associated operations of creation, insertion, and deletion are similar to those for the LTP algorithm.

Step 3. Repeat step 2.

Step 4. Terminate the algorithm. The M-dimensional vectors H_i and Λ_i for every node i in the network contain the travel cost and travel time of the time-dependent least-cost paths from every node i to the destination node D for each time step $t \in S$.

IMPLEMENTATION AND COMPUTATIONAL EXPERIENCE

Implementation of the LTP Algorithm.

The implementation of this algorithm is similar to the implementation of a static label correcting algorithm. The three principal implementation issues are: the network representation, the data structure of the SE list, and the path storage.

The network representation is more complicated than in the static case, because travel times need to be specified for every time step (M steps) for every arc. The most efficient way to store the network is the "backward star" structure, since at step 2 of the algorithm we need all the arcs that end at a specific node. The forward and backward star structures were described in the previous chapter. To handle the time-dependent trip times, we use the second dimension of the backward star to store pointers to an $|A|xM$ matrix, where $|A|$ is the number of arcs of the network. The required memory to store this structure is the minimum possible, $|N|+|A|(M+2)$ units.

The structure of the SE list for the label correcting algorithms has been extensively studied in the literature. Any SE list structure is appropriate for the proposed algorithm: a simple list with any priority rule, a queue, a double ended queue, as well as Glover et al.'s (1985) partitioning shortest path scheme with two SE lists. In this application, a double ended queue (deque) structure has been implemented. The deque structure allows the insertion of nodes at both ends of the SE list according to a predetermined strategy, and removal from the beginning of the SE list. The deque is implemented along the same lines described in the previous chapter for the k-shortest path algorithm. The Creation, Insertion, and Deletion operations are defined in the same way as well.

The paths are maintained in an Mx2-dimensional array of pointers for each node. These pointers point to the successor node and its label address. This arrangement requires 2|N|M memory locations, the least possible.

In pseudo-code form the algorithm is summarized below:

```
    Program Time Dependent Shortest Path
       Call Creation
       Call Insertion(N)
       Do 1, While (SE list is not Empty)
          Call Deletion(CurrentNode)
          Do 2, For (All nodes J that can directly reach CurrentNode)
            NextNode = J
            InsertInSEList.=FALSE
            Do 3, For (t=l,M)
              CurrentTravelTime=TravelTime(NextNode, CurrentNode,t)
              NewLabel=LABEL(CurrentNode,t+CurrentTravelTime)+CurrentTravelTime
              If (LABEL(NextNode,t) <NewLabel) Then
                  LABEL(NextNode,t)=NewLabel
                  InsertInSEList=TRUE
                  PathPointer(NextNode,t,1)=NodeCurrent
                  PathPointer(NextNode,t,2)=t+CurrentTravelTime
               EndIf
3           Continue
              If (InsertInSEList) Call Insetion(NextNode)
2        Continue
1     Continue

       Procedure Creation
          Do, For (Node= 1, V- 1)   Deque(Node)=0
          Deque(N)=999999
          FIRST=N
          LAST=N

       Procedure Deletion(CurrentNode)
          CurrentNode=FIRST
    FIRST=Deque(CurrentNode)
    Deque(CurrentNode)=-1
Procedure Insertion(Node)
    If (Deque(Node)=O) Then
        Deque(LAST)=Node
        LAST=Node
        Deque(Node)=999999
    Else
        If (Deque(Node)=- 1) Then
            Deque(Node)=FIRST
            FIRST=Node
        End If
    EndIf
```

Where:

LABEL(Node,t) is a variable that holds the M-vector labels for every node.

PathPointer(Node,t,1) is a pointer that points to previous node while PathPointer(Node,t,2) points to the corresponding time of arrival at the previous node of the shortest path from this node to the destination node N.

InsertInSEList is a logical variable which is used to determine if a label of a node was changed.

NewLabel is an auxiliary variable that temporarily holds the new label of the next node.

Note that all the variables are typed with lower size letters.

The most time consuming part of the algorithm is Step 2 (see Algorithm description), where each element of the M-vector is updated for every node adjacent to the scanned node. This Step corresponds to loop 2 in the pseudo-code, and requires 4Md computational time units, where d is the indegree of the scanned node (number of iterations of loop 2). The inner loop 3 can be efficiently vectorized because of the absence of inter-dependencies, and the number of iterations M is usually greater than 64 (the number of registers in the CRAY's vector processor) which leads to maximum vectorization speed-up.

The efficiency of the algorithm, however, depends essentially on the total number of scanned nodes before the process terminates. The lower bound on this number is the total number of nodes in the network (|N|), while an upper bound is |N|2M. The upper bound is obtained by direct extension of the results for the corresponding static label correcting case. As shown in the next section, this upper bound is not a tight bound in practical applications. The complexity of the algorithm is that of Step 2 (loop 2) multiplied by the upper bound on the number of repetitions of this Step (iteration number of loop l), or O(|N|3M2) in the general case that the maximum indegree of a node is |N|-1.

This implementation was coded in the FORTRAN CFI77 language, and run on a CRAY Y-MP/8 supercomputer. The results from the tests are presented in the next section.

Computational Experiments

Four different sets of networks are used to test this new algorithm. Set 1 consists of five random networks with structure similar to street networks and with the number of nodes ranging from 100 to 2500. The number of time steps is held constant at 240. The travel times for each time step are generated in such a way that the FIFO criterion holds for every link. Specifically, a randomly generated number is accepted as travel time for a given time step only if the absolute value of its difference with the travel time of the previous time step does not exceed the length of the time interval between the two steps. Set 1 was designed to test the relation of the performance of the algorithm to the network size.

Set 2 contains five different representations of the same random network consisting of 1000 nodes, 2500 arcs and varying numbers of time steps that range from 120 to 640. In set 3, the number of arcs ranges from 1000 to 11500, while both the number of nodes and the number of time steps are kept constant at 1000 and 240, respectively. This set is used to estimate the relationship between the execution time and the average degree of a node in a network.

Finally, set 4 consists of one real street network, that of the core area of Austin, TX, consisting of 625 nodes and 1742 arcs. Time-dependent travel times for this network were produced from DYNASMART for a simulated peak period of 50.3 minutes. This peak period is discretized into 503 time intervals of 0.1 minute each, and the travel time for each time interval is generated.

Tables 28 to 31 present the computation times in CPU milliseconds for each set. All runs were performed on a CRAY Y-MP/8 supercomputer, using the CFT77 FORTRAN compiler. This computer has eight CPU’s with vector pipeline architecture. The algorithm is coded in such a way as to allow vectorization when applicable. Vectorization is especially well suited for Step 2 of the algorithm, in which M iterations are performed, since no dependency exists between any two iterations and the number M is usually larger than the number that CRAY considers the minimum number of iterations for maximum speed-up. However, no attempt was made to exploit other hardware characteristics of the CRAY beyond vectorization. In addition, in order to smooth out the effect of the destination node choice on the execution time, thirty runs were performed for thirty different destinations for every network, and the average computation time is reported. It must be stressed, however, that the algorithm is not machine dependent. It can be applied on any computer, although it will perform better on a vector processor architecture.

Tables 28 and 29 contain the results for network sets 1 and 2; these results indicate that the computation time increases almost linearly with the number of nodes and the number of time steps in the network.

The results of Table 30, on the other hand, suggest a nonlinear relation between the execution time and the average degree of a node in the network. An exponential model was calibrated from these data using regression, yielding the following relationship:

$$\text{Computation Time}=22.13\ d1.4$$

where d is the average indegree of a node in the network.

Table 31 contains the averages and standard deviations of the computation time and the total number of scanned nodes for the real street network of the Austin, TX core area.

Table 28. Computation Times in Milliseconds, for Different Network Sizes

100 Nodes 250 Arcs 240 Time Int.	500 Nodes 1250 Arcs 240 Time Int.	1000 Nodes 2500 Arcs 240 Time Int.	1500 Nodes 5000 Arcs 240 Time Int.	2500 Nodes 8000 Arcs 240 Time Int.
5.97	35.73	73.28	141.42	235.04

Table 29. Computation Time in Milliseconds, for Different Number of Time Steps Varies

1000 Nodes 2500 Arcs 120 Time Int.	1000 Nodes 2500 Arcs 240 Time Int.	1000 Nodes 2500 Arcs 360 Time Int.	1000 Nodes 2500 Arcs 480 Time Int.	1000 Nodes 2500 Arcs 640 Time Int.
37.95	73.28	102.46	131.56	158.54

Table 30. Computation Time in Milliseconds, for a Network with 1000 Nodes, 240 Time Steps, and Varying Number of Arcs

1000 Nodes **2ooo Arcs** 240 Time Int.	1000 Nodes **3000 Arcs** 240 Time Int.	1000 Nodes **6000 Arcs** 240 Time Int.	1000 Nodes **9000 Arcs** 240 Time Int.	1000 Nodes 11500 Arcs 240 Time Int.
55.89	91.88	253.95	448.19	**624.86**

Table 31. Performance of the Algorithm on the Real Street Network

	Computation time in milliseconds	Total number of scanned nodes
Mean	107.41	**736**
St. Deviation	11.82	81

The total number of scanned nodes is the main factor that affects the performance of the algorithm. The lower bound for this number is the number of nodes in the network, |N|, while an upper bound was found to be |N|2 M in a previous section. The results in Table 31 show that for the tested network of 625 nodes, the total number of scannings was 736, or 1.18|N| which is considerably less than the theoretical upper bound. Moreover, from the low values of the standard deviations, it can be inferred that the algorithm is reasonably stable.

Combining the above results, we can conclude that as is common with shortest path problems, the actual computational performance for the networks considered here is on the order of INIMdl.4, which is far from the worst case theoretical complexity O(|N|3M2).
In addition, as mentioned previously, the algorithm vectorized efficiently. The algorithm was tested on the Austin core street network with the vectorization feature disabled for the same destination nodes as above, and the average execution time was found to be 728.02 milliseconds. This means that the vectorization in this case yielded a speed up of 6.74 (speedup is defined as the ratio of the total computation time of the algorithm without vectorization to the corresponding computation time with vectorization).

Next, we compare our approach to the expanded static case proposed by Dreyfus (1969). We implemented that scheme as efficiently as we would, and achieved an execution time of 2.2 milliseconds to find the time-dependent least-time path between one origin and one destination, for one time step on the Austin core network. In order to compare it to our proposed algorithm, the time-dependent least-time paths must be calculated from all 625 nodes of the network, to one destination and for 503 time steps for each node. This calculation would require a total time of 0.0022x625x503 = 691.25 seconds. However, the calculation of one path for one node and one time step produces at the same time the paths to the destination from every node along the path for one time step. The maximum number of nodes in a path for the real street network was 72. Therefore, we can estimate a lower bound on the total execution time, by assuming that every time one path is computed, 72 other not previously calculated paths are obtained at the same time. This lower bound is 9.6 seconds, which greatly exceeds the 0.107 seconds achieved with the proposed algorithm

The proposed algorithms take advantage of two main characteristics of networks with time-dependent arcs. One is that only few paths between a given OD pair become best paths at any point in time. Usually, three or four paths are interchanged as best paths at different time steps, one path often maintaining its best path status for most of the time. For example, the maximum number of paths that we observed during the testing of the real street network was seventeen (out of a possible 503).

The second characteristic of dynamic networks is that even if different paths were best at different times between a given OD, these paths would be likely to share the same next-to-the-origin node (i.e. second node in the path). This means that most of the best paths from a given node result from the scanning of just one of the neighboring nodes. The effectiveness of our algorithm is attributed to these two reasons. Specifically, the fewer the paths that are best at different times, the closer is the behavior of the algorithm to that of static label correcting algorithms. In the extreme case that the same path remained best between a given OD pair for all the time steps, the corresponding origin node would contribute to the total computation time of the algorithm as if the network were static. Even if more than one path were best for a given OD pair, these paths could be calculated in just one scanning of a neighbor node. From Table 3 1, we can see that for the real street network of Austin, TX only 111 (736-625) nodes were scanned for a second time, although in general different paths were best at different times for a given origin node.

CHAPTER 8
CONCLUSION

The procedures presented in this report constitute a major advance in the state-of-the-art of dynamic assignment and traffic simulation-assignment for large-scale networks, especially in connection with ATIS/ATMS applications. The success of ATIS/ATMS depends on the ability to use the real-time information available to the central controller for real-time routing and traffic control decisions. The procedures developed in this study provide and essential component in the overall decision-support methodologies needed for ATIS/ATMS planning and operation.

The framework and algorithms presented in this report provide a flexible and modular modelling capability that can evolve as the results of future research, in areas such as tripmaker behavior and response to information become available. The procedures can also be interfaced with traffic control modules as well as other ATMS support functions in the context of an overall ATMS architecture.

To summarize, the following major procedures were developed in this study:

1. DYNASMART: A descriptive simulation-assignment framework, that meets or exceeds all functional requirements set forth in the statement of work. Its purpose is to predict the temporal and spatial patterns of flows in a network for given time-dependent origin-destination trip desires, network characteristics including traffic control schemes, and information supply strategies, as summarized in Figure 54.

2. System Optimal Single Class Dynamic Traffic Assignment Algorithm: The purpose of this algorithm is to determine the paths to which all vehicles should be directed to, given time-dependent O-D trip desires and prevailing traffic control scheme, so as to minimize overall user cost (time) in the network over the duration of interest. This algorithm consists of an iterative procedure, where DYNASMART is used as a simulator to represent traffic propagation in the network for a given time-dependent path assignment, as summarized in Figure 55.

3. User Equilibrium Single Class Dynamic Traffic Assignment: This algorithm is a variant of the SO version. It solves for a time-dependent assignment of vehicles to specific paths, between each O-D pair, for given time-dependent O-D trip desires, so that no user can reduce his/her travel time by unilaterally switching paths for the same departure time. DYNASMART is again used as a simulator within the iterative search procedure, as summarized in Figure 56.

4. Multiple User Classes Dynamic Traffic Assignment Algorithm: This algorithm recognizes several classes of network users on the basis of information supply and user behavior, including: SO users, UE users, boundedly rational users, and those with pre-specified paths. It solves simultaneously for the paths to which SO users should be directed to so as to minimize overall travel time in the network, as well as the paths which UE users will follow to satisfy their requirements, recognizing that some users may be following their own heuristic path selection rules while others may just have to be viewed as externally specified. This algorithm is only possible because of the flexibility of DYNASMART to consider all these classes. It is summarized in Figure 57.

5. Rolling Horizon Implementation Framework for SO Single Class and MUC Algorithms: This Framework is intended for quasi real-time implementations of the assignment algorithms in situations where the time-dependent O-D trip desires cannot be assumed known a priori with sufficient reliability. It is intended as the principal manner in which the DTA algorithms would be used in real-time as an ATMS support function.

The above procedures are rather complex entities that incorporate a large number of components and modules. Among those, major development was necessary as part of this study in the area of path processing algorithms, described in Chapters 6 and 7. In particular, the following path processing procedures have been developed:

1. K-Shortest Path (K-SP) Computation: This algorithm is used in the path processing component of DYNASMART to solve for the K best paths from all origin nodes to each given destination node for a given set of link travel times.

2. K-Shortest Path Update: This algorithm updates the travel times and re-sorts a set of K stored paths; it is used in the path processing component of DYNASMART to update the paths obtained using the K-SP computation algorithm in the time intervals between successive computations (see Chapter 6 for detailed discussion of the logic for this strategy).

3. K-SP Computation for Multiple User Classes: Special version of the K-SP computation algorithm that guarantees the best K; paths are found for each vehicle class I, i = l,...,U (U is the number of classes). It is used instead of the single class algorithm in the path processing component of DYNASMART in the presence of multiple vehicle classes.

4. K-SP Update for Multiple User Classes: Special version of the single class update algorithm for use in conjunction with multiple vehicle classes in DYNASMART.

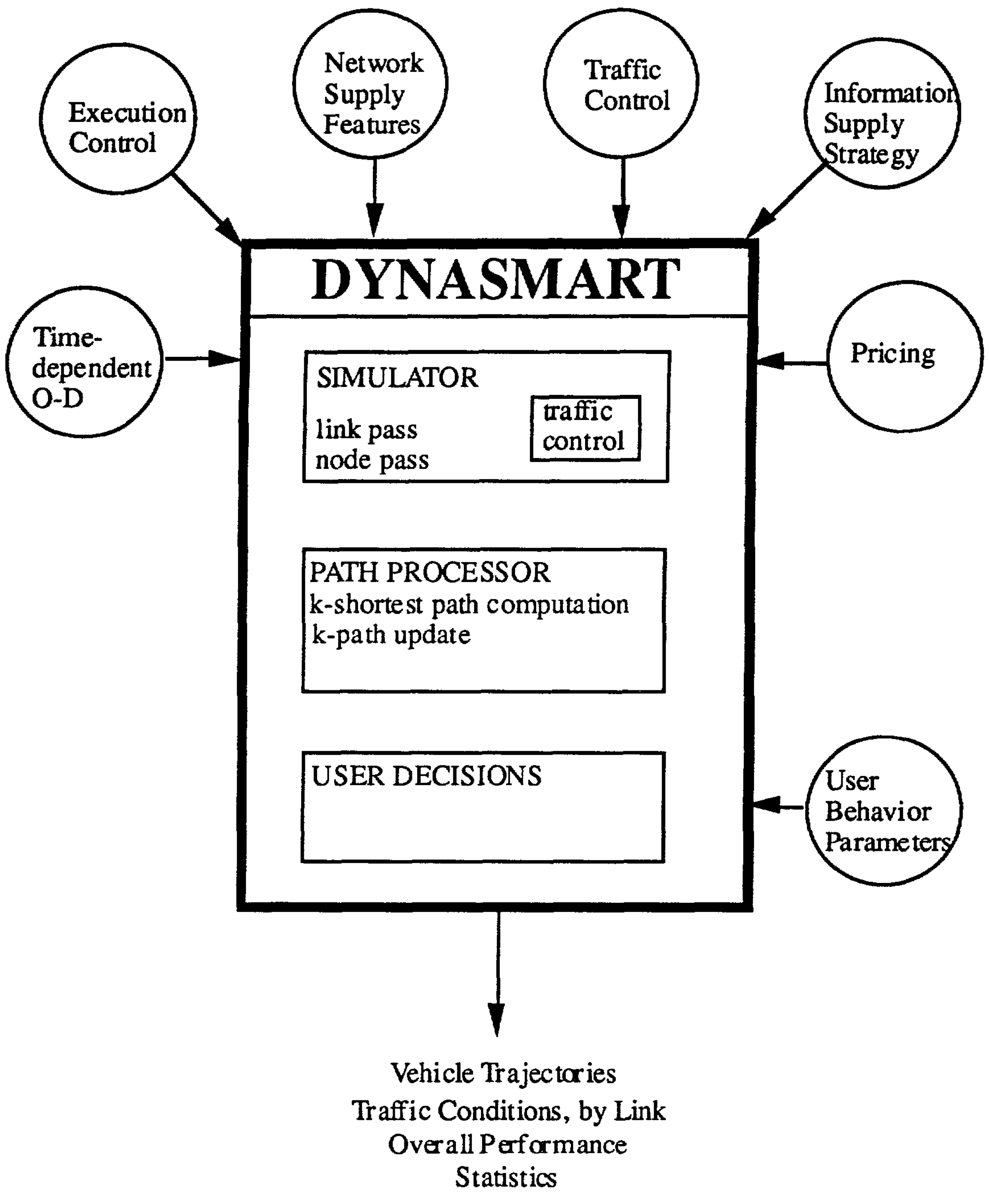
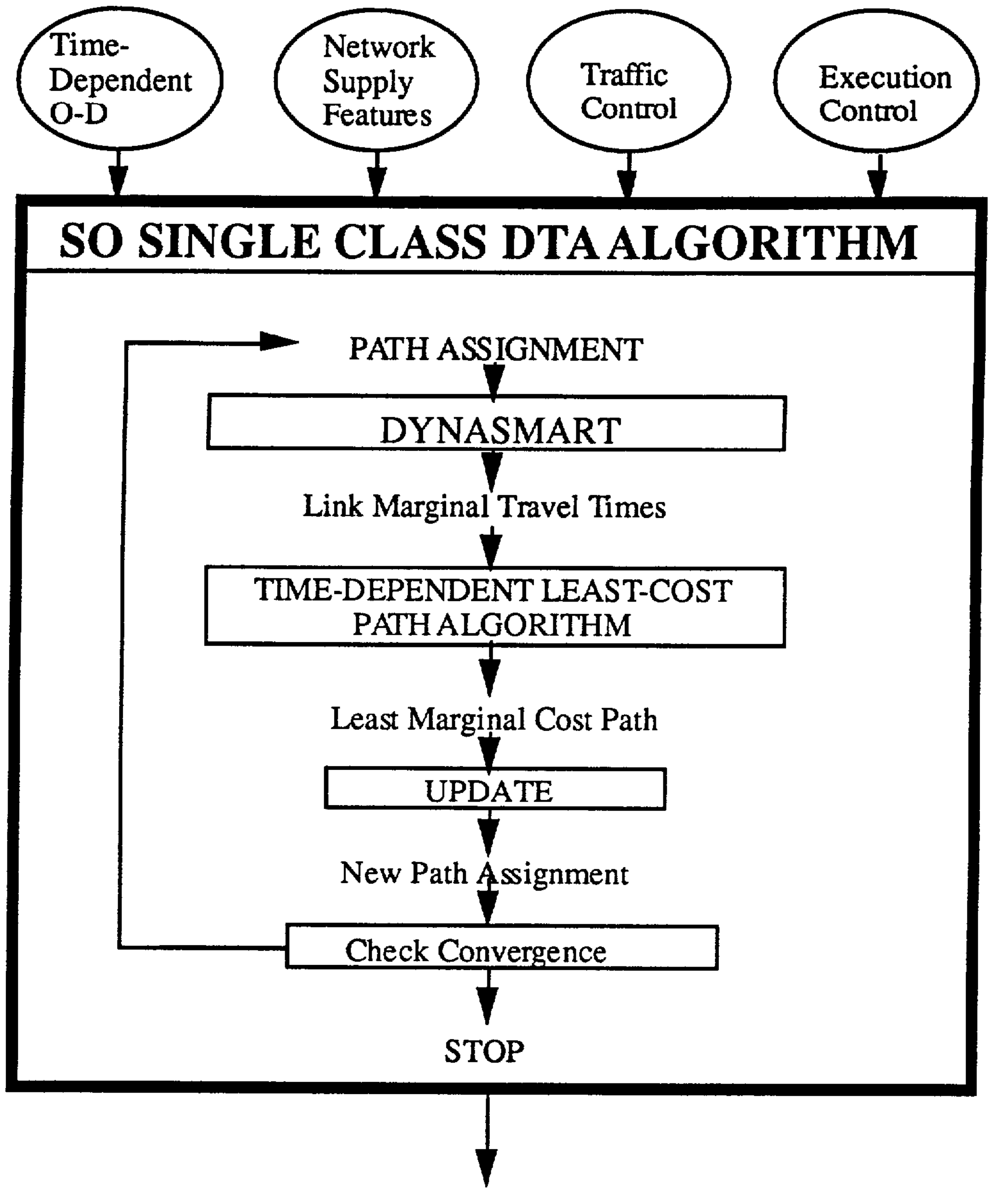


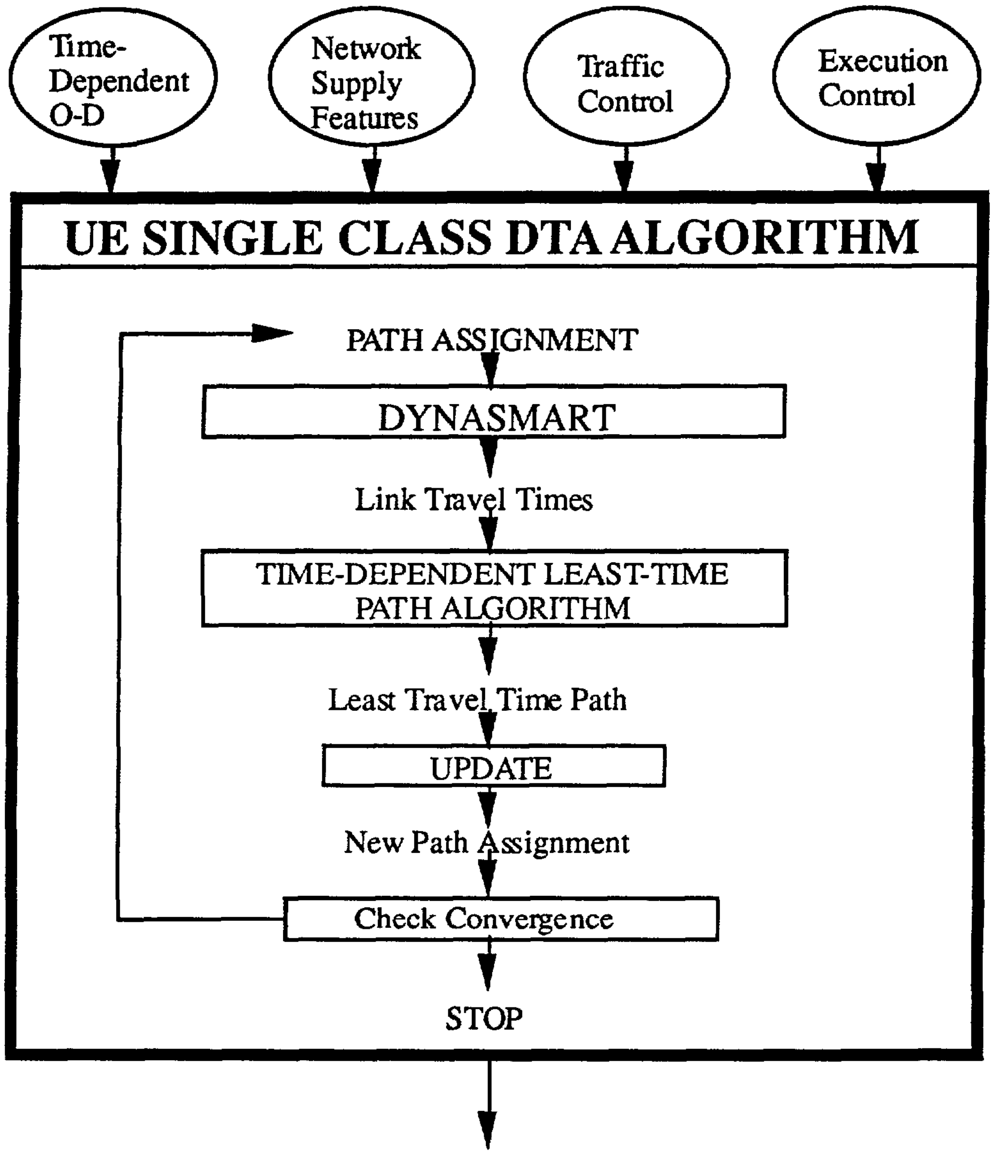
Figure 54. Summary of DYNASMART Simulation-Assignment Framework



Path Assignment: Number of Vehicles on Each Path for Each O-D for Each Assignment Interval

Specific Path for Each Departing Vehicle for Given O-D at Given Time

Figure 55. Summary of System Optimal Single Class Dynamic Traffic Assignment Algorithm



Path Assignment: Number of Vehicles on Each Path
for Each O-D for Each Assignment Interval

Specific Path for Each Departing Vehicle for Given O-D
at Given Time

Figure 56. Summary of User Equilibrium Single Class Dynamic Traffic Assignment Algorithm

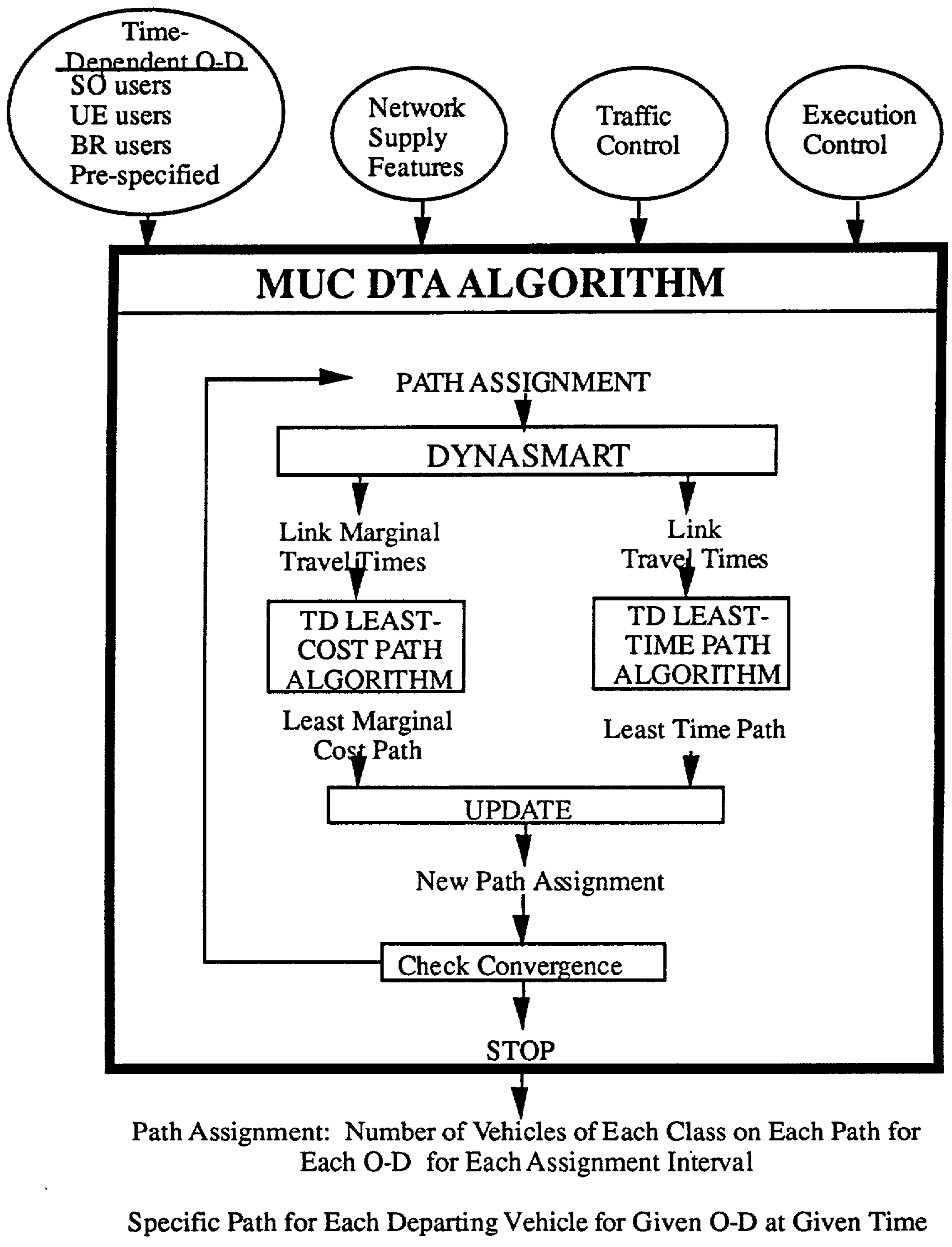

Time-Dependent O-D
SO users
UE users
BR users
Pre-specified
Network Supply Features
Traffic Control
Execution Control
MUC DTA ALGORITHM
PATH ASSIGNMENT
DYNASMART
Link Marginal Travel Times
Link Travel Times
TD LEAST-COST PATH ALGORITHM
TD LEAST-TIME PATH ALGORITHM
Least Marginal Cost Path
Least Time Path
UPDATE
New Path Assignment
Check Convergence
STOP
Path Assignment: Number of Vehicles of Each Class on Each Path for Each O-D for Each Assignment Interval
Specific Path for Each Departing Vehicle for Given O-D at Given Time

Figure 57. Summary of Multiple User Classes Dynamic Traffic Assignment Algorithm

5. Time-Dependent Least Time Path Algorithm: This algorithm computes the least-time path from all origin nodes to a given destination node for all departure time intervals (from the origin) for a given network with time-dependent link travel times. This algorithm is used in UE single user class DTA algorithm, as well as in the MUC DTA algorithm, where it receives its input data from DYNASMART.

6. Time-Dependent Least Cost Path Algorithm: This algorithm computes the least cost path from all origin nodes to a given destination, for given time-varying costs on the network links. These costs are different from the travel times per se. In this study, they consist of the marginal trip times, in the context of the SO single user class DTA algorithm and the MUC DTA algorithm.

Several aspects of the procedures developed here can benefit from additional investigation. In addition to the various modelling elements that could be improved through further field observations, guidelines pertaining to the choice of various user-controlled parameters in the dynamic assignment process are necessary. These include issues of the appropriate length of the assignment interval relative to the simulation interval, frequency of path updates, assumptions made in the rolling horizon implementation and the sensitivity to various modelling and execution assumptions described in connection with the procedures in question. These issues will form the basis of continuing investigation that builds on the foundation developed in this study.

APPENDIX A

DESCRIPTION OF INPUT AND OUTPUT DATA FILES

DESCRIPTION OF INPUT AND OUTPUT DATA FILES

This Appendix documents the input data set and output data files in the DYNASMART network assignment-simulation model. This description is intended primarily to illustrate the kind of information input to DYNASMART, and the kinds of reports its produces. This description is not meant to be a self-contained user's guide. The input and output file nomenclature applies primarily to the implementation of DYNASMART on the University of Texas' CRAY Y-MP computer. Because it is written in CRAY FORTRAN (CFT 77), this version has been found to be highly portable to other environments, including CONVEX C220, IBM RISC/6000, and SUN SPARC II workstations with only very minor modifications. However, some of the output file structure may need to be modified for implementations on different platforms. .

The current version of DYNASMART consists of 20 subroutines, and its program structure is very flexible for adding other functions, such as different behavioral rules and different vehicle classes. DYNASMART can also be applied in different situations. For example, a modified version of DYNASMART is used as the simulator in the SO, UE, and MNC dynamic traffic assignment algorithms.

INPUT DATA DESCRIPTION

Data sets and parameters used in DYNASMART cover transportation planning, traffic simulation, traffic control and user behavior components. This section illustrates these data sets and parameters.

1. Network data

The data set is used to define network configurations.

A. Basic Data

NZONES : number of zones in the network

NNODES : number of nodes in the network

NARCS : number of arc-chains

N : number of links (segments)

NDESTS : number of destinations

KAY : "K" for the K-shortest path

B. Zonal data

IPZ(l..NZONES) : Demand zone numbers; each demand zone must have a unique identification number

C. Destinations

IDZ(l..NZONES) : Each zone must have a centroid as a destination. The destination node is a network node, and must be within this demand zone. The order is the same as zonal data. There will be zeros for some zones, which means they do not have destination nodes.

D. The Mapping between demand zones and network nodes

IZONE(network node) = demand zone number

For each network node, there must be an associated demand zone number.

E. Link data

IUNOD : upstream node

IDNOD : downstream node

LENGTH : Length of the link (in feet)

14 : 0 : no generation

1 : volume from the zone of the upstream node

2 : volume from the zone of the downstream node

NLANES : number of lanes

VMAX : the maximum velocity (miles/minute)

SAT : saturation flow rate (vehicles/second)

LINK_IDEN : link identification

1 : freeway link

2 : freeway segment with detector

3:onramp

4 : off ramp

5 : arterial

6 : HOV lane

BAY : left-turn bay

Example:

```
  10    50   168   168    10    5
   1     2     3     4     5    6     7     a     9    lo
   2     5    13    18    25   30    35    36    37    44
   1     1
   2     1
   3     1
   4     2
   5     2
   6     2
   7     3
   a     1
   9     0
  10     0
   1     2 2640  2  2  0.333  0.50  5   0
 . 1     7 2640  2  2  0.333  0.50  5   0
   2     12640  2  2  0.333  0.50  5   0
   2     3 2640  2  2  0.333  0.50  5   0

   2     8 2640  2  2  0.333  0.50  5   0
```

2. Demand Data

A flexible dynamic demand input format is used in DYNASMART. Users need to define the number of loading intervals and associated time periods. For each period, an OD matrix (zone-to-zone) needs to be prepared in order to generate and load vehicles into the network.

Example:

```
  7
0.0  5.0  10.0 15.0  20.0  25.0  30.0  35.0
    0.0       1.0       1.0       1.0       1.0       1.5
    1.0       1.0       1.0       1.0
    1.5       0.0       1.0       1.0       1.0       1.5
    1.0       1.0       1.0       1.0
    1.5       1.0       0.0       1.0       1.0       1.5
    1.0       1.0       1.0       1.0
    1.5       1.0       1.0       0.0       1.0       1.5
    1.0       1.0       1.0       1.0
    1.5       1.0       1.0       1.0       0.0       1.5
    1.0       1.0       1.0       1.0
    1.5       1.0       1.0       1.0       1.0       0.0
    1.0       1.0       1.0       1.0
```

Alternatively, DYNASMART accepts input data directly in the form of a vehicles file, containing O-D information, and possibly a pre-specified initial path, for each generated vehicle. This capability is used primarily in conjunction with the SO, UE and MUC dynamic assignment algorithms.

3. Scenario input data

In this data set, users can define various parameters and variables describing a particular scenario for a simulation run. Below is an example list of such parameters used in some of our experiments. It is expected that additional parameters will be included in this file for specific applications.

RIBFA : relative indifference band
BOUND : threshold bound (minutes)
RNUMBER : a multiplication factor for demand generation (load factor)
ISEED : random number seed (123457)
IPINIT : index for initial routes
    0 : randomly select from K paths
    1: select best current path at origin
COM_FRAC : fraction of compliant vehicles
NTTO : maximum simulation length (in minutes)
TII : simulation interval (in minutes)
INDEX-SIG : index for signal control
    0 : deactivate signal control
    1 : keep signal control
CLASS and CLASSPRO : define multiple user classes for System Optimal case
    1 : vehicles with prescribed paths
    2 : vehicles with SO paths
    3 : vehicles with UE paths
    4 : vehicles with real time information
CLASS2 and CLASSPR02 : define multiple user vehicle classes in DYNASMART
    1 : non-equipped vehicles, passenger car (PC)
    2 : non-equipped vehicles, truck
    3 : non-equipped vehicles, high occupancy vehicles (HOV)
    4 : equipped vehicles, PC
    5 : equipped vehicles, truck
    6 : equipped vehicles, HOV
    6 : buses
OPTIONS: to activate various functions

Example:

```
0.00 0.00 0.00 1.00      123457 1
 0.5
  120 0.1
```

```
    1  : index for signal control
    2 :  index for left capacity
    1 :  option for left turn test
    0  : option for bay (1: bay, 0 w/o bay 1
    1 :  option for vms
    1  : option for bus operation
     4
0.00 0.00 0.00 0.00
     6
0.5   0    0    0.5    0.00    0.00
```

4. Signal control data

In this file, users need to define the control type and parameters for each intersection in the surface street network.

A. Signal Node

NODE(I,l) : node number

NODE(I,2) : control type

1 : no control

2 : yield sign

3 : stop sign

4 : pretimed control

5 : actuated signal control

NODE(I,3) : phase number, needed only for control types 4 and 5

NODE(I,4) : cycle length, needed only for control types 4 and 5

B. Green time allocation

ITMP : node number

NSIGN(I,l) : phase number

Pretimed control (actuated signal control data)

NSIGN(I,2) : offset (the maximum green time)

NSIGN(I,3) : green time (the minimum green time)

NSIGN(I,4) : amber time

NSIGN(I,5) : number of inbound links in this phase

NSIGN(I,6: 11) : associated links' number

C. Movement for each approach for each phase

GMOVE(I,phase number, l-8)

Example:

```
   1 1 2     6 0
   2 4 2     6 0
   3 4 2     6 0
   4 4 2     6 0
   5 4 2     6 0
   6 1 2     6 0
   7 4 2     6 0
   a 5 2     6 0
 2     1     0     25     5     2     1     3
 2     2     0    25     5     2    44     a
 3     1     025      5     2     2     9
 3     2     025      5     2     2     4
 4     1     025      5     2     3     5
 4     2     0    25     5     2    10    45
```

51		0	25	5	2	44	11
5	2		025	5	2	4	6
7	1	0	2 5	5	2	1	13
7	2	0	25	5	2	1	8
a	1	25	10	5	2	2	14
a	2	25	10	5	2	7	9

5. ramp control data

In this file, ramp and variable message signs are specified, especially for freeway operations.

DEC_NUM : detector number for ramp control

VMS_NUM : Number of VMS

HOV-OCCP : % of high vehicle occupancy

A. Ramp Data

Detector ID number

From Node

To Node

Upstream Position (feet)

Downstream Position (feet)

From node of controlled ramp

To node of controlled ramp

CONS1 : ramp parameter 1

CONS2 : ramp parameter 2

Ramp Rate : (vehicles/second)

B. VMS data

VMS ID number (a number is assigned for each VMS)

Type : 1 : speed advisory

2 : route advisory

3. congestion warning

Location : From node and To Node

Parameters for each type :

1. the speed reduction factor (speed limit, suggested speed)
2. the specific route in K-shortest paths

   (the assigned path number, destination number)
3. k% of vehicles : divert to other paths

   (k factor, the path assigned)

Example Data for Ramp Metering

```
 11
  1    45    44   260   250    10    45   .32    .20  0.50
  2    46    45   260   250    16    46   .320   .20  0.50
  3    47    46   260   250    22    47   .320   .20  0.50
  4    48    47   260   250    28    48   .320   .20  0.50
  5    49    48   260   250    33    49   .320   .20  0.50
  6    50    49   260   250    36    50   .320   .20  0.50
```

Example Data for VMS

3

1	47	46	6	6
2	23	17	1	0
3	19	13	5	3

6. Movement data

This file needs to be prepared for left-turn operation. Movements are defined for every link, and the format is as follow:

From Node
To Node
Left_Turn Node
Straight Node
Right Node
Other

Example:

1	2	4 4	3	a	
1	7	a	13		
2	1		7	0	0
2	3	0	4	9	43
2	a	9	14	7	
2	44		5		43
3	2	a	1	44	
3	4		5	10	

7. Incident data

The data for each incident includes the start and end times of the incident and its severity. This information is pre specified for a particular scenario evaluated by DYNASMART. However, the data can be readily modified in a real time execution. If a link is closed, all the vehicles will be rerouted after reaching the switching point, i.e. the upstream node of the link.

INCI_NUM	: total number of incidents
INCI(i,l)	: link number
INCI(i,2)	: start time of incident i
INCI(i,3)	: end time of incident i
INCI(i,4)	: severity of incident i

Example:

```
 2
1  2  0.0  5.0  0.5
2  3  0.0  5.0  0.5
```

8. Bus data

The input data for bus operation includes :

BUS ID : au identifier for bus

Start Time : the start time of the bus

Average dwell time

Number of Nodes in the route

The sequence of Nodes

Operation Index : 0 : no stop

1: stop at the near block

2: stop at the midblock

3: stop at the midblock bus bay

Example:

```
 3
 1  1  2  1.0  1.0  6
 2    a   14   20   26   25
 0    1    2    1    2    0
 2  18  24  1.0  1.0  a
24   30   29   34   33   32   31   25
 0    1    2    1    2    0    0    0
 3  26  31  1.0  1.0  7
31   25   19   13    7    1    2

 0      12      12      0      0
```

DESCRIPTIONS OF OUTPUT FILES

Output files are generated for each DYNASMART run. Although a wide variety of output information can be obtained from DYNASMART, it is not efficient to produce all the files due to the computational cost. Since fort.4 and fort.6 provide summary information of a simulation run, it is recommended that at least these two output files be produced. The numbers from 1 to 99 can be used as input and/or output file units on CRAY, so the description of these files follows this sequence.

File Organization

fort.4 : simulation process monitoring unit

fort6 : title and the summary information

fort, 18 : vehicle trajectory

fort 16 : ALINEA ramp metering unit

fort.30 . . . fort.39: Link Information Output Units

```
      c  --
      c  -- for the purpose of post data analysis
      c  --some link information will be stored in
      c  -- different files in order to perform detailed
      c  -- analysis
      c  -- fort.30 : generation volume
      c  -- fort.31 : volume on links
      c  -- fort.32 : vehicle-queue
      c  -- fort.33 : velocity
      c  -- fort.34 : concentration
      c  -- fort.35 : velocity for the moving vehicles
      c  -- fort.36 : concentration for the moving part
      c  -- fort.37 : reserved for latter use
      c  -- fort.38 : greentime for eachapproach
      c  -- fort.39 : number of vehicles crossing intersection
      c  --
```

fort.40 . . fort.49: Input Data Units

fort.90 . . fort.99: reserved for debugging work

File Listing

```
total 15243
drwx------    2 cedr132  cedr         4096 Jan 25 15:57 ./
drwxr-xr-x   43 cedr132  cedr         4096 Jan 25 16:18 ../
-rw-------    1 cedrl32  cedr           67 Jan 25 15:57 fort.10
-rw-------    1 cedrl32  cedr           67 Jan 25 15:57 fort.15
-rw-------    1 cedrl32  cedr         6206 Jan 25 15:57 fort.16
-rw-------    1 cedrl32  cedr     30591530 Jan 25 15:57 fort.17
-rw-------    1 cedrl32  cedr      5475128 Jan 25 15:57 fort.18
-rw-------    1 cedr132  cedr           97 Jan 25 15:57 fort.2
-rw-------    1 cedr132  cedr     13651342 Jan 25 15:57 fort.20
-rw-------    1 cedrl32  cedr           67 Jan 25 15:57 fort.21
```

```
-rw-------   1 cedr132  cedr       102893 Jan 25 15:57 fort.22
-rw-------   1 cedr132  cedr      4919024 Jan 25 15:57 fort.27
-rw-------   1 cedr132  cedr      1275264 Jan 25 15:57 fort.28
-rw-------   1 cedr132  cedr           67 Jan 25 15:57 fort.37
-rw-------   1 cedr132  cedr           67 Jan 25 15:57 fort.39
-rw-------   1 cedr132  cedr        67553 Jan 25 15:57 fort.4
-rw-------   1 cedr132  cedr        12027 Jan 25 15:57 fort.51
-rw-------   1 cedr132  cedr           67 Jan 25 15:57 fort.52
-rw-------   1 cedr132  cedr         2033 Jan 25 15:57 fort.6
-rw-------   1 cedr132  cedr           67 Jan 25 15:57 fort.61
-rw-------   1 cedr132  cedr           67 Jan 25 15:57 fort.7
-rw-------   1 cedr132  cedr           67 Jan 25 15:57 fort.8
-rw-------   1 cedr132  cedr           67 Jan 25 15:57 fort.9
-rw-------   1 cedr132  cedr         8798 Jan 25 15:57 fort.91
-rw-------   1 cedr132  cedr        19658 Jan 25 15:57 fort.92
-rw-------   1 cedr132  cedr      5576118 Jan 25 15:57 fort.93
-rw-------   1 cedr132  cedr           32 Jan 25 15:57 fort.94
-rw-------   1 cedr132  cedr           32 Jan 25 15:57 fort.95
-rw-------   1 cedr132  cedr          129 Jan 25 15:57 fort.96
-rw-------   1 cedr132  cedr           32 Jan 25 15:57 fort.97
-rw-------   1 cedr132  cedr       481160 Jan 25 15:57 fort.98
-rw-------   1 cedr132  cedr         6063 Jan 25 15:57 fort.99
```

fort.4 - intermediate output file for monioring the simulation process

IT : simulation time step

VEHICLES : total number of vehicles generated

TAGGED IN : Number of tagged vehicles in the network

OUT-N : Number of non-tagged vehicles that have reached their destination

OUT-T : Number of tagged vehicles that have reached their destination

```
IT:     1  VEHICLES:     0  TAGGED IN:     0  OUT-N:     0  OUT-T:     0
IT:     2  VEHICLES:     2  TAGGED IN:     0  OUT-N:     0  OUT-T:     0
IT:     3  VEHICLES:    44  TAGGED IN:     0  OUT-N:     0  OUT-T:     0
IT:     4  VEHICLES:   122  TAGGED IN:     0  OUT-N:     0  OUT-T:     0
IT:     5  VEHICLES:   191  TAGGED IN:     0  OUT-N:     0  OUT-T:     0
IT:     6  VEHICLES:   242  TAGGED IN:     0  OUT-N:     0  OUT-T:     0
IT:     7  VEHICLES:   309  TAGGED IN:     0  OUT-N:     0  OUT-T:     0
IT:     8  VEHICLES:   379  TAGGED IN:     0  OUT-N:     1  OUT-T:     0
IT:     9  VEHICLES:   427  TAGGED IN:     0  OUT-N:     2  OUT-T:     0
IT:    10  VEHICLES:   502  TAGGED IN:     0  OUT-N:     3  OUT-T:     0
IT:    11  VEHICLES:   571  TAGGED IN:     0  OUT-N:     5  OUT-T:     0


IT:    50  VEHICLES:  3027  TAGGED IN:     0  OUT-N:   531  OUT-T:     0
IT:    51  VEHICLES:  3074  TAGGED IN:     0  OUT-N:   560  OUT-T:     0
IT:    52  VEHICLES:  3195  TAGGED IN:   121  OUT-N:   582  OUT-T:     0
IT:    53  VEHICLES:  3297  TAGGED IN:   223  OUT-N:   615  OUT-T:     0
IT:    54  VEHICLES:  3410  TAGGED IN:   336  OUT-N:   652  OUT-T:     0
IT:    55  VEHICLES:  3508  TAGGED IN:   434  OUT-N:   717  OUT-T:     0
IT:    56  VEHICLES:  3616  TAGGED IN:   542  OUT-N:   757  OUT-T:     0
IT:    57  VEHICLES:  3718  TAGGED IN:   641  OUT-N:   792  OUT-T:     3
IT:    58  VEHICLES:  3837  TAGGED IN:   758  OUT-N:   824  OUT-T:     5
IT:    59  VEHICLES:  3931  TAGGED IN:   851  OUT-N:   851  OUT-T:     6
IT:    60  VEHICLES:  4071  TAGGED IN:   988  OUT-N:   883  OUT-T:     9
IT:    61  VEHICLES:  4079  TAGGED IN:   993  OUT-N:   910  OUT-T:    12
IT:    62  VEHICLES:  4226  TAGGED IN:  1139  OUT-N:   944  OUT-T:    13
IT:    63  VEHICLES:  4366  TAGGED IN:  1278  OUT-N:   996  OUT-T:    14
IT:    64  VEHICLES:  4460  TAGGED IN:  1367  OUT-N:  1062  OUT-T:    19
IT:    65  VEHICLES:  4562  TAGGED IN:  1468  OUT-N:  1106  OUT-T:    20
IT:    66  VEHICLES:  4652  TAGGED IN:  1557  OUT-N:  1149  OUT-T:    21
IT:    67  VEHICLES:  4789  TAGGED IN:  1685  OUT-N:  1195  OUT-T:    30
IT:    68  VEHICLES:  4887  TAGGED IN:  1775  OUT-N:  1241  OUT-T:    38

IT:   901  VEHICLES: 26214  TAGGED IN:    18  OUT-N:  6216  OUT-T: 19908
IT:   902  VEHICLES: 26231  TAGGED IN:    18  OUT-N:  6216  OUT-T: 19908
IT:   903  VEHICLES: 26231  TAGGED IN:    18  OUT-N:  6221  OUT-T: 19908
IT:   904  VEHICLES: 26233  TAGGED IN:    17  OUT-N:  6224  OUT-T: 19909
IT:   905  VEHICLES: 26233  TAGGED IN:    16  OUT-N:  6225  OUT-T: 19910
IT:   906  VEHICLES: 26233  TAGGED IN:    14  OUT-N:  6225  OUT-T: 19912
IT:   907  VEHICLES: 26233  TAGGED IN:    13  OUT-N:  6225  OUT-T: 19913
IT:   908  VEHICLES: 26233  TAGGED IN:    13  OUT-N:  6228  OUT-T: 19913
IT:   909  VEHICLES: 26233  TAGGED IN:    12  OUT-N:  6230  OUT-T: 19914
```

```
IT:    910 VEHICLES: 26233  TAGGED IN:     11  OUT-N:   6234 OUT-T: 19915
IT:    911 VEHICLES: 26233  TAGGED IN:      9  CUT-N:   6235 OUT-T: 19917
IT:    912 VEHICLES: 26233  TAGGED IN:      9  OUT-N:   6236 OUT-T: 19917
IT:    913 VEHICLES: 26233  TAGGED IN:      7  OUT-N:   6243 OUT-T: 19919
IT:    914 VEHICLES: 26233  TAGGED IN:      7  OUT-N:   6244 OUT-T: 19919
IT:    915 VEHICLES: 26233  TAGGED IN:      6  OUT-N:   6244 OUT-T: 19920
IT:    916 VEHICLES: 26257  TAGGED IN:      6  OUT-N:   6244 OUT-T: 19920
IT:    917 VEHICLES: 26257  TAGGED IN:      6  CUT-N:   6246 OUT-T: 19920
IT:    918 VEHICLES: 26257  TAGGED IN:      4  CUT-N:   6246 OUT-T: 19922
IT:    919 VEHICLES: 26257  TAGGED IN:      4  CUT-N:   6248 OUT-T  19922
IT:    920 VEHICLES: 26257  TAGGED IN:      1  CUT'-N:  6253 OUT-T: 19925
    661   711   707   644   711   684   683   675   696   707
    623   648   740   649   694   672   673   697   680   681
    656   734   686   711   674   715   723   694   705   666
    696   720   697   671   662   746   703     0     0     0
      0     0     0   684
TOTAL GONE OUT= 26179
```

fort.6 - major output file for system performance

Overall information for system performance is included in fort.6, and error message (if any) will be shown in this file.

EXAMPLE :

```
                 ****************************************************
                *   DYNASMART V1.0: Dynamic Network Assignment *
                *                     & Simulation Model          *
                *    Developed in                                 *
                *        The University of Texas at Austin        *
                *               March 20, 1994                    *
                ****************************************************
 ****************************************
 *      Basic Information               *
 ****************************************

 NETWORK DATA
        NODES : 50
       LINKS : 168
       DESTS : 10
       ZONES : 10

 SIGNAL  DATA
        NO CONTROL         16
        YIELD SIGNS        0
        STOP  SIGNS       : 0
        PRETIMED CONTRO   : 26
        ACTUATED CONTROL : 8

 RAMP  DATA
        RAMP CONTROL : 0

 ASSUMPTION

 LEFT_OPTION : 1

 PATH K : 5
       TIME for UPDATING : 3.
 INITIAL  ROUTES
       1. assign the best path
       2. Use current path information
 -_-------------------------------------
    Multiple User Class Percentages
 ---------------------------------------
  Prescribed Paths : 0.
  so         Paths : 0.
  UE         Paths : 0.
  REAL TIME Paths : 0.
 ---------------------------------------
    DYNASMART Multiple User Class Percentages
 ---------------------------------------
  Non-equipped PC     : 0.5
               TRUCK : 0.
               HOV    : 0.
  Equipped     PC    : 0.5
  Equipped     TRUCK : 0.
  Equipped     HOV   : 0.
 -------------------------------------
 *********************
  Loading Information
 *********************

T :5. fac :  5.00 TOT VEH:   1170 GEN:   1170 OUT_N:    385 OUT_T : 0
T :10. fac :  8.00 TOT VEH:   2946 GEN:   1776 OUT_N:    765 OUT_T :331
T :15. fac : 10.00 TOT VEH:   6546 GEN:   3600 OUT_N:     20 OUT_T :1903
T :20. fac : 10.00 TOT VEH:   9426 GEN:   2880 OUT_N:      0 OUT_T :2398
T :25. fac :  8.00 TOT VEH:  11994 GEN:   2568 OUT_N:      0 OUT_T :2549
T :30. fac :  5.00 TOT VEH:  13269 GEN:   1275 OUT_N:      0 OUT_T :2508
T :35. fac :  5.00 TOT VEH:  13314 GEN:     45 OUT_N:     13 OUT_T :2043
T :40. fac :  0.00 TOT VEH:  13314 GEN:      0 OUT_N:     32 OUT_T :345
  ******* Vehicle Information *******
        Total vehicles   : 13314
        Non-tag vehicles : 1215
        Tag ( stil in )  : 0
        Tag (reached  )  : 12099
        other ( ? )      : 0

 ***************************************
 *  OVERALL STATISTICS REPORT          *
 ***************************************

 MAXIMUM SIMULATION TIME = 120. MINS
 The SIMULATION INTERVALS = 427
 SIMULATION TIME = 42.7 MINS
 START-UP TIME           = 5.
 END OF TIME OF INTEREST = 30.
 TOTAL VEHICLES          = 12099
 WITH INFO = 6091     WITHOUT INFO = 6008
 ---------------------------------------------
  information for vehicle switching & decision

 total decision number :   79964
62      112     175     249     246     328     329     330     346   315
320     278     284     259     256     213     221     189     176   159
1244
 total switch number :  21721
62      562     1239    1268    1170    934     557     223     66    5
4       1       0       0       0       0       0       0       0     0
0
 ----------------------------------------------
 TOTAL TRIP TIMES (HRS)
       OVERALL : 1017.616785978
       NOINFO  : 515.6623351968
       INFO    : 501.9544507815
 AVERAGE TRIP TIMES (MINS)
       OVERALL : 5.04645071152
       NOINFO  : 5.149757009288
       INFO    : 4.944552133786
 ----------------------------------------------
 TOTAL OVERALL TIMES (HRS)
       OVERALL : 1137.550119312
       NOINFO  : 576.7256685302
```

```
        INFO    :  560.8244507816
 AVERAGE OVERALL TRIP TIMES (MINS)
        OVERALL:  5.641210609034
        NOINFO :  5.75957724897
        INFO   :  5.524456911327
 ------------------------------------
 TOTAL ENTRY QUEUETIMES (HRS)
        OVERALL:  119.9333333334
        NOINFO :  61.06333333336
           INFO : 58.87000000003
 AVERAGE ENTRY QUEUE TIMES (MINS)
        OVERALL    0.5947598975124
        NOINFO :  0.6098202396807
        INFO   :  0.5799047775409

 -------------------------------------------------
 TOTAL STOP TIME ( MINS )
        OVERALL:  26432.9790396
        NOINFO   :  13684.31511536
        INFO     :  12748.66392443
 AVERAGE STOP TIME ( MINS )
        OVERALL:  2.184724278007
        NOINFO   :  2.277682276191
        INFO     :  2.093032987101

 -------------------------------------------------
 TOTAL TRIP DISTANCE ( MILES )
        OVERALL:  16350.12499954
        NOINFO   :  8106.749999885
        INFO     :  8243.374999882
 TOTAL VEHICLES : 12099
 AVERAGE TRIP DISTANCE ( MILES )
        OVERALL:  1.351361682745
        NOINFO   :  1.349325898782
        INFO     :  1.353369725806
```

fort.16

Output describes the behavior of ramp-metering for every controlled ramp. It provides user defined interval report and lists all the ramp and associated flow rate (veh/sec)

```
**** The output file for ramp-control ****

  -- This file provides the changed ramp control
     and changed saturation flow rate.
  iteration 20

   34   58   82  106  127  137   9   29   53   77  122
 0.56 0.56 0.56 0.54 0.49 0.44 0.45 0.50 0.53 0.55 0.56

  iteratim 40

   34   58   82  106  127  137   9   29   53   77  122
 0.52 0.45 0.36 0.23 0.24 0.32 0.28 0.29 0.34 0.45 0.51

  iteration 60

   34   58   82  106  127  137   9   29   53   77  122
 0.49 0.24 0.12 0.02 0.04 0.28 0.23 0.06 0.05 0.25 0.43

  iteration 80

   34   58   82  106  127  137   9   29   53   77  122
 0.44 0.27 0.18 0.09 0.04 0.14 0.17 0.18 0.20 0.21 0.33

  iteration 100

   34   58   82  106  127  137   9   29   53   77  122
 0.48 0.19 0.17 0.02 0.02 0.23 0.20 0.20 0.28 0.39 0.42

  iteration 120

   34   58   82  106  127  137   9   29   53   77  122
 0.47 0.27 0.07 0.02 0.02 0.24 0.18 0.19 0.15 0.28 0.42

  iteration 140

   34   58   82  106  127  137   9   29   53   77  122
 0.43 0.23 0.11 0.02 0.02 0.30 0.23 0.22 0.23 0.32 0.33

  iteration 160

   34   58   82  106  127  137   9   29   53   77  122
 0.49 0.26 0.07 0.02 0.12 0.27 0.22 0.24 0.23 0.35 0.44

  iteration 700
  34   58   82  106  127  137   9   29   53   77  122
0.52 0.37 0.33 0.33 0.38 0.56 0.45 0.36 0.35 0.34 0.39

 iteration 720

  34   58   82  106  127  137   9   29   53   77  122
0.51 0.29 0.27 0.29 0.44 0.56 0.39 0.32 0.34 0.39 0.47

 iteration 740

  34   58   82  106  127  137   9   29   53   77  122
0.52 0.38 0.39 0.42 0.55 0.56 0.42 0.29 0.26 0.37 0.49

 iteration 760

  34   58   82  106  127  137   9   29   53   77  122
0.53 0.43 0.40 0.37 0.56 0.56 0.39 0.32 0.35 0.38 0.51

 iteration 780

  34   58   82  106  127  137   9   29   53   77  122
0.52 0.37 0.40 0.43 0.55 0.56 0.43 0.31 0.27 0.35 0.45

 iteration 800

  34   58   82  106  127  137   9   29   53   77  122
0.54 0.42 0.39 0.37 0.55 0.56 0.38 0.39 0.39 0.41 0.47

 iteration 820

  34   58   82  106  127  137   9   29   53   77  122
0.52 0.38 0.40 0.42 0.55 0.56 0.43 0.40 0.37 0.42 0.49

 iteration 840

  34   58   82  106  127  137   9   29   53   77  122
0.55 0.43 0.42 0.38 0.56 0.56 0.38 0.40 0.43 0.48 0.52

 iteration 860

  34   58   82  106  127  137   9   29   53   77  122
0.54 0.43 0.47 0.53 0.55 0.56 0.42 0.39 0.36 0.43 0.49

 iteration 880

  34   58   82  106  127  137   9   29   53   77  122
0.56 0.56 0.56 0.56 0.56 0.56 0.38 0.40 0.44 0.48 0.50
```

fort. 17 -- intermediate output file

This file provides information on the simulation process in terms of vehicle movements and records vehicles traversing from link to link. Information includes vehicle's ID, current link, next link, and time of transfer. The last index shows the previous number of traversing for this trip.

```
Vehicle Now Next Time :     1  138  142  0.27    0
Vehicle Now Next Time :     2  163  159  0.27    0
Vehicle Now Next Time :    41  138  142  0.28    0
Vehicle Now Next Time :    42  138  141  0.28    0
Vehicle Now Next Time :    43  163  159  0.28    0
Vehicle Now Next Time :    44  163  159  0.28    0
Vehicle Now Next Time :   121  138  142  0.28    0
Vehicle Now Next Time :   122  163  159  0.28    0
Vehicle Now Next Time :   188  138  142  0.28    0
Vehicle Now Next Time :   189  138  142  0.28    0
Vehicle Now Next Time :   190  163  159  0.28    0
Vehicle Now Next Time :   191  163  159  0.28    0
Vehicle Now Next Time :     3         15  0.50    0
Vehicle Now Next Time :     5    3     2  0.50    0
Vehicle Now Next Time :     6    4     7  0.50    0
Vehicle Now Next Time :     8   10     9  0.50    0
Vehicle Now Next Time :     9   11    15  0.50    0
Vehicle Now Next Time :    10   12    32  0.50    0
Vehicle Now Next Time :    11   13    12  0.50    0
Vehicle Now Next Time :    12   14    16  0.50    0
Vehicle Now Next Time :    15   17    41  0.50    0
Vehicle Now Next Time :    16   18     1  0.50    0
Vehicle Now Next Time :    18   20    44  0.50    0
Vehicle Now Next Time :    19   39    16  0.50    0
Vehicle Now Next Time :    21   42    19  0.50    0
Vehicle Now Next Time :    23   44    67  0.50    0
Vehicle Now Next Time :    24   63    41  0.50    0
Vehicle Now Next Time :    26   66    44  0.50    0
Vehicle Now Next Time :    28   68    90  0.50    0
Vehicle Now Next Time :    29   87    65  0.50    0
Vehicle Now Next Time :    31   90    67  0.50    0
Vehicle Now Next Time :    33   92   114  0.50    0
Vehicle Now Next Time :    34  111    88  0.50    0
Vehicle Now Next Time :    36  132   115  0.50    0
Vehicle Now Next Time :    37  133   119  0.50    0
Vehicle Now Next Time :    38  134   137  0.50    0
Vehicle Now Next Time :    39  135   127  0.50    0
Vehicle Now Next Time :    40  136   130  0.50    0
Vehicle Now Next Time :   239  138   142  0.28    0


Vehicle Now Next Time : 12000   95    99  0.82    0
Vehicle Now Next Time :  8769   98    94  5.32    3
Vehicle Now Next Time :  6497   98    94  7.31    5
Vehicle Now Next Tim : 11363    98   94   1.74     1
Vehicle Now Next Time :  9627    98   93   4.24     3
Vehicle Now Next Time :  8963    98   96   4.43     3
Vehicle Now Next Tim :  7979    98   94   5.69     4
Vehicle Now Next Time : 12191    98   95   0.56     0
Vehicle Now Next Time : 10322    99  106   1.50     0
Vehicle Now Next Time : 10466    99  106   1.66     0
Vehicle Now Next Time : 10882    99  106   1.41     0
Vehicle Now Next Time : 10105    99  102   2.82     2
Vehicle Now Next Time : 11031    99  104   1.80     1
Vehicle Now Next Time : 10452    99  102   3.02     2
Vehicle Now Next Tim : 10389    99  104   3.13     2
Vehicle Now Next Time : 12072    99  102   0.64     0
Vehicle Now Next Time : 12193   101  143   0.51     0
Vehicle Now Next Time :  9649   103   98   3.59     2
Vehicle Now Next Tim :  9413   103   98   3.39     2
Vehicle Now Next Time :  8933   103   98   4.94     3
Vehicle Now Next Tin-e :  9351   103   98   4.30     2
Vehicle Now Next Time :  6740   103   98   6.32     4
Vehicle Now Next Time :  8253   103   98   5.46     3
```

fort. 18

Fort. 18 lists all vehicles' trajectories. Some information in the first line include :
vehicle ID #
what kind of tag : 0 : not tagged
1 : Tagged vehicle but did not reach the destination before the end of simulation interval
2 : Tagged vehicle
information : 0 : without information
1 : with information
From : starting node
To : destination node
ST : Departure time (from 0.0)
ET:Arrivaltime(mins)
NN : number of nodes in this trip
Then, nodes of the path, cummulative trip times, link trip times and delay time will be listed under the vehicle information.

```
**** Output file for vehicles tranjectories ****
========================================
This file provides all the vehicles tranjectories.

 Vehicle( # tag inf From To ST ):      10      0     37      9  0.00  4.11
8
    37    38    39    40    41    42    43     9
  0.27  0.83  1.38  1.95  2.51  3.07  4.11
  0.27  0.55  0.56  0.56  0.56  0.56  1.04
  0.00  0.00  0.16  0.00  0.00  0.16  0.00

 Vehicle( # tag inf From To ST ):       2    0    0     44     29  0.00  4.74
8
    44    43    42    41    40    39    28    29
  0.27  0.83  1.39  1.95  2.51  3.70  4.74
  0.27  0.56  0.56  0.56  0.56  1.19  1.04
  0.00  0.00  0.16  0.00  0.00  0.29  0.14

 Vehicle( # tag inf From To ST ):       4    0    0      1      2  0.10  2.59
4
     1           1     2
  0.51  1.5;  2 59
  0.51  1.02  1:06
  0.00  0.12  0.16

 Vehicle{ # tag inf From To ST ):       5    0    0      2     35  0.10  7.37
8
     2     1     7    13    19    25    31    35
  0.50  1.85  2.93  4.04  5.15  6.21  7.37
  0.50  1.34  1.08  1.12  1.11  1.06  1.16
  0.00  0.44  0.00  0.12  0.11  0.16  0.00

 Vehicle( # tag inf From To ST ):       6    0    0      2     12  0.10  4.65
6
```

```
    2     3     4     5     6     12
  0.50  1.52  2.55  3.60  4.65
  0.50  1.01  1.03  1.05  1.05
  0.00  0.11  0.13  0.15  0.00

 Vehicle( # tag inf Frcxn To ST ): 11200 2   0    19     1 14.00 5.40
6
    19    20    19    13     7     1
  0.53  1.74  2.91  4.33  5.40
  0.53  1.20  1.18  1.42  1.07
  0.00  0.10  0.18  0.32  0.00

 Vehicle( # tag inf From To ST ): 11201 2   0    19    18 14.00 8.29
9
    19    25    31    35    36    34    30    24    18
  0.55  1.62  2.75  3.87  4.96  6.06  7.20  8.29
  0.55  1.07  1.13  1.12  1.09  1.10  1.14  1.09
  0.00  0.00  0.13  0.12  0.00  0.00  0.00  0.19

 Vehicle( # tag inf from To ST ): 11202   2   0    20    33 14.00 4.30
5
    20    26    31    32    33
  0.59  1.65  3.21  4.30
  0.59  1.06  1.56  1.09
  0.00  0.00  0.56  0.00

 Vehicle( # tag inf From  To ST ) :  11203   2   0    21     3 14.00 4.07
4
    21    15     9     3
  1.45  2.85  4.07
  1.45  1.39  1.22
  0.85  0.29  0.00

 Vehicle( # tag inf From To ST ): 11204   2   0    21     7 14.00 4.15
5
    21    20    19    13     7
  0.61  1.81  2.99  4.15
  0.61  1.20  1.18  1.17
  0.00  0.10  0.18  0.00

 Vehicle( # tag inf From to ST ): 11205   2   0    21    33 14.00 3.37
4
    21    22    28    33
  0.57  2.30  3.37
  0.57  1.73  1.07
  0.00  0.13  0.17

 Vehicle( # tag inf From To ST ):  11206   2   0    21    36 14.00 4.49
5
    21    27    32    35    36
  0.79  2.28  3.31  4.49
  0.79  1.49  1.03  1.18
  0.29  0.59  0.00  0.18
 Vehicle{ # tag inf From To ST ): 11207 2 0    21    17 14.00 5.81
5
    21    40    41    16    17
  2.61  3.25  4.61  5.81
  2.61  0.63  1.36  1.20
  2.11  0.00  0.46  0.10

 Vehicle( # tag inf From To ST ): 11208 2 0    22    37 14.00 10.86
8
    22    16    42    41    40    39    38    37
  1.15  7.80  8.46  9.10  9.71  10.29  10.86
  1.15  6.64  0.66  0.64  0.62  0.58  0.57
  0.65  5.44  0.00  0.14  0.12  0.00  0.00

 Vehicle( # tag inf From To ST ): 11210 2 0    22    10 14.00 4.04
5
    22    23    22    16    10
  0.61  1.88  2.96  4.04
  0.61  1.27  1.08  1.08
  0.11  0.00  0.00  0.18

 Vehicle( # tag inf From To ST ): 11211 2 0    22    14 14.00 6.27
6
    22    28    27    21    15    14
  0.89  2.44  3.54  5.05  6.27
```

fort.20 - intermediate output file

This file shows the demand generated from each zone for different time intervals. In DYNASMART, demand generation can follow different user-specified distributions. The unit for generation is vehicles per second.

```
   iterval  :  1
 zone  number  :  1
        0.0       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3
 zone  number  :  2
        0.3       0.0       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3
 zone  number  :  3
        0.3       0.3       0.0       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3
 zone  number  :  4
        0.3       0.3       0.3       0.0       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3
 zone  number  :  5
        0.3       0.3       0.3       0.3       0.0       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3       0.3       0.3       0.3       0.3
        0.3       0.3
```

fort-22 - intermediate output file

This output file shows the detail of signal operations. The information is generated for every time step, and records the change of signal operation. If there is no change, only the time step sequence will be listed.
AC : actuated signal control
N : node number
Phase : phase sequence
start of green time
end of green time
amber time
(Amber time is fixed as part of the input data.)

```
  tin-e step : 1
N Phase GR :      7     1    0   45   50
N Phase G R :     7     2   50   95  100
AcNPhaseGR   :       8     1    0   10   15
AcNPhaseGR   :       8     2   15   25   30
N Phase GR :      9     1    0   45   50
N Phase GR :      9     2   50   95  100
NPhaseGR :       10     1    0   45   50
NPhaseGR :       10     2   50   95  100
Ac N Phase G R :     11     1    0   10   15
Ac N Phase G R :     11     2   15   25   30
Ac N Phase G R :     14     1    0   10   15
Ac N Phase G R :     14     2   15   25   30
NPhaseGR :       15     1    0   45   50
N Phase G R :    15     2   50   95  100
NPhaseGR :       16     1    0   45   50
NPhaseGR   :     16     2   50   95  100
AcNPhaseGR :         17     1    0   10   15
Ac N Phase G R :     17     2   15   25   30
Ac N Phase G R :     20     1    0   10   15
Ac N Phase G R :     20     2   15   25   30
NPhaseGR :       21     1    0   45   50
NPhaseGR   :     21     2   50   95  100
NPhaseGR :       22     1    0   45   50
NPhaseGR   :     22     2   50   95  100
AcNPhaseGR :         23     1    0   10   15
Ac N Phase G R :     23     2   15   25   30
AcNPhaseGR :         26     1    0   10   15
Ac N Phase G R :     26     2   15   25   30
NPhaseGR :       27     1    0   45   50
NPhaseGR :       27     2   50   95  100
N Phase G R:     28     1    0   45   50
NPhaseGR :       28     2   50   95  100
Ac N Phase GR :      29     1    0   10   15
Ac N Phase G R :     29     2   15   25   30
```

```
N Phase G R :      32     1    0   45   50
N Phase G R :      32     2   50   95  100
N Phase G R :      33     1    0   45   50
N Phase G R :      33     2   50   95  100
  time step :  2
  time step :  3
  time step :  4
  time step :  5
  time step :  6
AcNPhaseGR   :       8      1  30   75   80
AcNPhaseGR   :       8      2  80  125  130
AcNPhaseGR   :      11      1.  30   75   80
AcNPhaseGR   :      11      2  80  125  130
AcNPhaseGR   :      14      1  30   75   80
Ac N Phase GR :     14      2  80  125  130
AcNPhaseGR   :      17      1  30   75   80
AcNPhaseGR   :      17      2  80  125  130
AcNPhaseGR   :      20      1  30   75   80
AcNPhaseGR   :      20      2  80  125  130
AcNPhaseGR   :      23      1  30   75   80
AcNPhaseGR   :      23      2  80  125  130
AcNPhaseGR   :      26      1  30   75   80
AcNPhaseGR   :      26      2  80  125  130
AcNPhaseGR   :      29      1  30   75   80
Ac N Phase GR :     29      2  80  125  130
```

fort.27 - intermediate output file

This file shows initial routes for unequipped vehicles.
Output data includes vehicle ID number, from node, to node and travel path.

```
 vehicle from to  1,  37,  9
path  37   38   39   40   41   42   43    9    0    0

 vehicle from to  2,  44,  29
path  44   43   42   41   40   39   28   29    0    0

 vehicle from to  3,   1,   8
path   1    2    8    0    0    0    0    0    0    0

 vehicle from to  4,   1,   2
path   1    7    1    2    0    0    0    0    0    0

 vehicle from to  5,   2,  35
path   2    1    7   13   19   25   31   35    0    0

 vehicle from to  6,   2,  12
path   2    3    4    5    6   12    0    0    0    0

 vehicle from to  7,   2,   9
path   2    8    9    0    0    0    0    0    0    0

 vehicle from to  8,   4,  22
path   4    3   43   42   41   40   22    0    0    0

 vehicle from to  9,   4,  29
path   4    5   11   17   23   29    0    0    0    0

 vehicle from to 10,   4,  18
path   4   10   11   12   18    0    0    0    0    0

 vehicle from to 11,   5,  28
path   5    4   10   16   22   28    0    0    0    0

 vehicle from to 12,   5,  26
path   5    6    5    4    3   43   42   41   40   27
path  26    0    0    0    0    0    0    0    0    0

 vehicle from to 13,   5,  10
path   5   11   10    0    0    0    0    0    0    0

 vehicle from to 14,   6,   5
path   6    5    4    5    0    0    0    0    0    0
```

fort.28 - final output for tagged vehicles

Fort.28 provide detailed information for tagged vehicles at the end of simulation.
J : vehicle ID number
I : link number from where vehicle is generated
DES : destination node number
ST : departure time (minute)
TT : travel time for this trip (minutes)
DIS : trip distance (miles)

```
J,I,DES,ST,TT,DIS :   3075    1  21   5.00   6.04   2.25
J,I,DES,ST,TT,DIS :   3076    2  44   5.00  15.00   2.75
J,I,DES,ST,TT,DIS :   3077    2  25   5.00   4.05   1.75
J,I,DES,ST,TT,DIS :   3078    3  16   5.00   7.11   2.75
J,I,DES,ST,TT,DIS :   3079    4  14   5.00   4.29   1.75
J,I,DES,ST,TT,DIS :   3080    5  31   5.00   5.06   2.25
J,I,DES,ST,TT,DIS :   3081    6  36   5.00  17.03   4.75
J,I,DES,ST,TT,DIS :   3082    7   7   5.00   5.35   2.25
J,I,DES,ST,TT,DIS :   3083    8  15   5.00   1.61   0.75
J,I,DES,ST,TT,DIS :   3084   10  33   5.00   7.29   3.75
J,I,DES,ST,TT,DIS :   3085   11  17   5.00   2.80   1.25
J,I,DES,ST,TT,DIS :   3086   12  21   5.00   4.86   1.75
J,I,DES,ST,TT,DIS :   3087   13  13   5.00   7.17   2.75
J,I,DES,ST,TT,DIS :   3088   14  36   5.00   7.41   3.25
J,I,DES,ST,TT,DIS :   3089   15  30   5.00   5.19   2.25
J,I,DES,ST,TT,DIS :   3090   16  23   5.00   3.96   1.75
J,I,DES,ST,TT,DIS :   3091   16  20   5.00   8.80   3.25
J,I,DES,ST,TT,DIS :   3092   17   5   5.00   2.76   1.25
J,I,DES,ST,TT,DIS :   3093   18   1   5.00   0.52   0.25
J,I,DES,ST,TT,DIS :   3094   19  21   5.00   4.31   1.75
J,I,DES,ST,TT,DIS :   3095   20   2   5.00   3.99   1.75
J,I,DES,ST,TT,DIS :   3096   21  12   5.00   6.99   2.75
J,I,DES,ST,TT,DIS :   3097   22   5   5.00   6.70   2.75
J,I,DES,ST,TT,DIS :   3098   24  33   5.00   8.31   3.25
J,I,DES,ST,TT,DIS :   3099   25  17   5.00   5.50   2.25
J,I,DES,ST,TT,DIS :   3100   26  14   5.00   1.70   0.75
J,I,DES,ST,TT,DIS :   3101   27  14   5.00   4.39   1.75
J,I,DES,ST,TT,DIS :   3102   29  26   5.00   7.82   2.25
J,I,DES,ST,TT,DIS :   3103   30  25   5.00   7.89   4.25
J,I,DES,ST,TT,DIS :   3104   31  12   5.00   4.38   1.75
J,I,DES,ST,TT,DIS :   3105   32  14   5.00   5.44   2.25
J,I,DES,ST,TT,DIS :   3106   33   8   5.00   4.18   1.75
J,I,DES,ST,TT,DIS :   3107   35  31   5.00   8.93   4.75
J,I,DES,ST,TT,DIS :   3108   36  29   5.00   5.71   2.25
J,I,DES,ST,TT,DIS :   3109   37  19   5.00   9.52   3.75
J,I,DES,ST,TT,DIS :   3110   39  21   5.00   8.99   3.25
```

REFERENCES

Aho, A., Hopcroft, J., and Ullman, J. (1983). Data Structures and Algorithms, Addison-Wesley Publishing Co., Reading, MA.

Aho, V. A., Sethi, R., and Ullman, J. (1987). Compilers - Principles. Techniques and Tools, Addison-Wesley Publishing Co., Reading, MA.

Albrecht, R. (1968). "Determination of minimal paths in finite, directed, weighted graphs." Computing 3, pp. 184-193. (in German.)

Allen, J.R., Kennedy, K., Porterfield, G., and Warren, J. (1983) "Conversion of Control Dependence to Data Dependence," ACM Computing Machinery, 1983.

Almasi, G.S. and Gottlieb, A. (1989). Highly Parallel Computing, The Benjamin/Cummings Publishing Co., Inc., CA.

Bellman, R. (1958). "On a routing problem,". Ouart. Appl. Math. 16, pp 87-90.

Ben-Akiva, M. (1985). "Dynamic Network Equilibrium Research," Transnortation Research, Vol. 19A, No.5/6, pp. 429-43 1.

Bertsekas, D.P. and Tsitsiklis, J.N. (1989). Parallel and Distributed Computation; Numerical Methods, Prentice-Hall, Englewood Cliffs, N.J.

Bertsekas, D.P. (1987). "Dynamic Programming: Deterministic and Stochastic Models," Prentice-Hall, Englewood Cliffs, N.J.

Bertsekas, D.P. (1991). Linear Network Optimization: Algorithms and Codes, The MIT Press, Cambridge, MA.

Boyce, D.E. (1984). "Urban Transportation Network Equilibrium and Design Models: Recent Achievements and Future Prospects," Environment and Planning A 16, pp. 1445 1474.

Boyce, D.E. (1989). "Route Guidance Systems for Managing Urban Transportation Networks: Review and Prospects," presented at the Tenth Italian Regional Science Association Conference, Rome, Italy.

Brawer, S. (1989). Introduction to Parallel Programming, Academic Press, Inc.

Carey, M. (1986). "A Constraint Qualification for a Dynamic Traffic Assignment Model," Transportation Science, Vo1.20, pp. 55-58.

Carey, M. (1987). "Optimal Time Varying Flows On Congested Networks," Operations Research, Vo1.35, No. 1, pp. 58-69.

Carey, M. (1992). "Nonconvexity of the Dynamic Assignment Problem," Transportation Research, Vol. 26B, No. 2, pp. 127-133.

Carre, B.A. (1971). "An algebra for network routing problems," J. Inst. Math Appl, 7, pp. 273-294.

Carson, J.S. and Law, A.M. (1977). "A note on Spirals algorithm for the all-pairs shortest-path problem," SIAM J. Comput. 6, pp. 696-699.

Chandler, R.E., Herman, R., and Montroll, E.W. (1958). "Traffic Dynamics: Studies in Car-Following," Operation Research, Vo1.6, No.2, pp. 165- 184.

Chang, G.L., Mahmassani, H.S. and Herman, R. (1985). "A Macroparticle Traffic Simulation Model to Investigate Peak-Period Commuter Decision Dynamics," Transportation Research Record 1005, pp. 107-120.

Chang, G.L., Mahmassani, H.S., and Engquist, M. L. (1988). " System Optimal Trip Scheduling and Routing in Commuting Networks," Transportation Research Record 1251, pp. 54-65.

Clarke, S., Krikorian, R., and Rousen, J. (1963). "Computing the N best loopless paths in a network" J. Soc. Indust. Appl. Math. 11, pp. 1096-l 102.

Cooke, K.L. and Halsey, E. (1966). "The Shortest Route Through a Network with Time-dependent Internodal Transit Times", Journal of Mathematical Analysis and Application 14, pp. 493-498.

Cray Research Inc, Parallel Processing Guide (1991). CF77 Vo1.4. Cray Research, Inc, Mendota Heights MN.

Dafermos, S. (1980). "Traffic Equilibrium and Variational Inequalities", Transportation Science 14, pp. 42-54.

Dafermos, S. (1982). "The General Multimodal Network Equilibrium Problem with Elastic Demand", Networks 12, pp. 57-72.

D'Ans, G.C. and Gazis, D.C. (1976). "Optimal Control of Oversaturated Store and Forward Transportation Networks", Transportation Science 10, pp. 1-19.

Dantzig, G.B. (1957). "Discrete variable problems." Operations Research 5, pp. 266-277.

Dantzig, G.B. (1960). "On the shortest route through a network. Management Science 6, pp. 187-190.

Denardo, E.V. and Fox, B.L. (1979). "Shortest-route methods: 1. reaching, pruning, and buckets." Operations Research 27, pp. 161-186.

Denardo, E.V. and Fox, B.L. (1979). "Shortest-route methods: 2. group knap-sacks, expanded networks, and branch-and-bound." Operations Research 27, pp. 215-248.

Deo, N. and Pang, C. (1984). "Shortest Path Algorithms: Taxonomy and Annotation." Networks 14, pp. 275-323.

Dial, R.B., Glover, F., Karney, D. and Klingman, D. (1979). "A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees." Networks 9, pp. 215-248.

Dial, R.B., (1969). "Algorithm 360: shortest path forest with topological ordering." Comm. ACM 12, pp. 632 - 633.

Dijkstra, E.W. (1959). "A note on two problems in connexion with graphs." Numer. Math. 1, pp. 269-27 1.

Drew, D. R. (1968). Traffic Flow Theory and Control , McGraw-Hill Book Company, New York.

Dreyfus, S.E. (1969). "An Appraisal of Some Shortest-Path Algorithms", Operations Research 17, pp. 395-412.

Elmaghraby, S.E. (1970). "The theory of networks and management science." Part 1, Management Science 17, pp. l-34.

Engquist, M. (1982) "A Successive Shortest Path Algorithm for the Assignment Problem", INFOR 20, pp. 370-384.

Even, S. (1979). Graph Algorithms, Md. Computer Science Press.

Farbey, B.A., Land, A.H. and Murchland, J.D. (1967). "The cascade algorithm for finding all shortest distances in a directed graph", Management Science 14, pp. 19-28.

Federal Highway Administration (1980). "Traffic Network Analysis with NETSIM - A User Guide", FHWA-IP-80-3, FHWA, Washington, D. C..

Fisk, C.S. and Boyce, D.E. (1983). "Alternative Variational Inequality Formulations of the Network Equilibrium-Travel Choice Problem", Transportation Science 17, pp. 454-463.

Florian, M.S., Nguyen, S. and Pallottino, S. (1981). "A Dual Simplex Algorithm for Finding All Shortest Paths," Networks 11, pp. 367-378.

Floyd, R.W. (1962). "Algorithm 97: shortest path", Comm. ACM 5, p. 345.

Ford, L.R. and Fulderson, D.R. (1962). Flows in Networks, Princeton University, Princeton, NJ.

Fox, B.L. (1973). "Calculating k-th shortest paths," INFORM, pp. 66-70.

Fox, B.L. (1975). "More on k-th shortest paths", Comm. ACM 18, p. 279.

Fox, B.L. (1978). "Data Structures and Computer Science Techniques in Operations Research", Operations Research 26, pp. 686-717.

Frank, H. (1969). "Shortest paths in probabilistic graphs", Operations Research 17, pp. 583-599.

Fredman, M.L. (1976). "New bounds on the complexity of the shortest path problem", SIAM J. Comput. 5, pp. 83-89.

Friesz, T.L., Luque, J., Tobin, R.L. and Wie, B.W. (1989). "Dynamic Network Traffic Assignment Considered as a Continuous Time Optimal Control Problem", Operations Research Vo1.37, No.6, pp. 893-901.

Gallo, G.S. and Pallotino, S. (1986). "Shortest Path Methods: A Unified Approach," Math. Programming Study 26, pp. 38-64.

Gallo, G.S. and Pallotino, S. (1988). "Shortest Path Algorithms," Annals of Operations Research 7, pp. 3-79.

Gartner, N.H. (1982). "OPAC: A Demand-Responsive Strategy for Traffic Signal Control", Transportation Research Record 906, pp. 75-8 1.

Gartner, N.H. (1983). "Simulation Study of OPAC : A Demand-Responsive Strategy for Traffic Signal Control", Transportation and Traffic Theory, Eds., Gartner, N.H. and Wilson, N.H.M., Elsevier Science Publishing Company.

Gazis, D. C., Herman, R., and Rothery, R.W., (1961). "Non-linear Follow the Leader Models of Traffic Flow," Operation Research, Vo1.9, No.4, PP.545-567.

Gerlough, D. L. and Huber, M.J., (1971). "Traffic Flow Theory", Special Report 165, Transportation Research Board, Washington, D. C.

Ghali, M.O. and Smith, M.J. (1991). "New Dynamic Model to Evaluate the Performance of Urban Traffic Control Systems and Route Guidance Strategies," Transnortation Research Record 1306, pp 33-39.

Ghali, M.O., Smith, M.J. (1991). "A Dynamic Traffic Assignment Model", 71st TRB Annual Meeting, Washington D.C.

Ghali, M.O. and Smith, M.J. (1992). "Optimal Dynamic Traffic Assignment of a Congested City Network", Proceedings of the Second International Capri Seminar on Urban Traffic Networks, Capri, Italy, July 1992.

Gilsinn, J. and Wizgall, C. (1973). "A performance comparison of labeling algorithms for calculating shortest path trees", Technical Note 772, National Bureau of Standards, Washington, DC.

Glover, F., Glover, R. and Klingman, D. (1984). "Computational study of an improved shortest path algorithm", Networks 14, pp. 25-36.

Glover, F., Glover, R. and Klingman, D. (1986). "The Threshold Shortest Path Algorithm", Networks 16, No. 1.

Glover, F., Klingman, D., Phillips, N. and Schneider, R.F. (1985). "New Polynomial Shortest Path Algorithms and Their Computational Attributes," Management Science 31, pp.1106-1128.

Glover, F., Klingman, D. and Napier, A. (1974). "A note on finding all shortest paths", Transportation Science 8, pp. 3-12.

Glover, F., Klingman, D. and Philips, N. (1985). "A new polynomially bounded Shortest Path Algorithm", Operations Research 33, pp. 65-73.

Golden, B.L. (1976). "Shortest-path algorithms: a comparison", Operations Research 24, pp. 1164-l 168.

Golden, B.L. and Ball, M. (1978). "Shortest paths with Euclidean distances: An explanatory model", Networks 8, pp. 297-314.

Golden, B.L. and Magnanti, T.L. (1977). "Deterministic Network Optimization - A bibliography", Networks 7, pp. 149-183.

Goldman, A.J. and Nemhauser, G.L. (1967). "A transport improvement problem transformable to a best-path problem", Transportation Science 1, pp. 295-307.

Goto, S., Ohtsuki, T. and Yohimura, T. (1976). "Sparse matrix techniques for the shortest path problem", IEEE Trans. Circuits and Systems CAS-23, pp. 752-758.

Goto, S. and Sangiovanni-Vincentelli, A. (1978). "A New Shortest Path Updating Algorithm", Networks 8, pp. 341-372.

Habbal, M.B., Koutsopoulos, H.N. and Let-man, S.R. "A decomposition algorithm for the all-pairs shortest path problem on massively parallel computer architectures", To appear in Transportation Science.

Hadlock, F.O. (1977). "A shortest path algorithm for grid graphs." Networks 7, pp. 323-334.

Hakimi, S.L. (1972). "Shortest paths in graphs-a review", IEEE International Symposium pn Circuit Theory, pp. 368-369.

Hall, R.W. (1986). "The Fastest Path Through a Network with Random Time-dependent Travel Times", Transportation Science 20, pp. 182-l 88.

Halpem, J. and Priess, I. (1974). "Shortest path with time constraints on movement and parking", Networks 4, pp. 241-253.

Hart, P.E., Nilsson, N.J. and Raphael, B. (1968). "A formal basis for the heuristic determination of minimum cost paths", IEEE Trans. System Sci. and Cybemetics SSC-4, pp. 100-107.

Hendrickson, C. and Kocur, G. (1981). "Schedule Delay and Departure Time Decisions in a Deterministic Model", Transportation Science 15, pp. 62-77.

Hendrickson, C. and Plank, E. (1984). "The Flexibility of Departure Times for Work Trips", Transportation Research 18A, pp. 25-36.

Herman, R. and Ardekani, S. (1984). "Characterizing Traffic Conditions in Urban Areas", ***Transportation Science,*** Vol. 18, No.2.

Herman, R. and Rothery, R.W. (1963). " Car-Following and Steady State Flow," Theory of Traffic Flow Symposium Proceedings, pp. l- 11.

Herman, R. and Prigogine, I. (1979). "A Two-Fluid Approach to Town Traffic", Science 204, pp. 148-151.

Ho, J.K. (1980). "A Successive Linear Optimization Approach to the Dynamic Traffic Assignment Problem", Transportation Science 14, p 295-305.

Hoffman, A.J. and Markowitz, J.M. (1963). "A note on shortest path, assignment, and transportation problems", Naval Res. Logist. Quart 10, pp. 375-379.

Hoffman, A.J. and Winograd, S. (1972). "Finding all shortest distances in a directed network", IBM J. Res. Develop. 16, pp. 412-414.

Hoffman, W. and Pavley, R. (1959). "A method for the solution of the n-th best path problem", J. Assoc. Comput. Mach. 6, pp. 506-514.

Hu, T.C. (1967). "Revised matrix algorithms for shortest paths", SIAM J. Appl. Math. 15 pp. 207-218; errata 15 (1967) 1517.

Hu, T.C. (1969). Integer Programming and Network Flows, Addison-Wesley, Reading, MA.

Hu, T.Y., Rothery, R.W. and Mahmassani, H.S. (1992)."DYNASMART: Dynamic Network Assignment-Simulation Model for Advanced Road Telematics", Working Paper DTFH61-90-C-00074-TWPl, Center for Transportation, The University of Texas at Austin.

Hurdle, V.P. (198 1). "Equilibrium Flows on Urban Freeways", Transportation Science 15, p 255-293.

Janson, B.N. (1991). "Dynamic Traffic Assignment with Schedule Delay", Presented at the 7 1st TRB Annual Meeting, Washington D.C.

Jayakrishnan, R. (1992) "In-Vehicle Information Systems for Network Traffic Control: A Simulation Framework to Study Alternative Guidance Strategies", PhD Dissertation, Department of Civil Engineering, The University of Texas at Austin.

Jensen, P.A. and Barnes, J.W. (1980). Network Flow Programming, Wiley, N.Y.

Johnson, D.B. (1973). "A note on Dijkstra's shortest path algorithm", J. Assoc. Comput Mach. 20, pp. 385-388.

Johnson, D.B. (1977). "Efficient algorithms for shortest paths in sparse networks", L Assoc. Comput. Mach. 24, pp. 1-13.

Johnson, E.L. (1972). "On shortest paths and sorting", Proceedings of 25th Conference of the Association for Computing Machinery, Boston, pp. 529-539.

Joksch, H.C. (1966). "The shortest route problem with constraints", J, Math. Anal. Appl. 14, pp. 191-197.

Jonker, R. and Volgenant, A. (1987). "A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems", Computing 38, pp. 325-340.

Kalaba, R. (1960). "On some communication network problems", Combinatorial Anal, Proc. Symp. Appl. Math, 10, pp.261-280.

Katoh, N., Ibaraki, T. and Mine, H. (1978). "An O(Kn2) algorithm for obtaining the K-th shortest simple paths in an undirected graph with non-negative arc length", Electron. Commun. Japan 61-A, pp. 1192-1206.

Katoh, N., Ibaraki, T. and Mine, H. (1982). "An efficient algorithm for k shortest simple paths", Networks 12, pp. 41 l-427.

Kaufman, D.E. and Smith, R.L. (1990). "Minimum Travel Time Paths in Dynamic Networks with Application to Intelligent Vehicle/Highway Systems" Working Paper, University of Michigan.

Kaufman, D.E. and Smith, R.L. (1993). "Fastest Paths in Time-Dependent Networks for IVHS Application", IVHS Journal, Vol. 1, No. 1, pp l- 11.

Kennington, J. and Helgason, R. (1980). Algorithms for Network Programming, Wiley, N.Y.

Kershenbaum, A. (198 1). "A Note on Finding Shortest Path Trees", Networks 11, pp. 399-400.

Kirby, R.F. and Potts, R.B. (1969). "The minimum route problem for networks with turn penalties and prohibitions", Transportation Research 3, pp. 397-408.

Klafszky, E. (1972). "Determination of shortest path in a network with time-dependent edge-lengths", Math. Operations for Sch. Statist 3, pp. 255-257.

Knuth, D.E. (1977). "A generalization of Dijkstra's algorithm", Information Processing Lett, 6, pp. l-5.

Land, A.H. and Stairs, S.W. (1967). "Extension of the cascade algorithm to large graphs", Management Science 14, pp. 29-33.

Law, A.M. and Kelton, W.D. (1982). Simulation Modeling and Analysis, McGraw-Hill Book Company.

Lawler, E.L. (1972). "A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem", Management Science 18, pp. 401-405.

Lawler, E.L. (1976). "Combinatorial Optimization: Networks and Matroids", Holt, Rinehart and Winston, New York.

Lawler, E.L. (1977). "Comment on computing the k shortest path in a graph", Comm, ACM 20, pp. 603-604.

Leboeuf, J.N., Tajima, T. and Dawson, J.M. (1979). "A Magnetohydrodynamic Particle Code for Fluid Simulation of Plasmas", Journal of Comparative Physics 3 1, 3, pp 379-408.

Lee, C.E., Rioux, T.W., Savur, V.S. and Copeland, C.R. (1977). "The TEXAS Model for Intersection Traffic",' Development. Research Report No. 184-1, Center for Transportation Research, The University of Texas at Austin,TX.

Lee, C.Y. (1961). "An algorithm for path connections and its applications", IRE Trans, Electronic Comput 10, pp. 346-365.

Lee, C.Y. (1962). "A note on the N-th shortest path problem", IRE Trans, Electronic Comput 11, pp. 572-573.

Lin, H.J., Machemehl, R.B., Lee, C.E. and Herman, R. (1984). "Guidelines for Use of Left-Turn Lanes and Signal Phases", Research Report 258- 1 Center for Transportation Research, The University of Texas at Austin.

Lindley, J. (1989). "Urban Freeway Congestion Problems and Solutions: An Update", ITE Journal 59, No. 12, p 21-23.

Lisco, T.E. (1983). "Procedure for Predicting Queues and Delays on Expressways in Urban Core Areas", Transportation Research Record 944, pp. 148-154.

Mahmassani, H.S. (1990). "Dynamic Models of Commuter Behavior: Experimental Investigation and Application to the Analysis of Planned Traffic Disruptions", Transportation Research Vol. 24A, No. 6, pp. 465-484.

Mahmassani, H.S. and Chang, G.L. (1985) ," Dynamic Aspects of Departure Time Choice Behavior in a commuting System: Theoretical Framework and Experimental Analysis", Transportation Research Record, 1037, pp.88- 101.

Mahmassani, H.S. and Chang, G.L. (1986) ," Experiments with Departure Time Choice Dynamics of Urban Commuters", Transportation Research, Vol. 20B, No. 4. pp. 297-320.

Mahmassani, H.S. and Chen, P.S. (1991). "Comparative assessment of origin-based and en-route real-time information under alternative user behavior", Transportation Research Record 1306, pp. 62-81.

Mahmassani, H.S. and Herman, R. (1984). " Dynamic User Equilibrium Departure Times and Route Choice in Idealized Traffic Arterials", Transportation Science 18, pp. 362-384.

Mahmassani, H.S., Herman, R., Walton, C.M., Jones, E.G., Baaj, M.H. and Jayakrishnan, R. (1989). "Travel Time Characteristics and Opportunities for Real-Time Information Systems in an Urban Commuting Corridor", Research Report GM-1989-F, Center for Transportation Research, The University of Texas at Austin.

Mahmassani, H.S., Hu, T. and Jayakrishnan, R. (1992). "Dynamic Traffic Assignment and Simulation for Advanced Network Informatics", Proceedings of the Second International Capri Seminar on Urban Traffic Networks, Capri, Italy.

Mahmassani H.S., Hu, T.Y., Peeta, S. and Ziliaskopoulos, A. (1993), Dynamic Traffic Assignment and Simulation Procedures for ADIS/ATMS Applications: Technical Documentation", Technical Report DTFH61-90-R-00074-FT, Center for Transportation Research, The University of Texas at Austin.

Mahmassani, H.S. and Jayakrishnan, R. (1990). "Dynamic Simulation-Assignment Methodology to Evaluate In-Vehicle Information Strategies in Urban Traffic Networks", Proceeding of Winter Simulation Conference, New Orleans, LA, pp. 763-769.

Mahmassani, H.S. and Jayakrishnan, R. (1991). "System performance and motor response under real-time information in a congested traffic corridor", Transportation Research 25A, pp. 293-308.

Mahmassani, H.S. and Mouskos, K.C. (1988). "Some Numerical Results on the Diagonalization Algorithm for Network Assignment with Asymmetric Interactions Between Cars and Trucks", Transportation Research B, 22B, pp. 275-290.

Mahmassani, H.S. and Mouskos, K.C. (1989). "Vectorization of Transportation Network Equilibrium Assignment Codes", in Impacts of Recent Commuter Advances on Operations Research, North-Holland.

Mahmassani, H. S. and Peeta, S. (1992). "System Optimal Dynamic Assignment for Electronic Route Guidance in a Congested Traffic Network", Proceedings of The 2nd International Capri Seminar on Urban Traffic Networks, Capri, Italy, July 1992.

Mahmassani, H.S. and Peeta, S. (1993). "Network Performance under System Optimal and User Equilibrium Dynamic Assignments: Implications for ATIS", Transportation Research Record 1408, pp.83-93.

Mahmassani, H. S., Peeta, S., Chang, G. L. and Junchaya, T., (1992) "A review of dynamic assignment and traffic simulation models for ATIS/ATMS applications", Technical Report DTFH61-90-R-00074-1, CTR, The University of Texas at Austin.

Mahmassani, H.S., Peeta, S., Hu, T. and Rothery, R. (1993). "Effect of Real-Time Information on Network Performance under Alternative Dynamic Assignment Rules", 21st PTRC Annual Summer Conference Proceeding, Manchester, UK.

Mahmassani, H.S. and Stephan, D.G. (1988). "Experimental Investigation of Route and Departure Time Dynamics of Urban Commuters", Transportation Research Record 1203, pp. 69-84.

Mahmassani, H.S., Williams, J.C. and Herman, R. (1984). "Investigation of Network-Level Traffic Flow Relationships: Some Simulation Results, Transportation Research Record 971, pp 121-130.

Mahmassani, H.S., Williams, J.C. and Herman, R. (1987). "Performance of Urban Transportation Networks", Transportation and Traffic Theory, Editors: Gartner, N.H. and Wilson, N.H.M., Elsevier Science Publishing Co.

Marshall, J. (1973). "On Lawler's K best solutions to discrete optimization problems", Management Science 19, pp. 834-835.

Masuda, E. (1977). "A study of the computational efficiency of shortest path algorithms", Master's Thesis, University of Electra-Communications.

May, A.D. (1990). Traffic Flow Fundamentals, Prentice Hall, Englewood Cliffs, N. J.

Merchant, D.K. and Nemhauser, G.L. (1978a). "A Model and an Algorithm for the Dynamic Traffic Assignment Problems", Transportation Science, Vol. 12, No.3, pp. 183-199.

Merchant, D.K., and Nemhauser, G.L. (1978b). "Optimality Conditions for a Dynamic Traffic Assignment Model", Transportation Science, Vol. 12, no.3, pp. 200-207.

Midler, J.L. (1969). "A Stochastic Multiperiod Multimode Transportation Model", Transportation Science 3, pp. l-7.

Minieka, E. (1974). "On computing sets of shortest paths in a graph", Comm. ACM 17, pp. 351-353.

Minieka, E. (1978). Optimization Algorithms for Networks and Graphs, Dekker, New York.

Minieka, E. and Shier, D.R. (1973). "A note on an algebra for the k best routes in a network", J. Inst. Math. Appl. 11, pp. 145-149.

Minty, G.J. (1958). "A variant on the shortest-route problem", Operations Research, 6, pp. 882.

Mobility 2000 (1990). "Proceedings of the National Workshop on IVHS", Dallas.

Moore, E.F. (1959). "The shortest path through a maze", Proceedings of the International Symposium on the Theory of Switching, Part II, 1957, Harvard University, Cambridge, MA, pp. 285-292.

Moravek, J. (1970). "A note upon minimal path problem", J. Math. Anal. Appl. 30, pp. 702-7 17.

Nagumey, A. (1984). "Comparative Tests of Multimodal Traffic Equilibrium Methods", Transportation Research B, 18B, pp. 469-485.

Nagurney, A. (1986). "Computational Comparisons of Algorithms for General Asymmetric Traffic Equilibrium Problems with Fixed and Elastic Demands", Transportation Research B, 20B, pp. 78-84.

Nemhauser, G.L. (1972). "A generalized permanent label setting algorithm for the shortest path between specified nodes", J. Math. Anal. Appl. 38, pp. 328-334.

Orda, A. and Rom, R. (1990). "Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length", Journal of the ACM 37, pp. 607-625.

Orda, A. and Rom, R. (1991). "Minimum-Weight Paths in Time-Dependent Networks", Networks 21, pp. 295-319.

Pallotino, S. (1984). "Shortest Path Methods: Complexity, Interrelations and New Propositions", Networks 14, pp. 257-267.

Papadimitriou C.H. and Steiglitz, K. (1982). Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, N.J.

Papageorgiou, M., Messmer, A. and Senninger, H. (1990) "Dynamic Network Traffic Assignment Via Feedback Regulation", presented at the 70th Annual Meeting of TRB, Washington, D.C.

Pape, U. (1968). "Some computational notes on the shortest route problem", Comput. J. 11, pp. 240.

Pape, U. (1974). "Implementation and Efficiency of Moore-algorithms for the Shortest Route Problems", Math. Programming 7, pp. 212-222.

Peeta, S., Mahmassani, H.S., Rothery, R. and Herman, R. (1991). "The Effectiveness of Real-Time Information Strategies in Situations of Non-Recurrent Congestion", Proceedings of The 2nd International Conference on Applications of Advanced Technologies in Transportation Engineering, Eds. Stephanedes and Sinha, pp. 409-413.

Perko, A. (1986). "Implementation of Algorithms for K shortest loopless paths", Networks, 16 pp. 149-160.

Perko, A. (1965). "Some computatuinal notes on the shortest route problem." Comput. J. 8, pp. 19-20.

Pierce, A.R. (1975). "Bibliography on algorithms for shortest path, shortest spanning tree, and related circuit routing problems (1956-1974)", Networks 5, pp. 129-149.

Pignataro, Louis J. (1983). Traffic Engineering, Prentice-Hall, Inc., Englewood Cliffs, N.J.

Pollack, M. (1961). "The kth Best Route through a Network", Operations Research 9, pp. 578-580.

Pollack, M. (1961). "Solutions of the kth Best Route through a Network-A Review." J. Math. Anal. Appl. 3, pp. 547-599.

Pollack, M. and Wiebenson, W. (1960). "Solutions of the shortest-route problem-a review", Operations Research 8, pp. 224-230.

Pollack, M. and Wiebenson, W. (1961). "Comments on "The shortest path problem" by Peart, Randolph, and Bartlett." Operations Research 9, pp. 411-412.

Potts, R.B. and Oliver, R. (1972). Flows in Transportation Networks, Academic, New York.

Ran, B., Boyce, D.E. and LeBlanc, L.J. (1993). "A New Class of Instantaneous Dynamic User-Optimal Traffic Assignment Models", Operations Research, Vol. 41, No. 1.

Ran, B., and Shimazaki, T. (1989). "A General Model and Algorithm for the Dynamic Traffic Assignment Problems", Proceedings of the Fifth World Conference on Transport Research, Yokohoma, Japan.

Robillard, P. (1974). "Multipath Traffic Assignment with Dynamic Input Flows", Transportation Research 8, pp. 567-573.

Rosenthal, A. (1974). "On finding shortest paths in nonnegative networks", Discrete Math. 10, pp. 159-162.

Sakarovitch, M. (1968). "The k shortest routes and k shortest chains in a graph", Transportation Research 2, pp. 1- 11.

Saksena, J.P. and S. Kumar, S. (1966). "The routing problem with K specified nodes", Operations Research 14, pp. 909-913.

Shapiro, J.F. (1968) "Shortest route methods for finite state space deterministic dynamic programming problems ", SIAM J. Appl. Math. 16, pp. 1232-1250.

Sheffi, Y. (1985). Urban Transportation Networks: Equilibrium Analysis with Mathematicial Programming Methods, Prentice- Hall, NJ.

Sheffi, Y., Mahmassani, H.S. and Powell, W. (1982). "A Transportation Network Evacuation Model", Transportation Research 16A, pp. 209-2 18.

Sheffi, Y. and Powell, W. (1982). "An Algorithm for the Equilibrium Assignment Problem with Random Link Times", Networks 12, pp. 191-207.

Shier, D.R. (1974). "Computational experience with an algorithm for finding the k shortest paths in a network", J. Res. Nat. Bur. Standards, Sect. B. 78B, pp. 139-165.

Shier, D.R. (1976). "Iterative methods for determining the k shortest paths in a network", Networks 6, pp. 205-229.

Shier, D.R. (1979). "On algorithms for finding the k shortest paths in a network", Networks 9, pp. 195-214.

Shier, D.R. Witzgall, C. (1981). "Properties of Labeling Methods for Determining Shortest Path Trees," J. Res. Natl. Bureau of Standards 86, pp. 317.

Skiscim, CC. and Golden, B.L. (1987). "Computing k shortest path lengths in Euclidean networks ", Networks 17, pp. 341-352.

Smeed, R.J. (1967). "Some circumstances in which vehicles will reach their destinations earlier by starting later ", Transportation Science 1, pp. 308-317.

Smith, M.J. (1991). "A New Dynamic Traffic Model and the Existence and Calculation of Dynamic User Equilibria on Congested Capacity-Constrained Road Networks", Presented at the 71st Annual Meeting of TRB, Washington DC.

Spira, P.M. (1973). "A new algorithm for finding all shortest paths in a graph of positive arcs in average time O(n2 log2n) ", SIAM J. Comput, 2, pp. 28-32.

Spira, P.M. and Pan, A (1975) "On finding and updating spanning trees and shortest paths", SIAM J. Comput 4, pp. 375-380.

Steenbrink, PA. (1974), Optimization of Transport Networks, Wiley, Bristol .

Suurballe, J.W. (1974). "Disjoint paths in networks", Networks 4 , pp. 125-145.

Tabourier, Y. (1973). "All shortest distances in a graph. An improvement to Dantzig's inductive algorithm", Discrete Math, 4, pp. 83-87.

Tarjan, R.E. (1983). "Data Structures and Network Algorithms, SIAM regional conference series in mathematics," SIAM, Philadelphia, PA.

Tittemore, L.H., Birdsall, M.R., Hill, D.M. and Hammond, R.H. (1972). "An Analysis of Urban Area Travel by Time of Day", U.S. Department of Transportation, Washington, D.C.

Transportation Research Board (198 1). "The Application of Traffic Simulation Models", Special Report 194, Transportation Research Board, Washington, D.C.

Transportation Research Board (1985). "Highway Capacity Manual", Special Report 209, Transportation Research Board, Washington, D.C.

Transportation Research Center (1987). "The TRANSYT-7F User's Manual", University of Florida, Gainesville, Fla.

Tseng, P., Bertsekas, D.P. and Tsitsiklis, J.N. (1990). "Partially Asynchronous Parallel Algorithms for Network Flow and Other Problems," SIAM J. Control and Optimization 28 pp. 678-710.

U. S. Department of Transportation (1992). "IVHS Strategic Plan Report to Congress", Washington, D.C.

Van Aerde, M., Voss, J. , Ugge, A. and Case, E. R. (1989) "Managing Traffic Congestion in Combined Freeway and Traffic Signal Networks," ITE Journal Feb, pp. 36-42.

Van Aerde, M. and Yagar, S. (1988) "Dynamic Integrated Freeway/Traffic Signal Networks: Problems and Proposed Solutions," Transportation Reseach Vol. 22A, No.6. pp. 435-443.

Van Aerde, M. and Yagar, S. (1988) "Dynamic Integrated Freeway/Traffic Signal Networks: A Routing-Based Modelling Approach," Transportation Research Vol. 22A, No. 6.pp. 445-453.

Van Vliet, D. (1978). "Improved shortest path algorithm for transport networks." Transportation Research 12, pp. 7-20.

Wallace, C.E., White, F. J., and Wilbur, A. D. (1991)." A Permitted-Movement Model for TRANSYT-7F", Transportation Research Record 1112, pp. 45-5 1.

Weintraub, A. (1973). "The shortest and K-shortest routes as assignment problems", Networks 3, pp. 61-73.

Wie, B.W. (1990). "Dynamic Analysis of User Optimized Network Flows With Elastic Travel Demand", presented at the 70th TRB Annual Meeting, Washington, D.C.

Williams, T.A. and White, G.P (1973). "A note on Yen's algorithm for finding the length of all shortest paths in n-node nonnegative-distance network", J. Assoc. Comput. Mach. 20, pp. 389-390.

Wolsey, L.A. (1973). "Generalized dynamic programming methods in integer programming l', Math. Prog. 4, pp. 222-232.

Wongseelashote, A. (1976). "An algebra for determining all path-values in a network with application to k-shortest-paths problems ", Networks 6, pp. 307-334.

Yen, J.Y. (1969). "Some algorithms for finding the shortest routes through general networks", In Computing Methods . Optimization Problems-2, L.A. Zadeh, L.W. Neustadt, and A.V. Balakrishnan, Eds., Academic New York, pp. 377-388.

Yen, J.Y. (1970a). "A shortest path algorithm", Ph.D. Thesis, University of California, Berkeley.

Yen, J.Y. (1970b). "An algorithm for finding shortest routes from all source nodes to a given destination in general networks", Quart. Appl. Math. 27, pp. 526-530.

Yen, J.Y. (197Oc). "Finding the K shortest loopless paths in a network", Management Science17, pp. 712-716.

Yen, J.Y. (1971a). "On Elmaghraby's "The theory of networks and management science." Management Science 18, pp. 84-86.

Yen, J.Y. (1971b). "On Hu's decomposition algorithm for shortest paths in a network. Operations Research 19, pp.983-985.

Yen, J.Y. (1972a). "Finding the lengths of all shortest paths in n-node nonnegative distance complete networks using l/2 n3 additions and n^3 comparisons", J. Assoc, Comput. Mach, 19, pp. 423,424.

Yen, J.Y. (1972b). "On the efficiency of a direct search method to locate negative cycles in a network", Management Science 19, pp. 333-335.

Yen, J.Y. (1975). "Shortest Path Network Problems", Mathematical Systems in Economics, Heft 18. Hain, Meisenheim am Glan.

Yuval, G. (1976). "An algorithm for finding all shortest paths using N2.8 1 infinite-precision multiplications ", Information Processing Lett 4, pp. 155- 156.

Ziliaskopoulos, A. and Mahmassani, H.S. (1992). "Design and Implementation of a Shortest Path Algorithm with Time-Dependent Arc Costs", Proceedings of the 5th Advanced Technology Conference, Washington D.C., pp. 1072-1093.

Ziliaskopoulos, A. and Mahmassani, H.S. (1993). "A Time-Dependent Shortest Path Algorithm For Real-Time Intelligent Vehicle/Highway Systems Applications", Transportation Research Record 1408.

S.A.Zenios. (1991). "On the fine-grain decomposition of multicommodity transportation problems", To appear in SIAM joumal on