In-Vehicle Information Systems for network Traffic Control: A Simulation Framework to Study Alternative Guidance Strategies

R. Jayakrishnan, Hani S. Mahmassani

Center for Transportation Research
The University of Texas at Austin
3208 Red River, Suite 200
Austin, TX  78705-2650

Southwest Region University Transportation Center
Texas Transportation Institute
Texas A&M University System
College Station, Texas  77843-3135

Supported by a grant from the U.S. Department of Transportation, University Transportation Centers Program.

Efforts are underway around the world to use advanced telecommunication and information technologies for improving the traffic quality in congested urban areas through new approaches to affect better traffic patterns. However, these efforts to-date have proceeded without much insight on several key elements with profound effect on the resulting system performance. This research develops a simulation framework to study certain aspects that influence the performance of traffic networks under information.

A framework is developed that integrates the modelling of three key elements of traffic systems under information, namely, the traffic flow simulation, the path-processing aspects and the driver response to information. The simulation moves the vehicles using macroscopic traffic flow relations in discretized network segments, while tracking their positions. The boundedly rational behavioral model assumed for the driver response captures the driver decisions to stay on suboptimal but sufficing paths despite the provided route information. The framework is applied to candidate networks under information, to study the system performance under different levels of usage of technology and different driver behavior parameters.

Two different programs were developed: one for networks with parallel highways towards a single destination, and one for networks of general shapes and multiple destinations. The former model with faster path processing is also used for studying an idealized corridor for its stochastic-dynamic equilibration behavior under information using iterative simulations with available utility functions. The latter model is used for a realistic city network similar to the core network of Austin, Texas.

The path processing component is developed carefully, and is flexible enough to model the driver behavior of selecting from a few paths under non mandatory guidance. Efficient data structures are sued to the efficient enumeration and updating the k-shortest paths. If these paths are not updated every simulation time step, the trip times of the existing k-paths are updated by efficient routines using two possible algorithms: one intended for sequential processors and another for a processor with vectorization capabilities.

The results provide important insights on the effectiveness of in-vehicle information. Only a relatively small fraction (less than 30%) of the drivers may need to be equipped to obtain almost all of the advantages of guidance, and the system could get worse for higher percentages depending on the network context.

In-vehicle Information Systems, Traffic Control, Congestion, Traffic Networks, Simulation

No restrictions. This document is available to the public through NTIS:
National Technical Information Service
5285 Port Royal Road
Alexandria, Virginia  22312
IN-VEHICLE INFORMATION SYSTEMS FOR NETWORK TRAFFIC CONTROL: A SIMULATION FRAMEWORK TO STUDY ALTERNATIVE GUIDANCE STRATEGIES

by

R. Jayakrishnan

and

Hani S. Mahmassai

SWUTC Report Number:
SWUTC/92/71248-1

Study Title:
Information and Telecommunication Approaches to Improve Transportation System Performance

Southwest Region University Transportation Center
Center for Transportation Research
The University of Texas at Austin
3208 Red River, Suite 200
Austin, Texas 78705-2650

June 1992
IN-VEHICLE INFORMATION SYSTEMS FOR NETWORK TRAFFIC CONTROL: A SIMULATION FRAMEWORK TO STUDY ALTERNATIVE GUIDANCE STRATEGIES

ABSTRACT

Efforts are underway around the world to use advanced telecommunication and information technologies for improving the traffic quality in congested urban areas through new approaches to effect better traffic patterns. However, these efforts to-date have proceeded with out much insights on several key elements with profound effect on the resulting system performance. This research develops a simulation framework to study certain aspects that influence the performance of traffic networks under information.

A framework is developed that integrates the modelling of three key elements of traffic systems under information, namely, the traffic flow simulation, the path-processing aspects and the driver response to information. The simulation moves the vehicles using macroscopic traffic flow relations in discretized network segments, while tracking their positions. The boundedly rational behavioral model assumed for the driver response captures the driver decisions to stay on suboptimal but sufficing paths despite the provided route information. The framework is applied to candidate networks under information, to study the system performance under difference levels of usage of technology and different driver behavior parameters.
Two different programs were developed: one for networks with parallel highways towards a single destination, and one for networks of general shapes and multiple destinations. The former model with faster path processing is also used for studying an idealized corridor for its stochastic-dynamic equilibration behavior under information using iterative simulations with available utility functions. The latter model is used for a realistic city network similar to the core network of Austin, Texas.

The path processing component is developed carefully, and is flexible enough to model the driver behavior of selecting from a few paths under non mandatory guidance. Efficient data structures are used to the efficient enumeration and updating the k-shortest paths. If these paths are not updated every simulation time step, the trip times of the existing k-paths are updated by efficient routines using two possible algorithms: one intended for sequential processors and another for a processor with vectorization capabilities.

The results provide important insights on the effectiveness of in-vehicle information. Only a relatively small fraction (less than 30%) of the drivers may need to be equipped to obtain almost all of the advantages of guidance, and the system could get worse for higher percentages depending on the network context.
TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Motivation and problem definition 1
1.2 Objectives 2
1.3 Background Review 5
  1.3.1 IVHS: Developments to-date 6
  1.3.2 IVHS: Technology 8
  1.3.3 Benefits of information supply in networks 10
  1.3.4 Evaluation of networks under information 11
     1.3.4.1 Network assignment models 12
     1.3.4.2 Simulation models 14
  1.3.5 Related topics: Path-processing and user behavior 16
1.4 Research overview and layout of the thesis 17

Chapter 2

MODELLING FRAMEWORK

2.1 Modelling Approach 21
  2.1.1 Alternative methods of analysis 21
  2.1.2 Simulation-based methodology and conceptual overview 24
2.2 Traffic simulation approach 28
  2.2.1 Traffic flow model 30
  2.2.2 Incident modelling 37
2.3 Driver behavior 40
  2.3.1 Background review on driver behavior 40
  2.3.2 Behavioral models 42
  2.3.3 Simplified behavioral model 49
2.4 K-shortest paths 50
  2.4.1 Shortest path vs. k-shortest paths 51
  2.4.2 Background review 52
     2.4.2.1 Literature review 53
     2.4.2.2 K-shortest paths: Fundamental algorithms 55
  2.4.3 Methodology of binary heap data structure 59
Chapter 3

SIMULATION-ASSIGNMENT MODEL

3.1 Structure of the simulation program 67
   3.1.1 Component models 68
   3.1.2 Integration of component models 72
3.2 Network modelling 73
   3.2.1 Data structure for storage of network geometry 73
   3.2.2 Network path storage 79
3.3 Shortest paths 82
   3.3.1 Data structure for heaps 83
   3.3.2 Path enumeration and heap operations 85
3.4 Path trip time calculation and storage 91
   3.4.1 Sequential path trip time calculations 93
   3.4.2 Vectorized path trip time calculations 95
3.5 Traffic flow modelling 98
   3.5.1 Traffic loading 99
   3.5.2 Traffic flow modelling components 101
     3.5.2.1 Particle generation block 102
     3.5.2.2 Particle movement block 106
     3.5.2.3 Calculation of vehicle flux constraints 110
     3.5.2.4 Segment-to-segment movement of particles 110
     3.5.2.5 Segment condition updating 112
   3.5.3 Segment trip time and segment-queue delay calculations 113
   3.5.4 Incident modelling 115
3.6 Behavioral modelling 116

Chapter 4

SIMULATION EXPERIMENTS

4.1 Information supply strategy 120
4.2 Experimental design 121
   4.2.1 Network types 122
     4.2.1.1 Corridor network 123
     4.2.1.2 Austin core general network 124
   4.2.2 Behavioral model parameters 134
   4.2.3 Level of information 135
   4.2.4 Traffic loading patterns 135
Chapter 1

INTRODUCTION

1.1 MOTIVATION AND PROBLEM DEFINITION

While traffic conditions have been worsening at an alarming rate in many urban areas around the world, traffic engineers and policy makers are becoming increasingly aware of the shortcomings and ineffectiveness of traditional and mostly local approaches to tackle the problem. The growth of traffic demand outpaces the improvements attained even with network-wide signalization and freeway control schemes. The very high reliance on private automobiles causes the problem to be even more acute in the U.S. urban context than in many other parts of the world. Augmentation of network capacity via additional infrastructure is prohibitively expensive and is no longer considered a viable solution. With the availability of advanced and not-so-expensive communication technology, more and more efforts, around the world, are being directed towards utilizing the existing networks more efficiently by giving drivers real-time traffic information and attempting to distribute the traffic flow in better ways. Over the past couple of years, IVHS (Intelligent Vehicle-Highway Systems) has emerged as one of the most significant and challenging topics for transportation researchers, and generated considerable enthusiasm and caution among the transportation professionals around the world.

Unfortunately, the excitement for implementing the hardware and experimenting with traffic guidance does not appear to have been accompanied by concerted attempts to investigate the fundamental dynamics of traffic under
information, especially the implications of driver behavior. These dynamics are not all intuitively predictable, and it seems essential that frameworks for modelling and analyzing traffic networks under real-time information be developed, so that expensive and possibly counterproductive large-scale real-world experimentation can be prevented. It is also important to understand what might happen to the actions and satisfaction of drivers over time, as the traffic system stabilizes under information into an equilibrium, if it ever does, so that the effectiveness of information strategies can be evaluated more comprehensively.

This research effort tries to integrate the main components of a traffic network under information, namely, 1) the traffic flow, 2) the driver behavior and 3) the network information dissemination, into a single modelling framework which can be used to study the underlying dynamics and evaluate alternative designs of such traffic systems. The usefulness of such a framework for studies on the system dynamics and equilibration characteristics is also demonstrated in the research.

This next section provides an explanation of the specific objectives of the research. The chapter continues with a background review which includes discussions on the technology behind traffic networks under information, the developments to-date in related research and organizational matters, and the literature and state of the art in topics related to analyzing information-driven networks. The last section of the chapter explains the research plan and the lay out of this thesis.

1.2. OBJECTIVES

The main objective of the research is to develop a modelling framework
for analyzing realistic networks operating under information supply, which includes the development and implementation of the efficient algorithmic procedures necessary. The framework is intended to keep the behavioral and informational details to simple and tractable basics, while ensuring that they are sufficient to capture the essential aspects of the traffic system dynamics. This is necessitated by the unavailability of calibrated models of sufficient sophistication for many elements of traffic systems under information supply.

The development of the framework is carried out keeping in mind that detailed models of behavioral and informational details will be available in the future, even though there is a lack of real-world data for calibrating such models at present. Thus the intent is also to develop a modelling framework that is flexible and modular enough to be improved in the future with such models.

The next objective is to understand fundamental processes in traffic networks with information/guidance systems and derive conclusions as to how they affect the performance of such networks. This requires experimentation with representative traffic networks to evaluate their performance under different informational and behavioral scenarios. The main parameters representing the level of information in the network are the fraction of drivers who have access to information and the frequency of updating of the information on network conditions (arc and path trip times). On the driver behavioral side, the main factor is the drivers’ propensity to change their routes (or equivalently, their inertia to stay on an acceptable route) under information. Such driver response can be captured using models based on ‘bounded rationality’, reflecting the driver’s indifference to minimal advantages from route switching.

Experimentation with representative networks serve two objectives: (1)
they validate the simulation framework with regard to its consistency and correctness when used for the analysis of different network, informational and behavioral scenarios, and (2) they solidify the insights on network performance and demonstrate the robustness of the conclusions. To achieve this end, simulations are carried out on alternative scenarios, comparing the results with the performance of the same networks when there is no information supply. An implicit objective here is also to understand the 'limits' of the simulation framework in terms of the size of the network scenarios that can be analyzed.

Analyzing a specific aspect of network performance, namely its medium to longer term dynamic evolution is also an important objective. While simulating the networks for certain periods and comparing the performance with appropriate base-case scenarios are necessary, as mentioned above, that does not provide insights into the evolution of the system, when information strategies are implemented. Thus, a special-purpose iterative simulation framework is developed here to study the characteristics of the network equilibration under information, which is the evolution of the network traffic patterns to a state when the driver departure patterns are in steady-state equilibrium according based on the stochastic utility perceived by the drivers. Experimentation on the sensitivity of the stochastic-dynamic network equilibration process to the utility function assumed for the drivers is another related factor studied with this framework.

The computational efficiency of the methodology, especially under modern high-performance computing environments with vector and/or parallel processors, is a major focus in this research. This is motivated by the ambitious objectives of possible future applications of the framework (with plausible modifications) as a performance predictor in real-time route control or information systems, for which
fast and efficient computational performance is critical.

In summary, the overall objective is to make meaningful inroads into the complex questions of network performance under information with a simulation-based tool that is developed with an overall perspective on many different component phenomena that are currently not fully understood. In the future, it is expected that fine-tuning of the framework in keeping with the state-of-the-art will be possible and will be attempted.

1.3. BACKGROUND REVIEW

Most of the existing literature on traffic networks under information reports on the experience from some of the one- and two-way communication systems under experimentation around the world. The literature is rather limited with respect to the methods for analyzing the performance of such systems and the related theoretical issues. However, there is a large number of published works on topics related to the different components of the problem addressed by this study. These include: (1) network assignment, both the theoretical developments and the application of assignment for problems related to networks under information, (2) simulation frameworks, and (3) other topics such as path-processing (specifically shortest-path techniques) and driver behavior.

This section starts with a discussion of the technological and implementation developments on this topic, and continues with sections on the three related topics of significance, mentioned above. Additional reviews on specific aspects of some of these topics are provided in later chapters where such topics are considered.
1.3.1 IVHS: Developments to-date

A brief historical account of the technological developments is warranted here. Probably the first of such studies was sponsored by Federal Highway Administration in the early 70's (Electronic Route Guidance System, ERGS; Rosen, et al., 1970). In-vehicle directional guidance, based on preslected origin-destination information was attempted in that study. Unfortunately, a planned experiment in Washington D.C, and the research effort itself were abandoned by the U.S. Congress for various reasons in 1971. The next development in this area occurred when CACS (Comprehensive Automobile Traffic Control System) was undertaken in Japan in the mid 70's (Fuji, 1986). CACS proved the feasibility of ERGS technology and became the forerunner of current demonstration studies in Japan, namely RACS (Shibano et al. 1989) for expressways, and AMTICS (Okamoto, 1989) for surface streets.

The Japanese developments were followed shortly by two similar efforts in Europe: ALL-SCOUT in Germany (Von Tomkewitsch, 1987) and AUTOGUIDE in England (Jeffery et al., 1987; Belcher et al., 1987,1989). ALL-SCOUT used infrared transmitters and receivers with roadside beacons and on-board displays. AUTOGUIDE uses onboard directional arrow displays and the roadside beacons are expected to give coverage over the entire London area by the mid 1990's. The purported success of these systems has encouraged two new joint efforts, PROMETHEUS (Programme for European Traffic with Highest Efficiency and Unprecedented Safety) and DRIVE (Dedicated Road Infrastructure for Vehicle Safety in Europe), by the European community.

Such global interest in this topic is accompanied by some renewed efforts in the U.S. The PATHFINDER project initiated and field-tested on the Santa
Monica Freeway Corridor in Los Angeles, CA, by Federal Highway Administration with General Motors and California Department of Transportation is the first of what appears to be several demonstration projects in major American cities including Chicago, Detroit, Minneapolis and Houston. The Program for Advanced Technology for the Highway (PATH) which has started as a joint research venture between the California Department of Transportation and the University of California also deserves mention.

One of the most promising developments in the U.S. is the formation of IVHS-America, a nonprofit educational and scientific association that is expected to co-ordinate IVHS efforts in the public and private sector groups, especially in research and development. This organization is headquartered in Washington, D.C., and had its first meetings in August 1990. While the U.S. government funding of IVHS was only $4 million for 1990, it is $20 million for 1991. The funding is expected to grow, as $50 million is under consideration by the U.S. Congress for 1992.

IVHS encompasses several dimensions. The long-term developments in what are now known as the 'Smart Cars' and 'Smart Highways' of the future include electronic communication for automated route guidance, toll collection, hazard warnings, fleet dispatch, automated vehicle identification, automated vehicle headway control and collision avoidance etc (see Shladover, 1989, for a very ambitious list of future IVHS capabilities). According to the Mobility-2000 panel that met in March 1990 in Dallas, the IVHS program covers four broad areas: (1) Advanced Traveller Information Systems (ATIS), (2) Advanced Traffic Management Systems (ATMS), (3) Advanced Vehicle Control Systems (AVCS) and (4) Commercial Vehicle Operation Systems (CVO). Research, demonstration
or implementation projects are being initiated around the U.S. in all these areas.

1.3.2. IVHS: technology

The state of the art in technology related to information supply in traffic networks appears to be much more advanced than the know-how in utilizing it. The technological systems used can be grouped into five broad areas (Jeffery, 1988): Autonomous navigational aids, area broadcast systems, local road-side transmitter systems, mobile radio systems and local roadside transceiver systems.

The autonomous navigational aids are stand-alone systems within the vehicles and are commercially available in many forms at present. Most of these rely on finding the locations of the vehicles using what is called the 'dead-reckoning' method, which involves the continuous updating of its co-ordinates from a start-point. The ETAK system that updates vehicle positions based on the earth's magnetic field, which was used in the Pathfinder project in Los Angeles is an example of such this kind. Navigational aids using the radio location systems such as DECCA Navigator and Loran-C, or the satellite system NAVSTAR-GPS (Global Positioning System) have also been used (see French, 1987). The driver-interface includes (1) simple directional aids with arrow displays, such as DRIVEGUIDE from Nissan, NAVCOM from Toyota; (2) map display systems, where the drivers position is shown on a map (stored on a CD-ROM) with highlighted paths towards the destination, such as used in the CLASS system from Chrysler and the Travelpilot system of ETAK; or (3) route guidance aids, which show the 'best' paths to the destination, using a map or directional display (e.g., the CARIN system from Philips in the Netherlands).

The area broadcasting systems are used for communication to the vehicles
using radio, where the communicated data is used possibly for updating of the in-vehicle map storage, usually in conjunction with driver-optimized in-car route guidance systems. An example is the Radio Data System (RDS) defined by the European Broadcasting Union (1984), which superimposes digital messages on normal VHF radio broadcasts.

Local road-side transmitter systems can be based on local highway advisory radio systems (HAR) using frequencies just below and just above the standard AM broadcast, or based on road-side beacons using microwave or infrared transmission. These beacons can be used in conjunction with map display or directional aid systems for hazard warning, network updates and route guidance advice. The AUTO-SCOUT system (the predecessor of ALI-SCOUT) of Siemens, Germany, is of this kind.

Mobile radio systems for two-way communications are typically based on cellular radio systems with several duplex channels, with jurisdiction within 'cells' of a few miles radius in an urban network. The technology exists to automatically transfer communications to another unit when a vehicle crosses the cell boundaries. Incident information within a cell could be transmitted and superposed on an in-vehicle map display unit, to give an example of the advantages of such a system.

Road-side transceiver systems with two-way communication is also possible now. In fact the pioneer system ERGS in the U.S. proposed to use such technology. The road-side units could be controlled by a central computer, and can be used by the controller to receive vehicle location and destination information and to transmit guidance information to the vehicle. Infrared (as in ALI-SCOUT), microwave or radio (as in the Japanese case) are used for two-way
communication. Inductive loop detectors buried on the roadways can be used in conjunction with these road-side units to provide information on traffic conditions to the central controller.

The literature related to the hardware technology is quite extensive, and is beyond the scope of this review. Due to the importance of knowing how the on-board systems affect the drivers, from the safety and comfort standpoints, research efforts that have been initiated on the ergonomical aspects of in-vehicle navigation equipment are worthy of mention. Problems related to moving-map displays have been studied by Dingus et al. (1989). Dewar (1988) examines the informational overload that drivers could face from a psychological perspective. Jovanis et al. (1988) have looked into certain safety implications of such systems.

1.3.3. Benefits of Information Supply in Networks

The main thrust of most of the above-mentioned efforts is on the technological aspects of such systems. From a conceptual standpoint there have been attempts by a few researchers to lay out the various theoretical components of traffic under information (Boyce, 1988; Merchent, 1989; May et al., 1989). To a large extent the lack of observational work has undoubtedly been a serious limitation in these efforts.

There have not been many published results on the possible benefits that may be derived from in-vehicle information systems, in a real network. Jones et al. (1989), studied the travel time variability in a real city network and concluded that there could be advantages to individual drivers from vehicle diversion in such a network (see Mahmassani et al. 1989). The AUTOGUIDE pilot studies have also resulted in a few early results on the benefits (MVA consultants, 1989). But there
are no empirical conclusions available about the advantages that can be derived when a complete system is in place, with a large majority of the vehicles equipped to receive guidance information.

Conclusions from various research efforts to predict the benefits of Information/guidance techniques have been at best disjoint and often conflicting. Jeffery (1987) combined results from Tsuji et al (1985) and his own earlier work at Transport and Road Research Laboratory on static driver information systems (Jeffery et al, 1981) and concluded that average travel time benefits could be up to 10 percent. In contrast to this, Al-Deek et al (1989) at the University of California at Berkeley, using FREQ8PC and TRANSYT-7F simulations of the Santa Monica Freeway SMART corridor, estimated up to 33 percent savings, which seems too high and could quite probably be due to the assumption that freeway incidents do not affect the surface street traffic. Smith et al (1989) used the CONTRAM program, and estimated the possible benefits of AUTOGUIDE to be 2.5 to 6 percent for all the drivers (6-7% for guided vehicles and 3% for unguided vehicles).

1.3.4. Evaluation of networks under information

To evaluate the effectiveness of in-vehicle information systems, Tsuji et al (1983, 1985) formulated a model based on the stochastic nature of the travel times, and tested the model using the CACS pilot study data from Tokyo. This model cannot evaluate non-prescriptive guidance of vehicles, when the drivers themselves make route decisions based on the real-time information, which is expected to be the case in the United States, at least during time when such systems gain popularity. A rather simplified model of static equilibrium under
information for systems with one or two routes was developed by Arnott et al (1990). This model was intended primarily for gaining basic insights rather than to evaluate alternative information strategies, but was able to confirm that there indeed could be cases when information does not benefit the drivers. Koutsopoulos et al (1989) attempted to estimate the 'benefits' of information by examining the effect of reducing the variance of link trip times in a static stochastic user equilibrium assignment model. This approach treats information in a generic sense, regardless of the nature and type of the information, which could be a critical factor in determining its impacts. Furthermore, it is not clear how useful a static approach to this problem really is, nor how valid the assumption of equilibrium is in this context.

The next two sub-sections discuss the literature on two broad groups of approaches that are relevant to the evaluation of traffic networks under information, namely the network assignment approach and the simulation approach.

1.3.4.1. Network assignment models

One approach for understanding information-driven traffic networks that is receiving a lot of consideration is based on dynamic network assignment principles, both for system optimum and user equilibrium (though several widely differing definitions appear to have been used in this regard). As is well known, traffic assignment finds the link flows for given demands which can be used to study the network performance. Static network assignment has been studied in extensive detail over the last 25 years (see Sheffi, 1985), but dynamic assignment techniques, which attempt to find the equilibrium/optimal traffic flows in a network at as a function of the time of day, have been studied only for about the
last decade. Merchant and Nemhauser (1978) did the pioneering work in this area, and formulated a system-optimal version of the problem, which is limited to a single destination. Carey (1987) developed cases where the problem, while still for a single destination, is convex and can be solved with piecewise linear approximations. Algorithms for optimal dynamic assignment based on control theory principles have also been proposed (Wie et al, 1989; Ran and Shimazaki, 1989; Boyce, Ran and Leblanc, 1991). Another development in this area is the heuristic assignment algorithm of Janson (1990) that gives approximate Dynamic User Equilibrium conditions. Dynamic assignment under slowly varying demand and responsive signal settings has recently been studied by Smith and Ghali (1990) using the CONTRAM heuristic assignment program. They also report on certain early attempts at developing approaches for equilibrium assignment under deterministic queuing theory principles for dynamic route-assignment with fixed departure times in networks with bottlenecks.

The state of the art in the extension of dynamic assignment methods to model the performance of general networks is still in a rudimentary and rather disjoint state at present. There is no comprehensive framework yet that accomplishes dynamic assignments for reasonably sized networks with exact theories. Two computer programs developed in Europe, SATURN (Van Vliet, 1982) and CONTRAM (Leonard et al, 1989), claim to perform dynamic assignment, though both are heuristic approaches with serious limitations regarding driver behavior and the representation of system performance. CONTRAM can perform incremental assignment (a quasi-dynamic approximation) over a few time slices of 10 to 15 minutes each, and has been used to study the effectiveness of route guidance, but only with a best-path route-choice principle (Smith and
Russam, 1989). SATURN can perform assignments with better representation of intersection control than CONTRAM, and has also been used for studying networks under information (van Vuren, 1991) but it has very similar limitations as CONTRAM. Cantarella et al (1991) reports on an optimal dynamic assignment model which involves formulations that address some of the fundamental issues in dynamic assignment such as obviating the first-in-first-out requirement of average traffic flow. Though not directly applicable in studying network performance under information, the dynamic assignment and real-time routing based on optimal feedback control theory used recently for a portion of the Paris freeway network by Papageorgiou and Messer (1991) is worth mentioning here, as it appears to be potentially valuable in better guidance of traffic.

Further review of the theoretical aspects and related literature on static, dynamic and stochastic network assignment with system optimal and user-optimal perspectives is provided in chapter 2 (section 2.1.1) and chapter 5 (section 5.2) 1.3.4.2.  

**Simulation models**

The above account on the state-of-the-art reveals the need for a detailed framework, that provides realistic representations of traffic dynamics and user behavior phenomena and evaluates the effects of information. Simulation models may be the only way to incorporate many of the underlying phenomena in traffic networks under real-time information. There have been efforts at developing such models at the University of Texas as part of this study (Mahmassani et al, 1990) and at Queen's University (van Aerde et al., 1988; Blum et al., 1989).

The simulation model developed at Queen's University, called the INTEGRATION model, handles the simulation of network traffic including intersection delays in somewhat more detail than the Texas framework, for which
the component for modelling of intersections is only under development. On the other hand, the INTEGRATION framework does not have a meaningful model of the driver behavior, in that the drivers are assumed to always select the best available route. Furthermore, INTEGRATION uses best-path routing and does not involve a large-scale path processing component which is necessary in a flexible framework. The work that is reported here is on the development of a modelling system at Texas that incorporates the driver behavior in a simple and meaningful manner into a framework with large-scale and efficient path processing and sufficiently detailed traffic simulation.

Other simulation-based models that have recently been reported are the ASTERIX system (Barcelo, 1991), developed as part of the European DRIVE program, and NEMIS (Mauro, 1991). The former is in the early stages of development with an ambitious scope of integrating SATURN, CONTRAM and a few other existing programs to evaluate specific configurations of information supply, including even road-side beacon details. NEMIS is a microcomputer-based microscopic traffic simulation model with best-path routing, that seems to be limited by its computing environment. Research is being initiated under the PATH (Program for Advanced Technology for the Highway) at the University of California, Irvine to further develop simulation frameworks on the lines of the Texas framework reported in this thesis.

A dynamic simulation and assignment framework was developed by Mahmassani, Chang and Herman to investigate the day-to-day dynamics of traffic patterns in a single-destination commuting corridor with parallel routes and no cross-overs (Mahmassani et al., 1986; Chang et al., 1985.). This model was later modified for studies on the long-term effects of repair-related highway lane
closures (Mahmassani et al. 1988; Jayakrishnan, 1987). As the behavioral model in that framework was for the day-to-day decisions of the drivers, such a modelling system cannot be directly applied to the real-time problem here. It also needs modifications to simulate traffic in general networks. The simulation approach there, which keeps track of vehicle-bunches while moving them based on macroscopic traffic relations, holds promise as a viable approach for modelling realistic traffic networks within computational resource limits, and is made use of in this study, as explained in sections 2.2.1 and 3.5.2.

1.3.5. Related topics: Path-processing and user behavior

A key element in a general large network simulation, with route switching under information, is modelling the generation and updating of the shortest paths between nodes in the network performed by the traffic information center or, alternatively, the on-board computers. Depending on how mandatory the system is, the drivers could be utilizing the information to form their own choice sets of routes to select from or switch to, resulting in routes that are not always the shortest paths. If routing or assignment to the best path only is simulated, then only simple shortest path trees need to be built, as in the case of CONTRAM (Leonard et al. 1989), and the routing program used by ALI-SCOUT (Haeussermann, 1984). As said above, the modelling approach in INTEGRATION (van Aerde et al, 1988) also is based on routing to the shortest path only, disregarding the driver decisions. The current study incorporates a model with flexibility to have more than one path in the driver's route choice set. Thus k-shortest path routines are necessary. This is useful in saving computational resources too by not finding the paths every time step. This is due to the fact that
the best path(s) could still be in the set of paths if only the trip times on the k
paths are updated during the steps between two successive applications of the k-
shortest path finding algorithm. A detailed review of the literature on shortest path
algorithms, a presentation of the fundamental methodologies, and a review of the
literature on data structures for shortest paths appear in chapter 2 (Sections 2.4.2
and 2.4.3).

The literature on driver behavior, which is also of interest in this research,
is quite vast. Research has been carried out on various aspects of driver decision-
making including the departure and route-choice decisions. Most of these research
efforts were focussed on travel demand forecasting, and are only indirectly
applicable to this research. Further reviews on the topic of driver decisions can be
found in chapter 2 (section 2.3.1).

1.4. RESEARCH OVERVIEW AND LAYOUT OF THE THESIS

This research starts with the development of simulation framework. First,
a simplified simulation program is developed to model multiple parallel highways
leading to a single destination. The main reason for developing this was to gain
certain insights on network performance from simple networks of a specific kind
of geometry, which nonetheless may be quite applicable to general commuting
networks and gives definitive pointers to the capabilities that a general network
modelling framework should have. The path-processing component in this
simplified program is very fast because it relies on a simple arc numbering
scheme and does not require shortest path algorithms. In addition, the network
needs to be acyclic so that the numbering scheme and a single-loop arc traffic
flow simulation can be accomplished. The network studied consists of three nine-
mile highways to a single destination with crossover opportunities at 4 points along the way to switch routes among the three highways. Two traffic generation patterns are simulated: An arbitrary departure distribution and a user-equilibrium departure pattern. Five different fractions of drivers with information are simulated, each for five different values for the behavioral model parameter capturing average driver route switching propensity. The overall and group-wise (equipped for information and otherwise) average trip times are compared along with the route-switching statistics.

The next part of the work is the selection, adaptation and implementation of the k-shortest path algorithms and the required efficient data structures. This, in conjunction with a sequential path trip time updating code was incorporated into a general network simulation framework, that is modified from the corridor model. A two-loop simulation method (different from the corridor simulations) is used for traffic flow simulation, where the arc-to-arc vehicle demands are determined in the first loop and their movement is performed in the second. Most of the discussion on the simulation framework in this thesis is based on this program as opposed to the corridor simulation program, as it is more general and more elaborate in the details. Two different codes are developed for the component for trip time updating of the k-shortest path tree: one for a sequential computing environment and the other for a vector processing environment. These codes are developed very carefully as this component is computationally the most demanding.

The next part of the work is the preparation of the data set modified from the large core network of Austin, Texas. This is used for a set of simulation runs with the same five fractions of drivers with information and the same five values
for the driver route switching propensity parameter as in the case of the corridor simulations, with the actual traffic demand data of 1985. The simulation outputs are then first checked for validating the correctness of the simulation framework, based on paths and route-switch decisions of randomly selected vehicles, lists of successive path trees from the path-processing module and congestion patterns on selected arcs in the network. Then the average trip times and other output measures are used for developing conclusions on the network performance under the different information scenarios.

The last part of the research examines the network equilibration issues and includes the use of iterative simulations using a calibrated utility function to find a stochastic-dynamic user equilibrium, under which no driver can unilaterally change his/her route or departure time and achieve a higher utility from the trip. The simplified corridor program is used for this study. The simulations are carried out for two values of fractions of drivers with information and two values for the driver switching propensity parameter. The overall utilities are compared between a starting equilibrium state without information supply and the final state after equilibration under information. The average trip time values and the schedule delay values are also compared. The sensitivity of network performance to the utility function is also studied by trying two other values for the coefficient for earliness on arrival. Again, the results are examined for conclusions and insights.

This dissertation is divided into six chapters, including this introductory chapter. Chapter 2 describes the modelling framework, and provides the theoretical background and qualitative overviews for the different components of the framework. Chapter 3 describes the actual details of the simulation-assignment model. This includes the code details, flowcharts and underlying data structures
of the model, as well as its capabilities and output details. The various simulation experiments for alternative information scenarios including the details of the networks studied and the input data details will be given in Chapter 4. This chapter also provides the results of the various simulation runs and qualitative conclusions from the outputs. The equilibration of networks under information is described in Chapter 5 along with a brief review of the background research and the adopted methodology, followed by a discussion of the results. Chapter 6 discusses the overall conclusions from the research, identifies its significant contributions to the knowledge on urban traffic networks under information, and concludes with pointers to future research needs and possible directions.
Chapter 2
MODELLING FRAMEWORK

This chapter explains the modelling framework adopted for analyzing traffic networks under information and also points out some other approaches that could be used, alluding to the relative advantages and disadvantages of these alternative approaches. The chapter has four major sections. The first one gives overall views on the different approaches and presents a conceptual view of the simulation-based framework. The next three sections explain the major components of the methodology, namely, the traffic simulation component, the driver behavior component and the path-processing component. For each of these components, a review of the related state-of-the-art is provided along with qualitative and conceptual descriptions of the underlying principles and assumptions. The details of the computer program implementation and the integration of these components are deferred to the next chapter.

2.1 MODELLING APPROACH

2.1.1 Alternative Methods of Analysis

There have been a few attempts at modelling traffic networks under information in the recent past, though they were largely simplistic and limited in scope. Based on the approaches adopted to-date, and also the knowledge of related traffic network problems, three broad groups of approaches seem conceivable. They are, 1) analytical methods, 2) traffic assignment based methods and 3) simulation-based methods. These approaches will be examined in that order now.

The problem seems to be too complex for comprehensive analytical
treatments. The analytical approaches developed to-date are rather simplified in nature and are mostly based on network equilibrium, as mentioned in section 1.3.4 (see Amott et al, 1990 and Koutsopoulos et al, 1989). More detailed analytical models could be developed by incorporating the mathematical treatments available for some of the components of the problem. For instance, traffic can be modelled with differential equations, using some of the fluid flow analogies (see section 2.2.1). Using such traffic flow models in analytical frameworks for traffic network problems has so far been attempted only for simplified corridors or multiple bottleneck problems and never for realistic networks. See Derzko, et al., 1983, for an example of numerical simulation with traffic flow models. Mahmassani and Herman, 1984, provides an example of incorporating traffic flow models into analytical frameworks, in the context of dynamic equilibrium in a two-route network.

Driver behavior models may also be incorporated into analytical frameworks. Such models are typically statistically calibrated using data on individual level discrete choice. The existing behavioral models are mostly for driver decisions of a day-to-day nature, for route and departure time determination (see Mahmassani and Chang, 1986 and Mahmassani and Stephan, 1988 for examples). There are no existing models of driver route choice under information during a trip. Nor is any data available for the calibration of such models. As and when such individual behavior models become available based on some of the ongoing research, they could be incorporated into analytical models by assuming identical drivers within groups with different characteristics. No such models exist at present.

Stochastic models of the effectiveness of route guidance do exist (Tsuji e:
al, 1983 and 1985; see section 1.3). These are generally not detailed enough to capture traffic and behavioral dynamics, and furthermore, they do not appear flexible enough to evaluate alternative configurations and strategies of information supply.

The second possible approach is based on network assignment, which use mathematical programming techniques to find link traffic from origin-destination demand, based on system optimal or user-equilibrium principles (see section 5.2 for a more detailed discussion on this topic). The literature on assignment-based approaches to study information-driven networks was examined in section 1.3.4.1. Static assignment models are well known, but they cannot capture the varying patterns of traffic under information over time. Dynamic assignment formulations which find link flows in traffic networks during multiple time periods, have been developed, but are solvable only for small networks with simplified and usually linear link-cost functions.

One drawback of assignment models is the use of link performance functions, which do not capture the development of congestion well and are unreliable under dynamic conditions. Certain properties of network traffic such as that the traffic assigned to a route during a time period travels only till certain fractions of the route length during different time periods, cause additional difficulties in the dynamic case.

The next limitation of assignment methods is that driver behavior is not accounted for in detail. System optimal assignment assumes complete driver compliance towards the traffic control (guidance in the case of concern here), which is not a correct assumption for most real-world situations. Static user equilibrium assignment assumes certain driver behavior (Wardrop's principles, see
section 5.1 and Sheffi, 1985, p.22), which cannot be directly extended to the dynamic case. At present there is no consensus on what conditions apply in the dynamic case (see section 5.1, for different conditions proposed to-date). Even if such conditions are determined in future, it is not clear if they will be ‘rich’ enough in their behavioral content to capture the dynamics of real-time driver response towards route information.

The above noted limitations of analytical models and assignment-based models prompted the development of a simulation-based evaluation framework in this research effort. A flexible simulation framework can incorporate different components of such systems, for which individual models exist, and predict the dynamic behavior of the system.

2.1.2 Simulation-based methodology and Conceptual overview

The framework adopted for this research integrates the traffic flow models, driver behavior models and information supply strategies into a single simulation model which has as its three main components, the traffic simulation program, the behavior modelling program and the network path-processing program. See figure 2.1 for the conceptual framework, with the modular organization of these components. This section explains how the framework works as well as the basic conceptual reasons for such a structure. The detailed conceptual descriptions of these components follow in this chapter and the details of their computer implementations are addressed in the next chapter.

The essential capability of the framework is to accept Origin-Destination traffic demand (see fig 2.1) at different times and simulate the traffic in the network by moving the vehicles through the network. Vehicles enter the network
Fig. 2.1 Modelling Framework
and their movement is simulated by the traffic simulation component which updates their positions on the basis of the average speeds on the network links, which in turn depend on the vehicular density in each link. When a vehicle reaches a node of the network where it needs to make a decision on which link to move into, the decision modelling component of the framework comes into play. This module utilizes the information on the various paths leading to the vehicle's destination (the path trip times, for instance) and models the driver decision, which results in the vehicle moving into a new link. The paths used by the decision modelling component are the ones generated efficiently by the path processor component. This component utilizes the network geometry (such as node connectivity) and the link traffic cost variables (trip times, in this case) to generate the shortest path or the k-shortest paths between the nodes. As the vehicles move from link to link, the vehicle densities change in the next time step for each link, thus causing new average speeds for the next time step. The vehicle positions are updated, based on the new prevailing speeds and the discrete-time-step simulation proceeds. As each vehicle reaches its destination, it is removed from the network and the information on its individual trip is used for general conclusions about the system after statistical aggregation.

The modularity of the framework reflects the need for it to be flexible enough for later modifications. In light of the relative dearth of knowledge on many of the underlying phenomena, the driver response and the capabilities of the information supply hardware system for path processing in particular, future modifications are to be expected and provided for. Advances in driver behavior modelling may produce models that include more system attributes, some of which describing the different routes that the drivers have varying levels of knowledge
about. Similarly, the information supply strategy may include the display of multiple routes to the driver. Such possibilities favored the development of the path processing component as a separate module capable of processing a large number of routes and storing them efficiently (i.e., with much lesser storage requirement than needed for keeping separate lists of successive nodes for each route). Modularity of the framework is a necessity for future enhancement of the decision-making and path-processing components.

Another reason for the modularity is that the information that needs to be exchanged between the modules is rather clearly defined, which also helps in the independent development of the program modules. The traffic flow simulation component requires information on the link selection of the vehicles at each node where they make decisions, and needs to supply the path processor with information on link costs (trip times). All the other microscopic aspects of traffic such as queuing at the entry points and link-ends, capacity reductions due to incidents etc can all be simulated internally and independently within this module.

The decision modelling component can be reasonably assumed to work on the data on route-related variables and individual driver characteristics to model the link selection. Aspects such as whether a utility maximizing or a boundedly-rational behavior model is used is internal to this component. The path processing component needs to work on the data on network geometry and link traffic conditions. Again, aspects such as whether a label-setting or a label-correcting combinatorial algorithm is used is internal to this module. Using such clear definition of the information-exchange, the modules could be developed avoiding the loss of integrity that could result if all the complex data flow between different aspects of the system are considered together.
The decision to have the framework programmed in the FORTRAN language was made based on a few factors. First, FORTRAN is relatively more efficient than other languages such as C, Pascal etc, in number manipulations, which is the core of the traffic simulation component. In the case of path processing which involves a lot of list manipulations, C would have been at least a comparable alternative, if not better, on conventional computers. On a vector-processing environment as is provided by most of the supercomputers such as the CRAY, FORTRAN is still more efficient than C for most of the existing path processing algorithms. It should be noted, though, that on a massively parallel connection machine environment, C could perform the path operations more efficiently, but parallel algorithms for path processing problems such as shortest path generation and path-trip-time updating etc, have not been fully developed yet.

It could also be argued that FORTRAN is not the best language for certain alternative possibilities of path-processing such as those which use heuristics and other artificial intelligence concepts to process the combinatorial problems involved in path-processing. Again, the current state of the art in such methods is not advanced enough for their efficient use for the kind of framework that is attempted here.

As all the components of the framework are in FORTRAN, the communication between the modules is through the COMMON storage areas, which helps keep the program development and debugging effort minimal, while rendering future modifications easier.

2.2 TRAFFIC SIMULATION APPROACH

The traffic simulation approach is similar to, but not the same as, what
was adopted in an earlier simulation program, the MacroParticle Simulation Model (MPSM), developed at the University of Texas at Austin (see Chang, Mahmassani and Herman, 1985). As in MPSM, the vehicles in this framework are moved in bunches called macroparticles, at prevailing local speeds derived from density-speed relations within discretized segments of highways.

The approach here is different from that of MPSM on account of the fact that the vehicle are moved through a network, as opposed to a single highway in MPSM. The links in MPSM had to be numbered in increasing order in the traffic direction, so that links could be selected sequentially for moving the vehicles in them. While a link is under consideration, the vehicles reaching the end of the link could be moved then itself into the downstream link, knowing the capacity available in the downstream link. This approach is not possible in the general network, as multiple links could be feeding into a single downstream link. This means that when the links are considered in the order of their numbers, certain links get precedence in using the available capacity in the downstream link, which is incorrect. Thus, a two-step simulation of traffic is adopted here for the general network framework. The vehicle movements within the links are carried out in the first step, and the link-to-link movements of vehicles are carried out, based on the demands into each link from upstream links, during the second step.

Another improvement over MPSM that is included in this framework is the incident modelling capability. A description of the theoretical traffic flow model used, as well as a discussion of other candidate models available, is given in the next section. A description of the incident modelling methodology is provided in the following section.
2.2.1 Traffic Flow Model

Seminal works on macroscopic traffic flow theory were by Lighthill and Whitham (1955) and Richards (1956). The well known basic macroscopic equation of traffic flow is equivalent to the fluid conservation equation that characterizes compressible flow, that is,

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

where,

- $q$ = traffic flow in vehicles per time period
- $k$ = concentration in vehicles per distance
- $g$ = net vehicle generation per unit time and distance
- $x$ = location
- $t$ = time

This equation relates the rate of change of flow in space to the rate of change of density in time. It is possible to analytically solve such an equation provided the flow rate and the density can be related through other equations and the boundary conditions are specified. Equation [2.2] shows the relationship between flow, concentration and speed, which is based on a fluid continuum analogy.

$$q = k \cdot v$$

where $v$ is the speed.

If a relationship is available between the speed and the vehicle density, then equation [2.1] can be solved using equation [2.2]. The simplest form of such an equation is a linear change in speed with respect to density, known as
Greenshield's relationship, which is,

\[ v = v_j \left[ 1 - \frac{k}{k_j} \right] \]

where,

\[ v_j \] = free-flow speed

\[ k_j \] = jam vehicular density

Equation [2.1] can now be solved using equations [2.2] and [2.3], but tractable solutions are possible only for the very simple cases as in a single corridor with simplified functions for traffic-generation.

The approach adopted here is to keep track of the vehicles or macro-particles (bunches of vehicles) so that the total occupancy within each discretized segment of the network is known, which in turn is used to calculate the vehicle density in the segments. Because the vehicle positions are known, the number of vehicles that move from each segment to another during the simulation time-step is also known, thereby obviating the need to use equation [2.2] as in the case of most macroscopic simulation models. In other words, the finite difference form of the partial differential equation [2.1] is solved using the difference values that result from the movement of the particles during simulation. This finite difference form is shown in equation [2.4].

\[ k_{j}^{t+1} = k_{j}^{t} + \frac{\Delta t}{\Delta x_j}[q_j^t - q_j^t + r_j^t - O_j^t] \]

where, at time \( t \),

\[ k_j^t \] = vehicle density during the period \( [(t-1)\Delta t, t\Delta t] \)
\[ \Delta t \] = simulation time increment

\[ \Delta x_j \] = length of segment \( j \)

\[ q_{ij}' \] = total inflow traffic volume rate from upstream during \( \Delta t \)

\[ q_{dj}' \] = total outflow traffic volume rate at downstream during \( \Delta t \)

\[ i_j' \] = rate of external traffic entry during \( \Delta t \)

\[ o_j' \] = rate of traffic exiting during \( \Delta t \)

This means that during the simulation, the concentration used for moving vehicles in a segment during a time step depends on the concentration during the previous time step and the net flux in the previous time step. Thus the vehicle densities for the next time step are calculated at the end of each time step using a speed concentration relationship similar to Equation [2.3]. Using these densities, the link-speeds can be calculated for the next time step. Certain difficulties are encountered in using the form of Greenshield’s equation [2.3] in a simulation. For instance, if a network segment reaches the jam concentration during any time segment, the speeds for the next time steps will be zero, which means that no vehicles would move, and the simulation effectively "shuts down". To prevent such a situation, a modified form of the Greenshield’s equation is used for the simulations here. This is shown in equation [2.5]. This provides for a minimum speed, \( v_j' \), at jam concentration that keeps the traffic moving.

\[ v_j' = (v_{j/-v_0}) (1-k_j/k_0)^{\alpha} + v_0 \]  \hspace{1cm} 2.5

where,
\[ v_j^t = \text{mean speed prevailing in segment } j \text{ at time } t \]
\[ v_{jj} = \text{mean free-flow speed in segment } j \]
\[ v_0 = \text{minimum speed at jam density} \]
\[ k_j = \text{prevailing vehicular density} \]
\[ k_0 = \text{jam density} \]
\[ \alpha = \text{a parameter} \]

This equation specifies a non-linear relationship between the speed and the corresponding density in a segment. \( v_0 \), \( k_0 \) and \( \alpha \) are assumed parameters controlling this relationship. The exponent \( \alpha \) determines the non-linearity of the relationship. It may be noted that the maximum vehicle density allowed in a segment may be less than the jam density parameter \( k_0 \). The original Greenshield's relationship (eqn. 2.3) is a special case of this relationship when \( v_0 = 0 \) and \( \alpha = 1 \).

About the macroparticle simulation approach, one more comment is in order. This traffic simulation model originated in plasma simulation research, due to a practical difficulty with continuum flow models. During the initialization period of continuum flow plasma simulation, unrealistically large speeds were encountered due to the infinitesimally small concentrations. The macroparticle approach was proposed as a solution to this problem, as it results in finite concentrations and the consequent realistic speeds (see Leboeuf et al, 1979). It should be noted though, that due to the maximum speed in the modified Greenshield's equation, this would not occur in the traffic simulations here.
It also seems appropriate to mention some of the underlying assumptions behind the traffic flow models used. First, the ability of these models to capture the delayed dynamics of traffic are questionable. One direct result of equations 2.1, 2.2 and 2.3 is the formation of shock waves. Suppressing the source/sink terms, equation 2.1 can be written as,

$$\frac{\partial k}{\partial t} + c \frac{\partial k}{\partial x} = 0 \quad 2.6$$

where,

$$c = \frac{\partial q}{\partial k}$$

This results in a solution, \( k(x,t) = F(x-ct) \) implying that when there is a discontinuity, shock waves are formed that travel at constant speeds. If the speed of a shock-wave (a discontinuity) is \( w \) and the flow and speed of the vehicles are \( q_1 \) and \( k_1 \) in front of it, and \( q_2 \) and \( k_2 \) behind it, then the number of cars that pass the shockwave in unit time are \( q_1 - wk_1 \) and \( q_2 - wk_2 \) from the back and front respectively, which means that the speed of the shock wave will be,

$$w = \frac{q_1 - q_2}{k_1 - k_2} \quad 2.7$$

What this model fails to capture is the delay due to driver reactions, which cause the shock waves not to be so sharp in reality. The driver behavior of wanting to travel at a particular speed, and seeing what lies ahead on the road are not captured by this model, as one would expect in a model that is almost
completely based on the fluid flow abstraction. For these reasons, Payne (1971) proposed a speed-density model that borrows certain concepts from the original microscopic car-following and lane-changing principles expounded by Herman et al (1959) and consists of terms that accounts for three processes - convection which is the tendency of the traffic to continue at the current speed, relaxation which is the tendency of the traffic to adjust itself towards an equilibrium speed-flow relationship, and anticipation which is the tendency of the traffic to adjust itself to different speeds based on the speeds ahead. Even though these are concepts based on microscopic driver behavior, Payne’s speed-density relationships are at the macroscopic level, and traffic is still considered similar to a fluid, but as a ‘higher-order continuum’. This model is shown in eqn 2.8.

$$\frac{dv}{dt} = -v \frac{\partial v}{\partial x} - \frac{1}{T} (v - v_e) - \frac{v}{kT} \frac{\partial k}{\partial x}$$ 2.8

The three terms on the right hand side are for convection, relaxation and anticipation, respectively. $v_e$ is the speed at the prevailing density according to a steady state speed-density relation, $T$ is the relaxation time and $v$ is the anticipation coefficient (with proper units). A traffic simulation using such a relationship requires a finite difference form of this equation (see Payne, 1979). It may be noted that the speed in each space segment is calculated based on the speed in the upstream segment and the density in the downstream segment during the previous time steps, thus rendering it ‘more dynamic’ than the simple continuum models. At least one commercially available traffic simulation package, FREFLO (Payne, 1979), uses Payne’s model. Another higher order model was

Higher order models were not used in this research for three reasons. First, there is no knowledge yet on how such a model can be applied for a network simulation as opposed to a freeway simulation. For instance, for a discrete segment at a node, it is not clear how the anticipation term capturing the traffic conditions on the downstream segment can be specified when there are multiple downstream segments at a node, and also when the division of traffic at a node depends on driver decisions. Second, no extensive calibration of such models have yet been attempted. Third, the higher order dynamic effects are important mainly in the case of incidents, the modelling of which was not expected to be a major part of this study. A capability for capturing capacity reduction due to an incident is available in the framework, but a simple speed density relationship should be sufficient for the kind of purposes that this capability is intended for.

The first issue can be resolved, possibly by applying such models to the interior segments in a chain of highway segments between two nodes (see Papageorgiu, 1985 for an example). If that is the case and calibrated versions of such models are available in future, it is a relatively simple matter to incorporate such 'dynamic' higher order traffic flow relationships into this framework.

Other forms of simple continuum speed-density relationships such as Greenberg’s model and Edie model (see Gazis, 1971, for a few different speed-density relationships including these as well as their relative advantages and disadvantages) may also be used in this framework. Changing the speed-density relationship can be accomplished with very little programming effort.
2.2.2 Incident Modelling

One important aspect of the dynamics of any traffic system is the possibility of occurrence of incidents. In the context of urban traffic networks, incidents can occur in the form of accidents, disabled vehicles, cargo spills etc. Such incidents bring about perturbations in the system. In a broader sense, there could be perturbations on the supply-side or the demand-side. The sudden change in demand due to a ball game or a music concert would be a short term demand-side perturbation and the increase in demand due to housing developments would be a long term demand-side perturbation. These can be studied easily with the present framework by appropriately specifying the input data.

Highway construction and maintenance may cause supply-side perturbations, but usually over a few days rather than during a day, and hence is not of concern in this study. See Jayakrishnan (1987) for a study on the effect of long term lane closures etc, on an urban traffic system without information technology. The effect of such long term perturbations on a given day is of importance. Short term supply side perturbations are caused by various incidents, as mentioned above. The effect of information in traffic diversion during an incident is of keen interest, and for this purpose, the framework includes an incident simulation component and future research is planned at the University of Texas using this capability. For the sake of completeness, this component is described here.

Incidents are characterized by a reduction in the capacity of certain highway segments over specified periods of time. Incident modelling is accomplished here by reducing the total lane-miles in specified segments, which translates into reduced values of the maximum number of vehicles that those
segments can hold at jam concentrations. It is assumed that the incident data is an external input and so there is no facility for generation of random incidents. This capability of the framework is quite flexible as it accepts the incident data in terms of the exact periods or multiple periods of capacity reduction by any factor on any highway segment.

There are some questions about the accuracy of the capacity reduction approach of incident modelling, which cannot be overlooked. Incidents are usually followed by external interference (police control). If only a fraction of the highway lanes are affected by the incident, then there could be excessive lane-changing maneuvers, often under police direction. The framework does not capture such details, but does cause queue formation if the jam density is reached due to the incident.

Discrete simulation of traffic flow follows an implicit assumption that there is no abrupt discontinuities in the flow, which is not the case with capacity reductions due to incidents, and some thought has been given to handling this difficulty. In the case of the start of an incident the vehicular densities calculated may be over the maximum allowed concentration because the total lane miles in a segment is reduced while all the vehicles in a segment remain in it. For this reason, no vehicle is allowed to enter such a segment from upstream till the density drops below the maximum. This in effect is equivalent to simulating a gradual reduction in capacity. Thus the only abrupt change is in the vehicle speeds which could drop drastically. This is not entirely unrealistic, as a sudden incident often cause an immediate drop in the speeds.

On the other hand, when the period of capacity reduction due to an incident is over, the densities will fall immediately. There could be an upstream
queue and a large number of vehicles could move in during a single simulation step, causing unrealistic vehicle flux between highway segments, followed by large numbers of macroparticles moving to the same spot on the highway segment. This problem is handled by specifying flux limits between segments, which prevent unrealistic dissipation of traffic after an incident is over. The flux limits are based on an assumed service rate of vehicle dissipation into each segment.

Flux limits between segments have a profound effect on the simulation, though. If these flux limits are generally smaller than the typical flux rates that result from vehicle movement (resulting from the effective solution of the finite difference form of eqn 2.1), there could be cases where the flux limits dictate the whole simulation. On simulating a freeway stretch the effect of this would be to erroneously model a completely steady flow of traffic in some cases. This occurs as follows. Consider a segment with an upstream influx at the flux limit of \( n \) per time step. There exists a 'critical density' and speed that causes \( n \) vehicles to leave from the downstream end every time step\(^1\). Such a 'critical density' is reached when the upstream flux limits hold over a reasonable time period, after which the traffic in the modelled segment stays at a steady state. This may also cause the downstream segments to reach steady states.

As some of the earlier experience from corridor simulations showed, such erroneous modelling can indeed occur under flux limit constraints. For this reason, the flux limits are allowed to prevail only when there is a queue upstream of a

\(^1\) The critical density occurs when there are \( mn \) vehicles in the segment, and the speed is such that it takes \( m \) time steps to travel the length of the segment. For small values of \( n \) and reasonably long segments, an \( m \) value can be found.
segment. This implies that in the case of traffic without incidents, other than during heavily congested periods, the flux limits do not apply at all. Thus the congestion development and propagation can indeed be captured more realistically. Additional details of the flux limit calculations are given in section 3.3.2.

2.3. DRIVER BEHAVIOR

There have been numerous research efforts in the past in the area of driver behavior, but most of these were directed from the need to develop better models of travel demand. Such models, mostly based on disaggregate choice between trip alternatives, are usually of long-term decisions, and do not address the real-time response of the drivers towards information. In this study, the focus is on how drivers accept the information provided, and how they select their routes based on that information. The existing literature in related topics is discussed in the next section. A conceptual framework for viewing the driver decision models is presented in section 2.3.2 and the behavioral model developed for this study is discussed in section 2.3.3.

2.3.1 Background Review on Driver Behavior

Many different decisions are made by a driver about a trip. These include decisions on whether to make a trip, when to make it and by what route. The first two sets of decisions are made prior to the trip while the last one is made prior to and during a trip. The models of driver decisions would be different for different trip purposes. Kanafani (1983), Manheim (1979) and Stopher et al (1976) provide very good overviews of demand analyses based on such behavioral models. In most of such works, the route choice itself is given only very limited
treatment, and is usually taken care of by network traffic assignment techniques. For this study, the route choice, both pre-trip and enroute, are of higher significance.

Most of the past research on route choice has assumed that the drivers have complete and perfect knowledge of the alternative routes available. It has been shown by Ben-akiva et al (1984) that driver information is limited due to poor dissemination of information and lack of experience with the network, in the context of inter-urban travel. Wachs (1967) showed that the length of residence (used as a proxy for the knowledge of the network), is a significant variable in the driver behavior of avoiding congestion.

How frequently the driver makes the trip is also an important factor in the route choice process. Benshoof (1970) and Tagliacozzo et al (1973) have shown that in the case of infrequent trips, even limited knowledge is enough to make the drivers think of alternative routes. For frequent trips such as to the work place, a driver may use the same route 89 % of the time (Smith R. W., 1969). One study (Technical Univ. of Helsinki, 1969) showed that urban drivers know of three to five alternatives routes.

While the research findings are fragmented, it is clear that the dynamics of route choice is not as simple as to be analyzed based on the equilibrium principles of Wardrop (1952) that drivers equilibrate to routes such that no driver can unilaterally change routes to reduce the trip cost, as used in most conventional route-choice models.

Mahmassani and Stephan (1988) report on a study of combined departure time and route choice in a controlled experimental set-up of a simulated traffic system with two alternative routes in which actual study participants were asked
to make daily commuting trip decisions. There was strong evidence of boundedly rational behavior (Simon, 1955) which indicates that drivers look for optimal gains only outside a threshold, within which the results are satisfying and sufficing for them. That study, while addressing day-to-day decisions, allows for more dynamics in the route choice than most previous studies and could point to similar decision structures in the real-time route decisions of the drivers.

Another important factor is the simple response of the drivers towards the type and content of network information provided to them. A Los Angeles study by Shirazi et al (1988) dealt with the driver response to radio reports, electronic freeway message signs and traffic information telephone, in conjunction with the Santa Monica Freeway 'Smart Corridor' study. The survey in this study was by telephone and it is not clear how much the participants' answers reflect the actual real-time decision process. Furthermore, the response towards information through advanced hardware (electronic maps, on-board computers etc.) was not studied at all. Numerous similar studies have been done by the Texas Transportation Institute (Dudek et al., 1971, 1983), but none provided any definitive insights on real-time driver response. To the author's knowledge, there has not been any study that recorded the route-related decisions made by real drivers under real-time information, along with the pertinent factors that influenced their decisions.

2.3.2 Behavioral Models.

There are various sets of decisions that are made regarding the travel route. The first set consists of the pre-trip route choice, and the next level is the enroute route choice. It is possible that one or the other or both may be made in the presence of information. The enroute choice decisions, by definition, would
be made more than once during a trip, and would thus require a dynamic model. There are indications that pre-trip decisions with information can have significant impact on the system performance as found by Mahmassani and Chen (1991) in a study related to this research. The present research focuses only on the second set of decisions, i.e., those taken during the trip. As can be expected, the following discussion pertain to the case of non-mandatory route guidance systems, as there are no driver decisions in the case of a mandatory guidance system.

Route-related decisions during a trip occur in various ways as shown in fig 2.2. There are two broad types of decisions made. First, called SW here (as they are mostly 'switch' decisions) are the binary choice decisions whether to change the current route or not. The other decisions, called SL here (as they are mostly 'selection' decisions) are made to decide which alternate route to move to. Typically the SL decisions are conditional on SW decisions. Both SW and SL decisions have different possible variations depending on the various conditions (or circumstances), under which these decisions are taken, as shown in Fig 2.2. It should be noted that this is only a conceptualization of the decision structure, constructed so as to help in the development of possible model forms and their calibration.

The decisions are taken under a few different circumstances, as in fig 2.2. There are probabilities associated for the occurrence of these different conditions in a traffic network. With the conceptualization here, once the probabilities for the

---

2 In fact, fig 2.2 may be considered to show five dimensions of decisions: (1) to be equipped or not, (2) to pay attention to information or not, (3) to switch routes or not, (4) what combination of own knowledge and displayed information to use and (5) which route to select. Of these, the 3rd and 5th dimensions are considered important from the model-building standpoint and are discussed in this chapter.
Fig. 2.2 Conceptual grouping of decision types
occurrence of these conditions are known and calibrated versions are available for the two models, SW and SL, for different conditions, the development of overall traffic patterns can be better understood. More importantly, modelling of traffic systems with such driver behavior can be attempted. The assumption behind such conceptualization is that the probabilities of the different decision conditions (or groups of conditions, as in fig 2.2 and as discussed below) can be found from much simpler field studies than the two types of models. Alternatively, conditional multidimensional model with condition-specific variables are possible, but the structure of the conditionality could cause difficulties in their calibration. Thus the conceptualization above is an attempt at determining how many different models need to be calibrated.

In the case of the SW decisions, one possibility is that the driver is equipped for information and pays attention to it. The other is when the driver is equipped, but doesn't pay attention to information, or, when the driver is not equipped at all. The former type can be called SW-a and the latter SW-b. It does not seem necessary to look for two separate variations for the two cases in SW-b, even though, it is possible that a driver may take more risks when equipped for information, because of the secure feeling from knowing that he/she can use the information later, if in trouble. SW-a decisions are typically based on the driver's affinity towards the route that he/she is currently on, the perceived reliability of the information and the trip quality on the current route and the alternatives. The SW-b decisions are based entirely on the drivers' information and knowledge.

The SL decisions follow the SW decisions. These decisions can be expected to be discrete choice decisions of selecting from available alternative routes. Again, three different variations can be identified easily. The drivers could
rely completely on the information provided (SL-a), use the knowledge or information they possess along with the provided information (SL-b) or use only their own information (SL-c). All three variations can follow either of the SW variations (see Fig 2.2). Even the possibility that a driver decides to use the information after a decision to switch routes without using information is considered.

The above discussion suggests that there are at least 5 different groups of conditions for which models are necessary, if such models are developed separately for each group. These are SW-a&b and SL-a,b&c. A brief discussion of possible forms for these models is given below.

The SW models can be based on a threshold behavior of staying on the current route until there is enough reason to consider alternative routes. This would be similar to satisficing behavior (Simon, 1955). A boundedly rational behavior of basing the decision to switch routes on an indifference threshold of advantage required to switch, is possible (see Mahmassani et al, 1986, for an example of indifference bands in dynamic departure time decisions). Such models for an individual driver may be of the form,

\[
\text{IF } C_a - CA_t > IB_a \text{, Switch; ELSE, stay on the current route.}
\]

where,

\[
C_a = \text{cost measure on the current route at time } t.
\]

\[
CA_t = \text{scaled Aggregate Cost measure on the alternate routes.}
\]

\[
IB_a = \text{indifference Band at time } t, \text{ while on the current route.}
\]

\[
CA_t = F_1(C_1, C_2, ..., C_n), \quad n = \text{number of alternative routes in the}
\]
choice set.

\[ C_i = F_2(\hat{A}_{pi}, \tilde{A}_{ii}, \hat{A}_{ii}) \]

\[ \hat{A}_{pi}, \hat{A}_{pi} = \text{vectors of perceived attributes and coefficients of routes } i. \]

\[ \tilde{A}_{ii}, \tilde{A}_{ii} = \text{vectors of attributes supplied by the information system (trip time, in the simple case.) and their coefficients, for route } i \]

The above model involves both the attributes of alternative routes perceived by the driver as well as supplied by the information system. The specification for an indifference band function is important and it needs to be dynamic for it to reflect reality. This would mean that it should be possible for its value to change over time during the trip. One possibility is to specify this band in terms of attribute values (trip times) supplied by the information system and actually experienced values during the trip, with an initial band value at the start of the trip, based on the individual's experience in the past.

If data is available on real-life driver route switching, then with specifications for \( F_1 \) and \( F_2 \), models of above form can be estimated using a switch/no-switch binary probit model, considering their specification to result in the random disturbance term (see Chang, 1985, for a similar estimation in the case of indifference bands for day-to-day departure time decisions and the details of generating correlated error structures for successive choices). This would require detailed data, which is currently not available, though.

Multinomial choice models are applicable for the SL decisions. These models assume that the drivers are selecting from alternatives according to their
utility function which has a deterministic part and a random part, i.e.,

\[ U_{ir} = V_{ir} + \epsilon_i \]

where,
- \( U_{ir} \) = Utility of individual \( i \) for alternative \( r \)
- \( V_{ir} \) = Deterministic utility of alternative \( r \) to individual \( i \)
- \( \epsilon_i \) = random term

= Function of a set of attributes of alternative \( r \) and individual \( i \)

If the random terms are assumed to be independently and identically Gumbel distributed\(^3\), the choice probabilities will be as follows.

\[ Pr(\text{selected route } = r) = \frac{e^{-\theta x_r}}{\sum_{all \in C_i} e^{-\theta x_r}} \]

where,
- \( x_r \) = Vector of attributes of route \( r \)
- \( \theta \) = Vector of coefficients
- \( C_i \) = Choice set of alternative routes, for driver \( i \).

Such logit models can be estimated with choice data using maximum likelihood methods. If the error terms are assumed to be Normal, then we get a

\(^3\) The cumulative Gumbel form is given by \( F(\omega) = Pr(\epsilon \leq \omega) = e^{-e^{-\omega-E}} \)

where \( \omega \) is any real number and \( E \) is Euler's constant (equal to 0.5708...).
probit model, which can be estimated easily only for a few alternative routes (which may indeed be the case in real decisions).

While the above discussion tries to provide a conceptual framework to view the decision processes involved, the unavailability of real-life data prevents the calibration of any of the models for use in the simulations in this research study. So a simple model based on assumed indifference towards minimal trip time advantages is selected, and is explained in the next section. This model does not capture all the details of the decisions listed above, but corresponds well to the SW decisions which could have a higher impact on the system. The drivers are assumed to have no alternative but the best route at the SL level.

2.3.3. Simplified Behavior Model

As mentioned earlier, the experimental evidence presented by Mahmassani and Stephan (1988) suggests that commuter route choice behavior exhibits a boundedly rational character, originally proposed by Simon (1955) in business decisions. In the context of route decisions, this means that drivers look for gains only outside a threshold of benefits (also called indifference band in this research), within which the results are satisfying and sufficing for them, similar to the model suggested in the previous section for the SW decisions. A simple indifference band mechanism is used to represent the driver behavior in selecting from route alternatives. When the driver decides not to select the 'best' route displayed by the information system, it is equivalent to driver non-compliance, which is an important capability of this model. The model is as follows.

\[
\delta_f(k) = 1, \text{ if } [TTC_f(k) - TTB_f(k)] > \max(\eta_f, TTC_p, \tau_f)
\]
\[ \delta_j(k) = \begin{cases} 1, & \text{indicates a route switch; } \ 0, & \text{no switch.} \end{cases} \]

\( \text{TTC}_j(k) = \text{Trip time from node } k \text{ in a network to the destination.} \)

\( \text{TTB}_j(k) = \text{Trip time on the best alternate path.} \)

\( \eta_j = \text{Relative indifference band threshold.} \)

\( \tau_j = \text{Minimum improvement needed for a switch.} \)

This model assumes an indifference band that is dynamic, but depends only on the trip time remaining on the current route, and thus decreases during the trip. The indifference band attribute for each driver is the \( \eta_j \) fraction, which is considered distributed across the driver population according to a specified distribution. The indifference band has a minimum value \( \tau_j \) which captures the absolute minimum advantage that an alternative route should provide in trip time compared to the current route for a driver to switch routes. This minimum value is expected to prevent the cases of route switching by drivers close to the destination, to gain unrealistically small trip time benefits.

2.4 K-SHORTEST PATHS

Modelling of vehicle movement patterns under information requires that the drivers' decisions of which path to follow are simulated. At the same time, it is also important to model which paths are displayed by the information system. These tasks are accomplished by the path processor, as was shown in Fig 2.1. In
this work, the approach adopted is to find k-shortest paths, instead of just the shortest paths as in most of the other modelling systems discussed in sec. 1.3. The reasons for finding k-shortest paths instead of just the shortest paths is discussed in the next section, and the methodology used to find the k-shortest paths is presented in the following section.

2.4.1 Shortest Path vs. k-Shortest Paths

When the vehicles are assigned only to the best path, the implicit assumption is that either the information supply strategy is always to show only the best paths to the driver or the driver always follows the best route that is displayed. This would mean that only shortest paths will have to be found by the framework from the nodes to the destinations during each time step in order to simulate the movement of the vehicles that reach each node. In reality, the situation is likely to be different on two counts: (1) the drivers may not follow the best path for their own reasons, and (2) routing to the best path may not necessarily be the best strategy from the standpoint of system performance. In order to retain enough flexibility in the framework to model such scenarios, the approach adopted here is to develop a path processor that finds the k-shortest paths and stores them for use during vehicle movements (refer to fig 2.1 for the overall framework structure). These paths may be found many times during the simulation, and the frequency with which the paths are found effectively reflects the reliability of the route trip times provided by the information system.

To explain further, there are different cases that dictate the development of a module for finding the k-shortest paths. First, it is conceivable that even though the information provided to the driver is for routing to the best path, his
decision to accept the information would depend on his own knowledge of the alternative paths (learned from experience, for instance). In such a case, the paths would need to be found with the link trip times perceived by the individual. In this case, we need k-shortest paths to model the driver decisions.

Alternatively, the information system may be providing the drivers with trip-time information on alternate paths but routing information only for the best path (if best path routing happens to be the chosen strategy) in which case the drivers may be making decisions on selecting the displayed route based on actual trip times on other routes (not perceived). Interestingly, even though this is a case of best path routing, the k-shortest path trip times should still be found for modelling the information supply.

A third case would be when the driver selects from a few displayed routes, in which case we need k-shortest paths for modelling information supply as well as driver decisions. Then there is the case of ‘compulsory’ routing to a few paths (which may be adopted if the system objectives are served better by not routing to the best paths alone). Here we need k-shortest paths for modelling the information display. The possibility of such cases, and the general lack of a comprehensive driver behavior model that applies to different cases indicate that the capability for finding k-shortest paths is an essential part in a flexible framework for studying different kinds of information-driven networks.

2.4.2 Background Review

Extensive studies have been done on shortest path algorithms starting with the works of Dijkstra (1959), Bellman (1958), Ford (1956) and Moore (1957). A review of the state of the art is provided in the first part of this section, which is
followed in the second part by a more detailed discussion of the methodology behind the main groups of algorithms, label-setting and label-correcting, and their extensions to the k-shortest path problem.

2.4.2.1 Literature Review

While many variations of algorithms have been developed since Dijkstra proposed his algorithm in 1959, there are usually only few fundamental differences between the algorithms, the main differences being very often in the computational implementations, namely the data structures and the sorting techniques. Implementations of Dijkstra’s label setting algorithm, which is applicable only to networks with non-negative arc costs, and the label correcting algorithms of Bellman, Ford and Moore are available and have been studied in detail (Dial et al. 1979). Application of these algorithms with the linked-list data structure of Dial et al (1979), heap structure and a few other sorting techniques were studied by Van Vliet (1978) with specific focus on road networks (see section 2.4.3 for some more details on these data structures). Goto et al (1976, 1978) proposes algorithms based on linear algebra that essentially update the paths and save on overhead computation. As they mention, these are very useful in repeated applications of the algorithm as in the case of traffic control. However, no research seems to have been pursued for extensions and implementations of these algorithm for the k-shortest paths problems.

Algorithms have been also developed specifically for the shortest paths between all node pairs, which would be helpful in modelling in-vehicle information scenarios, where trees rooted at different destination nodes have to be found. While it is straightforward to apply the one-to-all algorithms repeatedly from different root nodes, faster algorithms that find multiple trees efficiently in
a simultaneous manner may be possible. The earliest efforts towards this were by Floyd (1962) based on transitive closures and by Dantzig (1967) based on matrix linear algebra. But as Minieka (1978) points out, these do not necessarily be more efficient than repeated applications of Dijkstra's one-to-all algorithm.

Extensions of label setting and label correcting algorithms for the k-shortest paths have also been studied (Shier, 1979). Shier's studies were all on algorithms that allow loops in the paths. In a road network, this could let a path with, for instance, a one-block loop on the best path to be in the k-path set, which is unacceptble. When a check for the loops is introduced, extensions of the above algorithms would become inefficient. Clarke et al (1963) and Yen (1971) formulated algorithms which would find k loopless paths, but only between two specific nodes and not as a tree. A study on the implementation of these algorithms appears in Perko (1986), who concluded that a hybrid of these two algorithms may generally be the fastest. But these algorithms, when extended to one-to-many cases are not as efficient as the basic label setting or label correcting algorithms with a brute-force check for repetition of nodes in a path, as the paths are inserted to the temporary lists.

One of the recent algorithms for the one-to-all problem called the partitioning shortest path algorithm (Glover et al., 1985) and a variation of it called the threshold partitioning algorithm (Glover et al., 1984) have been receiving a lot of attention as to be among the most efficient. These are label correcting algorithms with efficient partitioning of the temporary node lists (see section 2.4.2.2 for an explanation of label-correcting algorithms). It is not known if these algorithms present any advantages on extension to the n-nodes and the k-shortest paths cases.
Unfortunately, while most of the above algorithms can be applied to the current problem, no single algorithm appears to be noticeably more efficient than the others in the specific case of interest in this research, which is the k-shortest, loopless paths between all node-pairs (or at least from all nodes to a few destination nodes) in a large network with non-negative, integer costs and a low (about 3) ratio of number of arcs to number of nodes. Even more significantly, not much research has been done in the areas of parallel or vectorizable implementations of any of these algorithms for application on a supercomputer, which would have been of tremendous value in large scale traffic network simulations supercomputers. One recent development in this area is a special-purpose parallel processing environment designed for path processing in the context of networks with vehicle guidance reported by Van Grol et al (1991), who are in the process of developing parallel shortest path algorithms.

2.4.2.2. **K-shortest paths: Fundamental algorithms.**

The essentials of label setting and label correcting algorithms are discussed in this section. Label setting Algorithms apply only to networks with non-negative arc costs. A general statement of the algorithm with a minor modification for the k-shortest paths is as follows.

Given: \( C(I,J) \), the costs on all the arcs among all nodes \( I \) and all nodes \( J \) connected by an arc from \( I \)

Step 1: Set origin node number as 1, i.e., set \( I=1 \). Set \( X(i) = (c_{1i}, c_{2i}, ..., c_{ki}) \), the path cost array, with \( c_{11} = 0 \) and \( c_{ik} = \infty \) for all other \( i = 1, ..., N \) and \( k = 1, ..., K \), where \( N \) is the number of nodes and \( K \) is the number of shortest paths per node.
Step 2: Find the smallest temporary component temp, corresponding to path $m$ for node $i$. For each node $j$ adjacent to $i$, insert the value $temp + C(i,j)$ into $X(j)$ if it is smaller than at least one of the elements in $X(j)$. Store the predecessor node-path combination, of this temporary cost as $i$ and $m$.

Step 3: Make temp permanent. Remove it from $X(i)$. Reduce the size of $X(i)$ by 1. Remove $X(i)$ itself if its size is 0.

Step 4: If all the $X$ lists are removed, STOP. Else, find the node $i$ whose smallest temporary component is minimum over all the $X$ lists, and go to Step 2.

As can be seen, the extension of the general label-setting algorithm for finding $k$-shortest paths is rather simple and straightforward and involves keeping a path-cost list for each node instead of just one value. The computational performance of this algorithm in the worst case can be found out as follows: $kN$ paths are finalized (closed) in as many iterations. During each iteration, $kN$ paths to adjacent nodes are compared for insertion of one path in the 'open' list (in the worst case of complete network connectivity). Thus, if the paths are sorted and kept, the algorithm requires $O(kN^2)$ computational effort.

In a similar way, label correcting algorithms also can be modified for $k$-shortest paths. As is well known, the difference between a label correcting algorithm and a label setting algorithm is that the distance labels to each node are not final, till the algorithm terminates. The general label correcting algorithm is as follows.
Given: \( C(i,j) \), the costs on all the arcs among all nodes \( i \) and all nodes \( j \) connected by an arc from \( i \).

Step 1: Set origin node number as 1, i.e., set \( i = 1 \). Set \( X(i) = (c_{i1}, c_{i2}, \ldots, c_{iK}) \), the path cost array, with \( c_{i1} = 0 \) and \( c_{ik} = \infty \) for all other \( i = 1, \ldots, N \) and \( k = 1, \ldots, K \), where \( N \) is the number of nodes and \( K \) is the number of shortest paths per node.

Step 2: For each node \( j \) adjacent to \( i \), insert the value \( TEMP + C(i,j) \) into \( X(j) \) if it is smaller than at least one of the elements in \( X(j) \). Store the predecessor node-path combination of this temporary cost as \( f \) and \( m \).

Step 3: If \( i < N \), then \( i = i + 1 \) and go to step 2. Else, test if any of \( X \) lists have changed since the last test. If there has been a change, set \( i = 1 \) and go to step 2. Else, STOP.

It can be seen that the label correcting algorithm has \( N^2 \) iterations over the nodes, and compares a value with \( KN \) values in a list (again, for the worst case) for replacement and thus performs at \( O(kN^3) \) effort compared to \( O(kN)^2 \) for the label setting algorithm. Of course, both these are efforts when the most straightforward computer implementations are used for the lists. One advantage that label correcting algorithms present is that the network costs can be negative. It has also been found that the performance of these algorithms are quite comparable in a lot of real-world applications (See Van Vliet, 1978) when the network is sparse (low arcs-to-node ratio) which is the case with urban street
networks and also the problem that is of concern in this research.

A third kind of Algorithm is Floyd's algorithm (Floyd, 1962) for shortest paths from all-nodes-to-all-nodes, which does not explore nodes one by one, but performs matrix operations with iterations to find shorter paths from any node i to any node j through all nodes k, thus resulting in $O(kN^2)$ computational effort, which is the case always, not just in the worst case. This algorithm allows negative costs on arcs, provided there are no negative cycles, as in the case of label correcting algorithms.

The computational performance of both the label setting and label correcting algorithms shown above can be improved tremendously, using efficient computer implementation of the lists and better algorithms for node selections etc. See Glover et al (1979, 1984), Shier (1979) and Ahuja et al (1990) for examples of efficient computer implementations.

Certain difficulties arise with respect to loops in the paths formed by the above given k-path extensions of shortest path algorithms. Loops are not a concern in the case of the shortest paths because an algorithm cannot find a path with a loop as the shortest, as removing the loop would always give a shorter path. This is not the case for k-shortest paths. Here, from the second to the k-th shortest paths, there can be loops which will be over a shorter path. Such paths cannot be in the choice set of the drivers, nor can they be displayed by an information system. Thus, it is necessary to consider the modifications needed to find loopless paths. As the labels are not finalized till the algorithm is terminated in the case of matrix algorithms such as Floyd's, there is no easy way of removing paths with loops and replacing them with other paths in the path lists. On the other hand, with label correcting and label setting algorithms there is the option of checking
the nodes up the predecessor-node chain for the same node which would confirm
the existence of a loop, when a new node is explored from another node. In the
worst case, this would make the label setting algorithm perform in \(O(k^2N^2)\), and
the label correcting algorithm in \(O(kN^4)\). In fact, if only \(n\) nodes are checked
back (\(n < 10\) may be sufficient for road networks), the label setting computational
effort is \(O(nk^2N^2)\), which is acceptable. This is one of the main reasons why
matrix algorithms were not selected in this study, in spite of the possibilities of
better vector processing with them.

The label setting and label correcting algorithms can be expected to have
comparable performances, even though label setting algorithms with efficient data
structures have been considered to perform better than the label correcting
algorithms (Ahuja, 1990), the latter being of advantage mostly in the case of arc-
costs which can be negative (for which the former cannot be used). As the trip
times on traffic network links are always positive, the label setting algorithm was
selected. Neither algorithm provides any significant advantages over the other
from the point of view of vectorizability either. Next, the data structures used is
discussed.

2.4.3 Methodology of Binary Heap Data Structure

One key aspect of the label setting shortest path algorithm is the search
over the current list of paths for insertion of new paths when new nodes are
explored. This means that the computational effort depends on how the path-list
is stored and how the list is modified. The computational efforts that were
discussed in the last section assumed that as this would be an \(O(N)\) operation, as
the list would remain sorted by path-costs as it is built over the iterations. A few different data structures have been used for improving the efficiency of this operation. A review of implementations based on different data structures is given next, and a discussion of the binary heap data structure used here follows.

Among the first efficient algorithms developed was Dial’s (1969), called address calculation sort. Dial used the property that the maximum reduction in a value in the path trip time list that is possible during an iteration is the maximum cost \( C_m \) of any arc in the network. Thus, if the arc costs are integers (or can be approximated by integers), then each path’s total cost can be uniquely associated with a number between 1 and \((C_m + 1)\), and all the path-costs with the same number are stored together in a thread, as a one-way linked-list. This would mean that much fewer numbers are compared during the list modification. This algorithm had better performance than most other label setting algorithms for integer networks (See Gilsin and Witzgall, 1973) with reasonable \( C_m \) values.

It can be seen that Dial’s algorithm is an extreme case of a more general sorting procedure, called bucket sorting (Denardo and Fox, 1979, Fox, 1978) where the values are linked to groups (buckets) of different sizes, where size = the difference between the maximum and minimum values linked to it + 1. The most efficient algorithm of this kind seems to be the one recently developed by Ahuja et al (1990) at MIT. They consider segmenting each bucket too (called two-level radix sort). These algorithms perform worse when the integer cost range is high, which is the case with the trip time costs on urban traffic networks (if real values in seconds are converted to integers), and for these reasons, other algorithms needed to be considered.
HEAP FORM

HEAP IN AN ARRAY FORM

( 2 3 2 3 6 8 2 7 4 \(\square\) 12 9 8 3 4 )

* The ordering is only between certain positions
* Position 2p and position 2p+1 are not larger in value to position p
* Position 1 always has the minimum value
* \(\rightarrow\) shows the relation ' < '

Fig. 2.3 Binary heap of a list of numbers
Thus, only $\log N$ comparisons are needed to delete from or add to the heap, as opposed to $N$ in the case of a non-heap array.

Fig.2.4 Deleting the minimum and reforming to a new heap
swapped with the parent or child as the case is and the checks are continued. This means that the number is checked with only $\log_d(N)$ numbers in the worst case.

In a simple sorted list, removal of the minimum can be done with no computational effort. But, once the smallest number is removed from a binary heap, it can be reformed by moving the smallest child (from position 2 or 3 for a binary heap) up, and then one of its children up and so on, a comparison of 2 numbers carried out every time, and the worst case computational effort will be $O(\log_2 N)$. In a d-heap, this may seem smaller at $O(\log_d N)$, but the minimum of the d children has to be found at every level and thus the actual computational effort need not be better than in a binary heap even though fewer levels of comparisons only are required. An example of reforming of the heap after the minimum is removed is shown in Fig 2.4.

It should be remembered that the list of path trip times in a label setting algorithm builds up from zero size, grows in size and finally reduces to zero size. This means that there is no sorting done on the list in both a regular array data structure and a heap data structure. Thus the computational effort is spent completely on deleting the minimum and reforming. As stated above, for a heap the delete-minimum, insertion and reform operations, each requires $O(\log N)$ effort and thus the overall effort is also $O(\log N)$. Compared to this, a regular list will require linear time for delete-minimum operation and reforming, while it takes $O(N)$ effort for insertion.

In the case of k-shortest paths, kN numbers are in the list at worst, and so the label setting algorithm performs in $O(kN\log_kN)$ in the worst case with
binary heaps, while its performance would be $O(kn^2)$ with a regular list. Of course, for networks with low connectivity, as in the case of traffic networks, these bounds are lower. For such reasons, a heap data structure is implemented for the shortest path finding module in the framework used in this research. The implementation details are discussed in the next chapter.

The discussion above was for the case of the $k$-shortest paths found from one node to all the nodes. In this research study, the shortest paths are needed towards the destinations of the drivers from the various nodes in the network. As there are typically much fewer destinations in a traffic network than there are nodes, the strategy adopted here is to repeat the shortest paths from the relatively few traveller-destination nodes (which do become the root node in the shortest path terminology) to all the nodes in the network.

It may be worthwhile to mention here that shortest path finding is only one part of the path-processing required in the modelling work here. As it may not be possible (nor necessary) to repeat the shortest paths during every simulation step, the methodology used here is to find the path trip times on the $k$-shortest paths during every time step, updating the composition of the paths themselves only every given number of time steps. This can be accomplished in much less computational effort than finding the paths. Thus the path trip-time aggregation component is the second part of the path processing module and will be discussed in detail in the next chapter.

Comments are also warranted on the ability for algorithms based on heap data structure to be vectorized. As the heap operations are inherently sequential in nature, these shortest path algorithms may be expected to be poor in vectorizability. Finding vectorizable/parallelizable algorithms for this problem is
beyond the scope of this study and is not attempted. To the author's knowledge, such algorithms are only in the early stages of development.
Chapter 3

SIMULATION-ASSIGNMENT MODEL

This chapter describes the simulation-assignment framework in detail, including the program details of all the component modules. The implementation of the code is discussed with the kind of detail that is unusual in similar research theses. This is because the efficient implementation is important to the model's ability to handle large networks with a general structure, which is a contribution of this research effort. Furthermore, such an in-depth discussion is expected to be very useful from the point of view of the continuing research efforts in this area at the University of Texas, for which this framework is expected to play a significant part.

This chapter includes discussions of the data structures used and the execution flow of the program. Section 3.1 describes the overall structure of the program and explains how the methodological linkages examined in chapter 2 are achieved in the program. Section 3.2 describes how the traffic network details are handled. Sections 3.3 and 3.4 describes the path processing component, section 3.5 describes the traffic simulation component, and section 3.6 describes the driver behavior modelling component.

3.1 STRUCTURE OF THE SIMULATION PROGRAM

The general network simulation program has three main conceptual components, as explained before, which are the traffic simulation component, the path processing component and the driver behavior modelling component. The next section explains how the three conceptual components are incorporated into
the program and the following section explains the real linkages that integrate
them together in the program.

3.1.1 Component Modules

The simulation program consists of various routines which can be
conceptually grouped together into three main modules for traffic simulation, path
processing and behavior modelling and a fourth module that handles the external
interface as well as the control of the simulation clock. The conceptual flow of the
program execution is shown in fig 3.1.

As can be seen, the MAIN program accepts the data for simulation. This
includes the network geometry, zonal traffic demand during different demand
intervals, incident data and the parameter values defining the market penetration
of the information, driver behavior etc. After the initialization of the arrays and
the necessary calculations to transform the input data including nodal traffic
generations, the simulation clock is set to zero (or the specified start-time). The
network is assumed to have free flow conditions at the start with no vehicles. The
arcs in the network may be discretized into several segments (thus making the arcs
'segment chains') as explained in section 3.2.1. A subroutine named ADDCHAIN
is called to add trip times on segment chains and return the arc trip times. Then
the initial k-shortest paths are enumerated using the subroutine called KSHORT.
Once this is completed, the simulation for the first time step is started with a call
to the PARTCO subroutine that carries out the traffic simulation. Once the
vehicles are moved during the time step, the time is incremented by $\Delta T$ and the
simulation proceeds with another call to ADDCHAIN and then PARTCO, if the
time has not reached the simulation time limit. The KSHORT routine is called
Fig. 3.1 Integration of conceptual modules (General network model)
only at user-specified intervals, the routine ROUTETM that adds the current trip times on existing shortest paths being called during the other intervals. If the time limit is reached, then the simulation stops and the statistics are computed and output.

The traffic simulation module includes three routines: PARTCO, CHANGE and RESTORE. The particle moving routine, PARTCO, checks for incidents that may have been specified to start or end during the current step in which case it calls the two routines CHANGE and RESTORE to alter the network characteristics to model incidents. It then goes over all the segments and moves vehicles to new positions. If a vehicle reaches the end of the current segment, the decisions-modelling routine, GETLINK, is called to find the next segment that the vehicle needs to move into. The next segment could be the following segment of the current arc or the first segment of a new arc, if the vehicle has reached a node. All the vehicles that reach the end of segments are flagged. Then the routine loops over all the segments once more, and moves in some of the flagged vehicles from upstream segments based on certain rules (see section 3.5.2 for the details on flow allocation rules) and leaves other flagged vehicles in a segment-end queue. At the end of the time step, the densities and speeds in the segments are calculated for use in the next time step for vehicle movement. The segment end queue lengths are also found out for trip time prediction by ROUTETM or KSHORT. More details on PARTCO will be given in section 3.5.

The path processing component consists of three routines: ADDCHAIN, ROUTETM and KSHORT. The arc trip times are found by ADDCHAIN, which finds the trip time on the arcs between nodes. This routine is needed because the K-shortest path routine needs the trip times between nodes, but the arc may be
divided into segments for traffic modelling purposes when the arc is too long. ADDCHAIN adds up the trip times on the segments that constitute each arc, in an efficient vectorized manner. Of course, if all the arcs are short enough, then segments and arcs may be identical, and ADDCHAIN could be disabled as an option.

KSHORT is the subroutine that finds the k shortest paths from the destinations to all the nodes. This routine uses the arc trip times calculated by ADDCHAIN, forms heaps with them (see section 3.3 for details) and carries out efficient heap operations to find the k-shortest paths using the label-setting algorithm described in section 2.4.2.

ROUTETM is the routine that updates the trip times on the shortest paths during the time steps between two successive calls to KSHORT. The predecessor-based route tree generated by KSHORT is traversed in this routine and the arc trip times are added. The k-th path in the list of paths found earlier by KSHORT may no longer be the k-th shortest path after this. In fact only some of these k paths are expected to be among the actual k shortest paths at the time of concern.

The third component of the framework, the decisions modelling component consists of two routines: GETLINK and BEGINRT. The routine BEGINRT is called whenever a vehicle is generated into the network to decide its initially assigned path. This routine could become more important in the future, when better initial route decision models become available, but currently it just copies a path from among the current k-shortest path set and assigns it to the vehicle\(^4\). This could be the shortest path or any randomly selected path from the

\(^4\) This path is stored in the vehicle specific array, JPATH (see section 3.2.2)
k paths, based on the option selected by the user. The GETLINK routine is called when each vehicle reaches the end of some segment. This routine first calculates the trip time on the current path of each vehicle (which is stored as a list of nodes). The alternative path trip times calculated by KSHORT or ROUTETM is then compared with this trip time. The route switch decision is made based on the individual driver's route-switch threshold, in accordance with the model explained in section 2.3.2.

3.1.2. Integration of Component Models

The communication between the above-described components is an important feature that was carefully considered during the program development. This is especially important from the point of view of flexibility and modularity of the program. Most of the communication is using COMMON storage areas, where the variables are stored for use by different routines. Appendix B shows the COMMON blocks and the subroutines that access them.

In addition to the common blocks, some of the single parameters of the simulation, such as the number of arcs, number of segments, number of destinations and the specified k for the k-shortest paths\(^5\) are transmitted between the subroutines as arguments of the call statements. For instance, the vehicle number J and the number of the segment that it currently occupies are arguments in the call to subroutine GETLINK from the routine PARTCO.

\(^5\) NARCS, N, NDESTS, and KAY respectively.
3.2. NETWORK MODELLING

The traffic network is built and stored based on the data provided externally. The data structures used for storing the network geometry are explained in section 3.2.1. The next part of the network storage consists of the arrays storing the k-shortest paths and the current paths of the vehicles, as explained in section 3.2.2.

3.2.1 Data structure for storage of network geometry

Storage of the network geometry using input data is accomplished by the MAIN program before any simulation is attempted. The main variables stored as part of the network geometry specification define the connectivity between the nodes. The external data required is information on which node (intersection, interchanges etc) in the traffic network is connected to which other nodes. In addition, external data is provided on the distances between the nodes (which are the lengths of the arcs), the number of lanes on each arc and their types (freeway, high-speed arterial, low-speed arterial etc). See Appendix C for the description of external data specifications.

The first set of data read by the MAIN program consist of the overall network characteristics such as the number of nodes, the number of arcs, number of zones, the number of destinations\textsuperscript{6} etc. Then the numbers of the zones are read and stored in an array and the numbers of the destination nodes in each zone is stored in another array\textsuperscript{7}, in the same order as the corresponding zones in the first

\textsuperscript{6} NNODES, NARCS, NZONES and NDESTS respectively.

\textsuperscript{7} Arrays IDZ and IPZ, respectively
array. Next, the program keeps all the destination node numbers in an array called IDESTS. These destination nodes may in fact be zonal centroids, specified in the input data with high-speed arcs from the real nodes (of course, the program treats these node just like any other). After that, the program reads the data on the zones within which each demand node (the zonal traffic generation is divided into these nodes) is located, and stores them in an array called IZONES. This completes the data necessary to calculate the traffic demands at each node based on the zonal traffic generation data, which is input at the end of the data file (see Appendix C).

If only a single demand node (equivalent to a centroid) is specified for each zone, and high-speed connectors are specified for nodes which are meant as the destination centroids, then it is equivalent to specifying traffic between specific O-D pairs, instead of among the zones. This is an important capability as the conventional traffic assignments techniques are always based on O-D pairs.

The arcs are specified in the input data with their upstream and downstream node numbers and other variables such as their length and the number of lanes. The arcs are numbered automatically according to the order in which they appear in the input data. If division of arcs into segments is specified as an option, then multiple segments are also created as parts of the arcs depending on whether the length of the arc is more than a specified limit (This is done to ensure sensible space discretization from the traffic simulation point of view, as mentioned before). The segments are numbered along with the arcs, with higher segment numbers associated with higher numbered arcs, the numbering being done in the order in which they are created. If the arc-division option is not specified,
the link numbers will be identical to the arc numbers\(^8\).

The MAIN program then selects the segments one by one, finds the other segments connected to it, and initializes the arrays, LINK and INLINK. The numbers of up to 6 downstream segments can be stored corresponding to each segment in the LINK array. Similarly, the numbers of up to 6 upstream segments can be stored in the INLINK array\(^9\). The reason for this kind of explicit double storage is the resulting computational execution efficiency. The LINK array is needed for some of the calculations regarding the vehicle movement into the downstream segments, while the INLINK array is needed for some of the calculations regarding flow allocation among segments upstream of a segment and incident on it (see section 3.5.2 for the details of particle movement in subroutine PARTCO). Thus, while it is possible to uniquely determine upstream connectivity from downstream connectivity data, it is much more efficient to store these explicitly so that repeated calculations are avoided. Furthermore, the storage needed for segment connectivity is expected to be much less critical compared to the path storage requirements in this modelling framework. Also stored (as LINK(1,7) and INLINK(1,7)) are the number of downstream and upstream segments. Again, the reason is to avoid unnecessary repeated calculations during simulation. See figure 3.2 for an illustration of the INLINK and LINK arrays.

The segments constituting each arc can be found once the first segment

---

\(^8\) Note that this makes sure that there are no numbers between 1 and NARCS that are not associated with any arc, and there are no numbers between 1 and N that are not associated with any segment. The former is required by the KSHORT routine, while the latter is required by the PARTCO routine.

\(^9\) It is easy to change these connectivity limits, and it may be needed if there are nodes which are simultaneously the demand and destination centroids with more than 6 high-speed two-way connectors.
SEGMENT CONNECTIVITY

Upstream segments
(upto 6 segments)

<table>
<thead>
<tr>
<th>J</th>
<th>L1</th>
<th>L10</th>
<th>L9</th>
<th>L11</th>
<th>L5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>L2</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

DOWNSTREAM SEGMENTS
(upto 6 segments)

<table>
<thead>
<tr>
<th>L5</th>
<th>L8</th>
<th>L6</th>
<th>L7</th>
<th>L1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>L4</td>
<td>L5</td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

ARC-SEGMENT-NODE RELATION

<table>
<thead>
<tr>
<th>A</th>
<th>INOD(A)</th>
<th>IDNOD(A)</th>
<th>LINK1(A)</th>
<th>LINKN(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>N1</td>
<td>N2</td>
<td>L1</td>
<td>3</td>
</tr>
<tr>
<td>A2</td>
<td>N2</td>
<td>N1</td>
<td>L4</td>
<td>2</td>
</tr>
</tbody>
</table>

Fig.3.2 Storage of network connectivity
is specified. The first segment needs to be known since the decision modelling routine needs to return the segment that the vehicle wants to move into, once the path and the first arc are selected. So the numbers of the first segments of all the arcs are stored in an array called LINK1. The following segments in an arc can be found using the LINK array, as there would only be single downstream segments. The number of segments in each chain is also stored at the beginning (LINKN array) for use in the ADDCHAIN routine for efficient vectorized computation in the ADDCHAIN routine. The upstream node corresponding to each arc is stored in the array IUNOD and the downstream nodes are placed in the array IDNOD. These arc-node-segment relationships are shown in figure 3.2. Again, it should be noted that some of these arrays could have been avoided, but considering the repeated identical calculations that would be avoided in long simulation runs, the slight inefficiency in storage (which is minuscule compared to the path storage needs) was deemed acceptable.

The nodes-to-arcs relationship is stored in a 'reverse star' form similar to the well-known forward star form. This is a very efficient way of storing node-arc relationship, and uses a pointer, NPOINT(node) to the first of the array locations storing the numbers of arcs that are incident on each node. Thus only NARCS+NNODES locations are needed to store the node-arc relationship. Note that the 'reverse star' form is different from the forward star form in that the arcs incident on a node are associated with the node instead of the arcs leaving a node. This was prompted by the fact that the shortest path routine that uses this information builds the routes from the destinations to different nodes as opposed to the usual approach of building the paths from an origin to different destinations. Figure 3.3 illustrates the reverse star storage. One comment is warranted about the
Fig. 3.3 Reverse Star storage of node-arc relationship
necessity of reverse star storage in this study, though. As all traffic networks are very sparse (i.e., low arcs/nodes ratio) this data structure does not save much storage as compared to a node-arc matrix similar to LINK or INLINK. The reason for adopting this storage was the efficiency it brings to the k-shortest path finding routine.

There are a few assumptions made about the details of the network geometry. For instance, the arcs are considered straight lines and also with constant grade, as no attempt is made to simulate the details of traffic on roads with curves and variable grades. Similarly, no data is accepted about traffic entry point configurations, as they are considered queues at mid points of segments with specified maximum service rates which depend on the number of lanes in the segments. Such details were omitted due to the presumption that they were not essential for the current purposes. Adding such features in the future is not expected to be difficult, nor would it affect the essence of the simulation-assignment methodology here.

3.2.2 Network path storage

Paths are found by the KSHORT routine at specified time periods from all the destination nodes. From each specified destination node\(^{10}\) the k shortest paths are built towards all the nodes. These paths are stored in predecessor notation in a 4-dimensional array called LASTNOD. This array stores the next node (say \(n_2\)) in the k shortest paths from each node (say \(n_1\)) towards each destination, and also the k-number (say \(k_n\)) of the path from the next node

---

\(^{10}\) These destinations are stored in the IDESTS array
onwards. This means that when the arc from $n_1$ to $n_2$ is added to the $k_m$-th path from $n_2$ we get the $k$-th path from $n_1$. It is important to keep in mind that this is slightly different from the usual predecessor storage of shortest paths. Here, the $k$-shortest path trees are built from each traffic destination node to all other nodes which is different from the usual method where the trees are rooted at origin nodes. This means that the path trees are built in the ‘opposite’ direction of traffic. Thus a predecessor node of a path from a node to a destination is in fact the successor node that the drivers would see while travelling on that route.

An illustration of the predecessor-based storage can be found in figure 3.4. Thus, as an example, if LASTNOD(1,520,6,1) is 515 and LASTNOD(1,520,6,2) is 4, then the next node in the 6th best path from node 520 to destination-1 (which is the 1st node number in the array IDESTS) is node 515 and the 6th best path from 520 to that destination is the path made up by the arc 520-515 and the 4th best path from node 515 to the same destination. The advantage of such a storage is that only two storage locations are needed to store each path\footnote{This kind of storage requires $2k$(NNODES)(NDESTS) locations and is independent of the number of nodes on these paths.}, which is more efficient than storing strings of nodes consisting of each path as separate lists.

Another type of path storage is needed for storing the current paths that the drivers perceive. This path needs to be stored when a driver selects it at the beginning of or during a trip, and needs to be kept till the next route-change decision. The difficulty here is that by the time these decisions are made, the set of paths in the LASTNOD array, of which the current path was one, would have
When IDESTS (ND) = D (i.e., ND is the Dth destination)

LASTNOD (D,N6,1,1) = N5
- (D,N6,1,2) = 1
- (D,N6,2,1) = N7
- (D,N6,2,2) = 1
- (D,N5,1,1) = N4
- (D,N5,1,2) = 1
- (D,N7,1,1) = N4
- (D,N7,1,2) = 1

so on

Fig. 3.4 Predecessor-based path storage. (Note: A predecessor node in shortest path storage terminology is in fact the successor node for a driver going to the destination, because the path tree is rooted at the traffic destination node)
been changed by subsequent updating of paths by KSHORT. This requires storing
one path for each vehicle. As these are single paths, a simple string-list of the
component nodes is sufficient. Thus the paths are stored in array JPATH\textsuperscript{12}. Every
time the path is changed as a result of the driver's decision modelled by the
GETLINK routine, the JPATH list is changed\textsuperscript{12}. One problem with this storage
is that the maximum number of nodes that can appear in a path is uncertain,
causing difficulties in reliable dimensioning of the JPATH list, which is quite
space consuming. In a square grid traffic network of $N$ nodes, the longest
'reasonable' string of nodes would be expected to have about $2\sqrt{N}$ nodes, which
can be used as a guideline for deciding the list dimension. Of course, elongated
networks may have longer strings. In the 660 node Austin-network-simulations
which are reported in the next chapter, about 60 to 65 nodes were found to be the
maximum and a dimension of 70 was sufficient.

3.3 SHORTEST PATHS

This section describes the first part of the path processing component,
namely the program to find the k-shortest paths, in detail. The theoretical
backdrop including alternative k-shortest path algorithms and the one selected here
was given in chapter 2 (section 2.4). In this section the computer implementation
is discussed. One important comment is in order about the following discussion.
The shortest path trees here are rooted at the destinations as opposed to origins

\textsuperscript{12} JPATH(J,K) where J denotes the vehicle and K the sequence number of the successive nodes
along the path. The pointer array ICURRNT(J) keeps track of the number of nodes in the JPATH
array that the vehicle has gone over.

\textsuperscript{13} A new set of nodes are copied to JPATH. Also, ICURRNT(J) is reset to 1 at this time.
which is the usual convention in shortest path literature. This may be kept in mind while reading the following sections, to avoid potential confusion.

3.3.1 Data structure for the heaps

The theoretical form of a pyramid shape heap is implemented as a priority array in the program. This means that the locations in the array have specific priority relationships among each other, the relationship being defined based on the values that can be placed in the locations. As mentioned in section 2.4.3, to form a binary heap, the Nth location should have a value smaller than or equal to the values at both the 2Nth and (2N+1)th locations. The shortest paths are found using the delete-minimum-and-reform, and add-and-reform operations (see section 2.4.3) on the heap formed from path trip times, which are discussed in the next section. In this framework, separate heaps are kept for paths to different destinations for the flexibility of using multitasking on parallel supercomputers.

The heaps corresponding to all the destinations are maintained in an array called HLIST1, which has the trip times to all the nodes that are explored at any time during the process of building k-shortest path trees from the destination nodes. This is a 2-dimensional array with the first dimension indicating the different destinations. Due to the necessity to know which path is removed during the delete-minimum operations, pointers are kept in lists, IPLIST1 and IPLIST2, with the same dimension as the heap lists. The trip time from the node stored in IPLIST1 to the destination is in the corresponding position in HLIST1. The k-number of the same path is stored in IPLIST2. See figure 3.5 for an illustration of these arrays.

When the heap-reformation operations are performed, some values in
Fig. 3.5 An example of path costs in the heap storage
HPLIST1 move from their current locations to other locations and thus it is essential to know where each path's value is stored. This is especially important because, some paths (usually the longest of the k paths from a node) may be removed from the heap due to shorter paths coming into the k-path list. Thus it is necessary to have pointers from the paths to the heap too, which is accomplished with the array NHPOINT\(^{14}\) (see figure 3.5.).

While the path trip-time heaps are kept during the shortest path enumeration, the heap is no longer required once all the paths are found. This is because after each node is closed (i.e., removed from the heap-top), a value is added to a regular 3-dimensional array DIST(D,N,K) showing the trip times on the K-th path from node N to the D-th destination. Along with this distance, the predecessor node and the rank of the path from there are stored in the corresponding two locations in the path storage array LASTNOD, as explained in 3.2.2. Only these two arrays are needed after the heap operations are over.

3.3.2. Path enumeration and heap operations

The first step performed in the KSHORT routine is the sorting of the reverse star (IFWDARC array. See figure 3.3) of each node. This step is expected to make the heap operations more efficient, as the arcs explored after each node is closed during the shortest path enumeration will have numerically increasing trip-time costs, causing smaller values to be added to the heap first. If this step is not performed, the smaller values could get added to farther spots on the heap array, and could cause more operations during the heap reformation (see section

\(^{14}\) NHPOINT(D,N,K) stores the location of the K-th path from node N to the D-th destination.
2.4.3). This step is not very critical in the current problem due to the low arcs/nodes ratio, but is still implemented for completeness of the approach.

A rough flow chart of the operations is shown in figures 3.6 & 3.7 and the KSHORT subroutine is shown in Appendix A. The heaps are first filled with a large number\(^1\). The algorithm is repeated for different destinations using a DO loop. There is absolutely no interdependency among the operations performed for different destinations (reminder: these are the tree roots, equivalent to 'origins' in usual terminology) pointing to the fact that multitasking can be employed to parallelize the algorithm over the destinations. This is not attempted here due to the unavailability of a computer with sufficient number of processors when the program was being developed. To suit the program for multitasked execution on the 8-processor machine which was expected to be available at the University of Texas soon (and is available at the time of writing of this thesis), all the arrays were written with an additional dimension to index the destinations.

As shown in figure 3.6, the heap operations are started by copying the reverse star arcs of the destination node to the first locations in the heap. As the reverse star arcs are already sorted according to their cost, they form a heap when they are copied in that order. When these values are copied to HPLIST1, the corresponding IPLIST1 and IPLIST2 values are set, as well as the NHPOINT and the LASTK values and the NNEXT value\(^1\). In the discussion from here on some

\(^1\) The large number used here is 88888.0

\(^1\) The trip time from the node that is stored in IPLIST1 to the destination appears in the heap list, HPLIST1. IPLIST2 stores the k-rank of this path. NHPOINT stores where each path appears on the heap. LASTK stores how many paths have been explored from each node. NNEXT stores the location where the next value will be added in the heap, which is updated after every path cost is added to the heap.
Sort forward star of all nodes by arc lengths

Array Initializations
Heap array (HPLIST1) filled with large number

Do for destinations, i.e. IDES = 1 to NDESTS

Copy the costs of the reverse star arcs (in IFWDARC array) of the destination in the first locations of the heap array, HPLIST1 NNEXT (heap location to insert next value) = number of arcs copied + 1

Remove the node at the first location. Fix its route cost and predecessors in DIST and LASTNOD arrays using IPLIST1 and IPLIST2 values. Place a large number at the first location

Heap empty?
no

Reform heap. Move the large value down the heap by swapping positions with the children. A new smallest value now occupies the heap top

Do for arcs in the forward star of the just-closed-node

Add arc lengths to the distance to the just deleted node - node insertion if necessary and heap reformation

Fig. 3.6 Flowchart of K-shortest path enumeration
Fig. 3.7 Path insertion logic
of the similar details will be assumed implicit and may not be mentioned.

The routine steps of the algorithm start with the removal of the minimum value, which is in the first location of HLIST1. When the minimum value is removed, the node from which that path starts\(^\text{17}\) (called NODCLOS till the next minimum is removed) will have its KCLOS value (the number of closed paths, as shown in figure 3.5) incremented by 1. Removing a value from the heap means that a large number is placed at that point. As the array is no longer a heap, this large value will have to be moved down the heap to a location where both its children are large numbers. As explained in section 2.4.3, for position 1 the children are the values at positions 2 and 3; for position 2 these are at 4 and 5 and so on. So the large number swaps positions with the smaller of its children as it moves down the heap\(^\text{18}\).

The next step is the addition of new paths to the heap. These paths are formed by adding arcs incident on the just-closed node to the path from that node. So the program loops over the reverse star of NODCLOS and checks if such paths should be added to the heap and if so where. As each arc incident on NODCLOS is examined, its upstream node\(^\text{19}\) becomes the newly explored node, called NEWNODE. The logic of path insertion is shown in figure 3.7. This step is quite complicated due to the fact that the paths are inserted in different ways depending on the current list of paths from each node. Different procedures are used based

\(^{17}\) Stored in IPLIST(D,1) after copying the initial reverse star to the heap.

\(^{18}\) Note that swapping positions involves resetting the IPLIST1, IPLIST2 arrays of the children, and the NHPOINT arrays of the paths which were at those positions.

\(^{19}\) Remember that the paths are explored from the destination nodes to other nodes, while the path themselves go towards the destination.
on whether all the explored paths are closed paths and whether the number of explored paths is smaller than K. The procedures are for inserting the path efficiently in the heap, so that the number of moves it would take to reach the heap-top is minimized and positions to which large values move down are efficiently used. In other words, the heap is kept "tight" by attempting to prevent areas of large numbers at smaller heap locations. Thus, for instance, when K paths have already been explored from a node and a new path which is smaller than at least one of them, has to be inserted, the longest path is removed from the heap and the new path is inserted there, thus preventing the additional spot with a large number that would have resulted on deleting the longest path\textsuperscript{20}.

The heap grows as more paths are added to it, but starts contracting once the rate of insertion of the paths is slower than the rate of removal of the paths, in which case the above procedure cannot prevent the occurrence of some large number spots nearer the top of the heap. While the array size used for the heap arrays are NNODES times K, the heap sizes experienced are rarely more than half of the array size in the simulations that chapter 4 reports on\textsuperscript{21}.

The heap reformation operation after inserting a path at a location is straightforward and involves comparisons and position-swapping with the parent, till a position is found where the parents are smaller. Note that no comparisons

\textsuperscript{20} Note that in this case the longer paths from the node of concern remain in the list at their old positions, but the IPLIST1, IPLIST2 and NHPOINT pointers are changed.

\textsuperscript{21} The heap size is reflected by the NNEXT variable which is the position in the array with only large numbers following it.
with the children are needed, as they are definitely larger\textsuperscript{22}.

3.4. Path trip time calculation and storage

This is the most computationally intensive component of the framework, as it is expected that this would be used much more frequently than the routine that finds the k-shortest paths. This component would conceivably be executed during every simulation time step for accurate routing of traffic, while the k-shortest path routine would be used only at specified intervals, due to its time intensiveness. The path trip time addition component adds up the updated arc trip times on the existing shortest paths during the time steps till the next set of shortest paths are found by KSHORT, and provides paths which may no longer have the same trip time rank order, but would still be reasonably short paths for the purpose of modelling vehicle routing. In the general network framework, this component is the ROUTETM subroutine, two versions of which will be discussed in this section.

While the available computing facilities suggest the development of vectorized routines to carry out path trip time calculations, the nature of the problem raises certain doubts on whether substantial vectorization can be achieved. This is because the trip time calculations are done on path trees, on which sufficient parallelism of addition operations exist only as we go farther away from the root of the tree, as there are only fewer branches near the root on which additions can be carried out simultaneously. A short discussion on these aspects

\textsuperscript{22} This is because if the new path is inserted at the position of a removed path, it is smaller than the removed path and will be smaller than its children. Also, if the path is added at the position NNEXT, which is the bottom end of the heap, there are only large numbers following it.
follows as a prelude to the following sections on the computer implementations of the ROUTETM routine.

Vectorizability could be achieved by storing the paths as separate lists and carrying out vector additions, but the number of paths involved in a large-sized problem as the one considered in this study would mean excessive memory requirements if the paths are stored separately\textsuperscript{23}, without making use of the fact that these paths share common arcs. Such storage capabilities were unavailable when the framework was being developed, and so the paths are stored with sequential predecessor pointers after the KSHORT routine finds them. Furthermore, the required number of addition operations increases linearly with the assumed maximum number of arcs on any path in the tree. If the assumed length is 100, a 100 to 1 speed-up from vectorization would be required for it to be comparable to a sequential addition methodology that is independent of the number of segments on the longest path and carries out one addition for each path using the predecessor path's length.

Two different implementations are discussed in the next two sections. Section 3.4.1 explains a sequential implementation. Section 3.4.2 explains a vectorizable routine, that still depends on the maximum number of arcs on any path in the tree, but does not assume that number, thus making it more efficient. The vectorized implementation uses indirect addressing techniques so that vectorization can be achieved with the efficient predecessor-based data structure of the k-shortest path storage, which is inherently sequential.

Two comments are needed before the discussions in the following

\textsuperscript{23} Example: For 10 paths from 600 nodes to 30 destinations with a maximum of 100 arcs per path, the storage requirement is $600 \times 30 \times 10 \times 100 = 18$ Million words.
sections, to avoid potential confusion. First, as mentioned before, the term 
'destination' here is equivalent to 'origin' in usual shortest path terminology, as 
we are considering paths rooted at the traffic destination nodes. Second, the term 
'predecessor node' is used here (and elsewhere) as in the usual terminology, but 
the predecessor node is in fact the successor node that a driver would see while 
going from a node to his destination.

3.4.1 Sequential path trip time calculations

This implementation uses a rather straight-forward tree traversal procedure 
to calculate the trip times on different paths in the tree. The methodology utilizes 
the fact that the set of k-th paths from all the nodes to a destination are always in 
the form of a tree\textsuperscript{24} even though the set of all k-shortest paths from all nodes to 
a destination need not be a tree\textsuperscript{25}. Furthermore, the predecessor path-pointer of 
the k-th path from any node to a destination can only point to a path with rank 
smaller than or equal to k\textsuperscript{26}. A third aspect used is that when a tree is made 
using the label-setting algorithm the node at the end of a branch (start-node of the 
"leaf" arc, in graph theory terminology) closes after all the other nodes that are 
on the path from that node to the destination.

One problem that is encountered here is that the arc trip times cannot be 
added along a path from a destination, because only the predecessor nodes are

\textsuperscript{24} i.e., there is only one predecessor path.

\textsuperscript{25} Because there could be paths of different ranks from a node to a destination that have different 
predecessor nodes, which is against the definition of a tree.

\textsuperscript{26} Proof: If this were not true, more than k-1 paths, which are shorter than this k-th path, could 
be found by adding the arc of concern to those paths from that predecessor to the destination
defined for a path and so the trip times can only be added from a node along the predecessor-chain towards a destination and not vice versa. This problem could not have been prevented by using pointers towards successors as there could be more than one k-path from the successors passing through the node of concern as opposed to a single pointer to a single predecessor path in the storage scheme used here. Thus, the additions can be done only from the end-nodes of branches towards the destination. The arc costs are added along the chain of predecessor nodes and this results in one trip time value which is for the path from that "leaf" node to the destination.

The methodology implemented here uses a two-pass technique to add trip times. Thus the addition is carried out from each branch end to the destination for the path trip time from that branch-end, followed by a second pass when arc-cost are subtracted along the way for path trip times to the intermediate nodes. When the chain of additions from a branch-end reaches a node from which the trip time on a path to the destination is already calculated, the total path trip time cost is calculated using the path trip time from that node, and the second pass starts from the branch end. Thus no node is visited more than twice during the trip time calculations (for each destination and each k-path).

The branch ends are found from an array NKORDER which stores the node numbers in the order of their closing during the development of the tree of the k-th paths, in KSHORT. When NKORDER(D,J,K) = M, it means that the K-th path from node N to the D-th destination was the M-th path to be closed from among the set of K-th paths. Thus the last node is definitely a branch end. The trip time calculations start from here and proceed towards the destination.

Once one branch is completed, higher nodes in the NKORDER array are
checked. The first node whose path trip time has not been calculated that is found on moving up in the list from the previous branch end is the next branch end\textsuperscript{27}. This node has to be a branch-end due to the property that the all the other nodes along a branch closes before the branch-end during label-setting path enumeration.

A branch in the \( k \)-th shortest path tree could end at a node with a pointer to a path that has a lower rank, and for this reason, the path trip times are found for \( k=1,2,3,... \) onwards, using a DO loop. The entire procedure is repeated for different destinations with an outer DO loop. Some computational results on this path trip time calculation method are provided in Chapter 4.

### 3.4.2 Vectorized path trip time calculations

This routine uses DO loop vectorization capabilities as well as the indirect address vectorization capabilities of the CFT77 Fortran compiler on the Cray X-MP and Y-MP supercomputers. The vectorization is done for each destination and each \( k \)-th path tree separately, with two outer loops incrementing the destinations and the path-order for each destination. The vectorization is performed over the nodes in the network during each iteration.

For each destination and each path-tree of rank \( k \), the algorithm starts by setting the total trip time cost from all nodes in the network as zero, in the DIST array. Also, the distance change indicator array called IC is filled with zeros corresponding to all the nodes except the destination. Then the iterations start and the steps are as follows (see figure 3.8). For simplicity, the destination and path

\textsuperscript{27} Note that the nodes appearing between this branch end and the previous branch end in NKORDER are the nodes on the path from the previous branch end to the destination, which have already been visited.
Fig. 3.8 Indirect address vector operations for iterative path trip time calculations
indices of the arrays are not shown.

Step 1: Vector loop 1 - Copy the distance array to A (array of old costs in figure 3.8).

Step 2: Vector loop 2 - Update the distances to all nodes by adding the arc distance to the predecessors (D(NKARC)) multiplied by the difference indicator IC to the DIST array of predecessors (indirectly addressed with predecessor node pointers). i.e.,

\[ \text{DIST}(M) = \text{DIST}(\text{LASTNOD}(M)) \]
\[ + D(\text{NKARC}(M)) \times \text{IC}(M), \quad \text{For all nodes } M \]

Step 3: Vector loop 3 - Compare the old cost array and the new one.

\[ \text{B}(M) = \text{A}(M) - \text{DIST}(M) \]

Step 4: Vector loop 4 - Set IC(M) as zero, for all M.

Step 5: Vector loop 5 - If the difference B(M) is more than zero for any M, Set flag for next iteration, and place 1 at the IC location.

\[ \text{IF } (\text{B}(M) \gt 0) \quad \text{IFLAG}=1; \quad \text{IC}(M)=1 \]

Step 6: If IFLAG = 1, set IFLAG = 0 and go to Step 1.

If IFLAG = 0, Stop.

Certain comments are in order here. First, the algorithm cannot have more iterations than the maximum number of nodes in any path. This is easy to see, because in every iteration the arcs being added are one level below the previous one (see fig 3.8) and the maximum number of levels is the maximum number of nodes on any path.

The most important vector loop is the loop in step 2. Here, the multiplication and value assignment operations are not carried out between values in corresponding array locations but between the values in certain array location
and the values in different locations whose addresses appear in the corresponding locations. Thus the LASTNOD points to the values in a different location, which is the distance to the predecessor node. Though this may seem to inhibit vectorization (and indeed may, on some vector processors), this does vectorize rather efficiently on the CRAY. This is due to the fact that the Gather/Scatter operation of collecting array values from appropriate locations for operations and placing them back when indirect addressing is used, is a hardware feature of the CRAY supercomputer.

Even though the number of operations within each loop is of the order of NNODES, the loop execution time increases much more slowly than the number of nodes, and for numbers above a few hundreds, the vectorization can provide excellent results. However, in the few comparative simulations carried out with this algorithm, the execution times were found to be two to three times those with the sequential algorithm. These results are being reported as this algorithm may not have been coded efficiently enough in addition to it being heavily dependent on the particular computer hardware configuration. It is doubtful, however, if it could be coded efficiently enough for it to perform faster than the sequential code.

3.5. TRAFFIC FLOW MODELLING

This section explains the traffic simulation component of the framework. Section 3.5.1 explains how the traffic demand data is transformed into the arc traffic loading which is the input into the simulation. Section 3.5.2 explains how vehicles are generated into the entry queues and how they are moved in the network. Section 3.5.3 elaborates on how the queue delays are calculated and section 3.5.4 discusses the incident simulation capability.
3.5.1 Traffic Loading

The traffic loading details in the program have been designed offer the capability of performing simulations based on origin-destination (node to node) traffic generation data as well as interzonal traffic demand data. Treatment of zone to zone traffic using zonal centroid nodes is the conventional method used in transportation planning, and the compatibility of the program with this method is important. To use dynamic demand data between all the nodes could be impractical in a large network. At the same time, also of importance is the capability to simulate small networks as well as partial networks (just the freeway network of a city being simulated, for example) where node to node origin-destination demand data may be available. The traffic data input and vehicle generation method developed are discussed below.

Each highway segment in the network receives the nodal traffic generation from one node (either the upstream or the downstream node of the arc that it is part of) or from none at all, as specified in the input data. The zonal traffic is divided into all the segments that receive traffic from nodes which are specified as demand nodes in the zones. As mentioned in section 3.2.1, based on how the data is specified the various cases discussed above can be handled. Zone-to-zone traffic with centroid nodes as origins and destinations can be simulated by specifying only highspeed centroid connector arcs in each zone to receive traffic and specifying only the centroid nodes to be the demand nodes in each zone. At the same time, by specifying one zone corresponding to each node, strict node to node traffic can be generated. As a third option, the zonal traffic can be evenly split for generation from all the arcs in the zone by not specifying any highspeed connector arcs or centroid nodes, and specifying all the highway segments to
generate traffic.

Before the simulation starts, this data is transformed to the traffic volume
generation rate, which is used to generate vehicles at the mid-point of each
segment in each arc. This is carried out as explained next. In this discussion, the
phrase 'demand into an arc' refers to the total vehicle entry along its length at
mid-points of its segments and should not be confused with the traffic flow from
other arcs incident on it.

In the data, each node is specified to be in a particular zone in the
network and receives traffic generation based on the demand from that zone. Each
arc is considered to have mid-block traffic generation depending on the node that
it is specified to be 'affiliated' to for traffic generation purposes. ILDEM array
stores this 'affiliation', by storing 1 if it is to the upstream node, 2 if it is to the
downstream node and 0 if to neither node (meaning there is no vehicle entry along
the arc). As each node is in a particular zone the segments receiving the demand
from each zone are known28.

The total zone-to-zone trip demand during each demand interval is input
as part of the data (see appendix C) and is read into the matrix array ZDEM. The
first step in the program is to calculate the fraction of total traffic generation in
each zone that is destined to each other zone during each demand interval29. At
the beginning of each demand interval, the expected traffic volume generated in
each zone and destined to each other zone is calculated. This is further divided

---

28 Stored in the array IZLIN. The total number of segments receiving traffic in each zone is stored in the array IZLIN.

29 Stored in array ZFDEM(from-zone, to-zone, interval)
into the segments generating traffic in each zone\textsuperscript{30}, resulting in expected numbers of vehicles generated in each segment during each time steps of the demand interval. As traffic volume has to be an integer number, the integer part of this expected number for a segment is generated every time step. The fractional part is handled by calling a random number for each segment in each time step to decide if an additional vehicle has to be generated. This technique avoids the round-off problems that otherwise arise\textsuperscript{31}.

As the vehicles are bunched together to form 'macroparticles' only at the segment entry, the above-said segment entry demand cannot be split into vehicles tagged with their destinations when they are assigned to different segments. When these macroparticles are formed at the segment-entries their destination zone is selected based on the fractions of the demand that goes to each other zone from the zone where they are generated, using a uniform random number. The destination node of the particle is selected and tagged with the particle. The vehicle generation and their subsequent moving is done in PARTCO subroutine, which is discussed next.

3.5.2. Traffic flow modelling component

The traffic flow modelling component has five main parts: 1) vehicle generation, macroparticle forming and entry into either the entry queue or the segment, 2) moving of particles within segments, 3) calculation of segment-to-

\textsuperscript{30} At present, this division is based on the lane miles in each segment relative to the total lane miles in the zone. It is easy to change this to equal division or a division based on some other criteria.

\textsuperscript{31} In fact, for most of the segments the expected number of vehicles generated during a time step in a realistic simulation could be much smaller than 1.
segment vehicle flux constraints, 4) moving of particles from one segment to another and 5) updating of segment condition at the end of the time step. These will be explained in that order in this section. It should be noted that the macroparticle simulation approach is maintained in the program, even though individual vehicles can be simulated with available computational resources, due to its potential usefulness in cheaper computing, if needed. Individual vehicles can be simulated by specifying the macroparticle size as one\textsuperscript{22}, and adjusting the sizes of certain arrays to handle a higher number of particles. In fact, this is the approach adopted for the simulations reported in chapters 4 and 5.

3.5.2.1 Part generation block

The vehicle generation and vehicle movement within each segment are carried out in PARTCO during a first loop over the segments. For a particular segment, the vehicle generation block is shown schematically in fig 3.9. Each segment has an entry queue of finite size and FIFO (first-in-first-out) discipline to store the macroparticles waiting to enter at its mid-point. The number of particles that will be entering during the time step in the segment is calculated as the minimum of three quantities: available capacity in the segment at the end of the previous time step, the new demand plus the existing demand and the maximum allowed entry volume itself.

The existing vehicle demand is available in terms of number of vehicles and in terms of number of macroparticles in the existing queue\textsuperscript{33}. The maximum

\textsuperscript{22} The MINUM variable specified in the DATA statements in MAIN shows the particle size.

\textsuperscript{33} The former in the array GEN and the latter in NTRYQ. These two arrays may reflect vehicle numbers that are different by less than the particle size, as some vehicles may not have yet formed into macroparticles in the previous time step.
Calculate available capacity

If (link density > 2/3(jam density)) entry limit halved

Add the particles in the queue that can enter the segment into the vehicle array, MTXJ, in the top free locations (of value 0)

If queue exists

Calculate the number of particles that will enter the segment (MNUM) i.e. min (available capacity, existing queue + new arrival, link entry limit)

if capacity is still available

Form particles from the newly arriving vehicles and add them to the segment vehicle array, MTXJ till the segment capacity is used up.

no available capacity for any newly arriving vehicles

Form particles from newly arriving vehicles and add them to the segment entry queue array MTQJ at the top available locations (with value 0)

Call random number routines and decide if each of these vehicles are equipped for information or not and how much their route switch indifference threshold fraction is.
Decide destination node using inter zonal demand.
Call BEGINRT to assign the initial path

Update the # of particles in the queue, NTRYQ(I) and total number of vehicles waiting to enter, GEN(I)

Go to
Particle movement (within segment) block

Fig. 3.9 Vehicle entry into the highway segments
allowed entry volume is the limit specified for ramps if the segment is of a freeway and the assumed value for entries along the segment for other segments. Different entry limits can be input for different segments as part of the data. When the segment density goes above 2/3rd of the maximum, this entry limit reduces to half to reflect reduced entries during congestion.

As the vehicle entries are calculated before the vehicles are moved into each segment from upstream segments (the second loop over the segments that is the 3rd part of traffic simulation as mentioned above), the implicit assumption is that vehicle entries have precedence over the mainline traffic, which is somewhat true for highways, especially freeways. When the maximum concentrations are not reached, such entry priority does not come into play.

As shown in figure 3.9, once the the number of vehicles that will enter\textsuperscript{34} is calculated, several different conditions are possible, depending on whether there is an existing queue and whether the queue will be completely cleared given the current allowed entry. Based on the appropriate conditions as shown in figure 3.9, the macroparticles are either placed in the queue or on the segment\textsuperscript{35}. In both cases, all their attributes are assigned when they are created. These include: the destination, initial route, whether they receive information or not, the start time, the behavioral model parameter $\eta_j$ (see section 2.3.2) and a tag showing if the particle entered during the specified period over which the statistics are accumulated.

\textsuperscript{34} The variable MNUM

\textsuperscript{35} Note that when the particle is generated into the segment they are placed in an array, MTXJ storing the unique particle number $J$. If it is generate d into a queue, the particle number is stored in a similar array MTQJ
The destination of the particles\(^\text{36}\) are determined by calling a random number as explained in section 3.5.1. Then the subroutine BEGINRT is called to decide the initial route\(^\text{37}\). A uniform random number is called and based on the fraction of vehicles receiving information\(^\text{38}\) it is decided if the particle receives information or not\(^\text{39}\). If it receives information, then a random number with a triangular distribution is called to decide the particle’s route switching propensity (parameter \(\eta_j\), see section 2.3.2). When the particle enters the segment its position is set at the mid-point of the segment\(^\text{40}\). If a queued particle enters the segment, it is removed from the queue array and added to the segment array\(^\text{41}\). If it is specified that the start time on the segment only is of concern\(^\text{42}\) then its start time is also reset.

The last step in the particle generation block is to update the queue lengths after the vehicle generation\(^\text{43}\). Once the particle generation is over, the

\(^{36}\) The position of the destination node in the IDESTS list of destinations is stored in JDEST(J) for particle J.

\(^{37}\) Returned from BEGINRT as a list of nodes in the array JDEST.

\(^{38}\) Specified as FRACINF in the DATA statements in PARTCO.

\(^{39}\) Stored in INFO(J) as 1, if it receives information and 0 otherwise.

\(^{40}\) As the particle position XPAR(J) is defined as its distance from the end of the current segment, this means that XPAR(J)=S(I)/2 when particle enters segment I of length S(I)

\(^{41}\) i.e., A zero is placed at the location in MTQJ where the particle was, and its number J is placed in MTXJ at the top-most location with zero currently.

\(^{42}\) i.e., If IQWAIT is specified as 0 in the DATA statements of PARTCO

\(^{43}\) Here GEN(J) is decreased by MNUM*MTNUM vehicles and NTRYQ(I) is decreased by MNUM particles.
PARTCO routine proceeds to the particle moving block, which is explained next.

3.5.2.2 Particle Moving Block

The particle moving block is the main component that uses the traffic flow relations explained in section 2.2.1 to move the particles within the segment. The procedure is shown in figure 3.10. This part is carried out for each segment, during the first loop over the segments in PARTCO.

For each segment, the particle array is examined, and the particles whose numbers appear in that array are moved one by one. The first check for each particle is whether it would reach the end of the segment during the current time step if it is moved at the current speed calculated at the end of the previous time step. This is done by means of the tentative new position\(^{44}\) defined as a distance from the end of the segment.

If the tentative position value becomes negative, it means that the particle will reach the end of the segment during the current time step. In this case, if the node it reaches is its destination node, the particle is removed from the system. Otherwise, the decision routine GETLINK is called to find the next segment that it needs to move into. Of course, if there is only one segment downstream as in the case of interior segments of an arc, the next segment is the downstream segment in the segment-connectivity array LINK (see section 3.2.1) corresponding to the current segment. If the tentative position value is a positive, the particle is moved to a new position by simply updating the particle position as the tentative position.

When a particle is to be moved into another segment during the current

\(^{44}\) Variable XPOS calculated as XPAR(J) - V(I)\*\Delta T, where XPAR(J) is the current position and V(I) is the segment speed calculated at the end of the previous time step.
for segment l

Particle = J (found from MTXJ array)

If the position of the particle is reduced by (V(l) * time increment), will it be going past the segment end?

yes

Has the vehicle reached the destination node?

yes

no

Update the vehicle position, XPAR(J) by the distance it moves

no

Call the decision modelling routine, GETLINK to find the segment it needs to move to, i.e., NEXLINK(J)

Remove the vehicle from the segment. Add the time needed to reach segment end to its trip time TTILNOW(J)

Increase the next segment’s upstream demand, INTOQ(NEXLINK(J)) by 1.
Store the vehicle number, its position in MTXJ of l and the the time of its arrival at link-end in INTOXI array of NEXLINK(J)

Particle = next in MTXJ(l)

no particles left to be moved in l

l = l+1

Fig. 3.10 Particle Moving Block (Within Links)
step and is temporarily positioned at the segment-end, its number J is added to a 3-dimensional array INTOOI (see fig 3.11) corresponding to the next segment\textsuperscript{45}. This array stores a set of three numbers for each particle, which are 1) the current segment of the particle 2) the particle number and 3) its position in the particle array of its current segment. These have to be stored so that the upstream segments' particle arrays can be changed properly when the particles from them are moved in during the second loop of PARTCO. The time remaining when the particle reached the segment-end is stored in TLEFT(J), for use in deciding the order in which the particles are to be moved when the second block of PARTCO routine (for moving the particles across the segments) loops over the segments. In the the particle has reached the destination and is removed from the system, the particle's trip time is noted\textsuperscript{46}.

There are also certain variables being updated at appropriate places during the moving of the particle for system performance calculations based on congestion experience. These are the variables storing the total time during which the particle moved through congestion, the total distance it travels through congestion and the total number of congested stretches it moved through\textsuperscript{47}, congestion being arbitrarily defined as the state when the density is over 2/3rd of the jam density.

\textsuperscript{45} An array INTOOI stores how many vehicles are already waiting to enter that next segment, which is the NEXLINK of the particle. Thus the specifics of this particle are added to the INTOOI array in the (INTOOI(NEXLINK)+1)th position.

\textsuperscript{46} The variable TITLNOW(J) storing the time the has been in the system is incremented by the time it took for it to reach segment-end during this time step.

\textsuperscript{47} CONGEST(J), CONGESD(J) and CONGESN(J) respectively.
Particle array for upstream segments

\[ MTXJ (L2, 1) = P1 \]
\[ \quad \text{"(L2, 2) = P2} \]
\[ \quad \text{"(L2, 3) = P3} \]
\[ \quad \text{"(L3, 1) = P4} \]
\[ \quad \text{"(L3, 2) = P5} \]

Particle numbers

Upstream Demand array for segment L1

for the first particle added to INTOOI
\[ \quad \text{"(L1, 1, 1) = L2} \]
\[ \quad \text{"(L1, 1, 2) = P1} \]
\[ \quad \text{"(L1, 1, 3) = 1} \]

粒子位置在MTXJ阵列中的上游段

segment number
particle number
particle position in the MTXJ array of upstream segment

second
\[ \quad \text{"(L1, 2, 1) = L2} \]
\[ \quad \text{"(L1, 2, 2) = P2} \]
\[ \quad \text{"(L1, 2, 3) = 2} \]

last
\[ \quad \text{"(L1, 5, 1) = L3} \]
\[ \quad \text{"(L1, 5, 2) = P5} \]
\[ \quad \text{"(L1, 5, 3) = 2} \]

INTOO (L1) = total demand = 2+3 = 5

Fig. 3.11 Particles stored for movement into a new link during current time step
3.5.2.3 Calculation of segment-to-segment vehicle flux constraints

As mentioned in the last chapter (section 2.2.2) calculation of flux constraints is necessary to prevent unrealistically large numbers of vehicles from moving from segment to segment, which might happen during quick dissipation of downstream congestion, especially in the case of abrupt capacity restoration after incidents. So flux constraints are calculated for all the segments upstream of each segment. These are calculated as the maximum number of particles discharged from the queue per time step assuming a queue discharge rate. However, the flux limit for a segment is active only when there were queues longer than the flux limit during the previous time step.

3.5.2.4 Segment-to-segment movement of particles

This block within the PARTCO routine carries out the second loop over the network segments and moves the particles that need to move in from the upstream segments during the current time step. The logic of this part is shown in figure 3.12. The entire procedure is based on the INTOOI array that stores the numbers of the particles that have reached the upstream segment-end, their current segment’s number and the position of the particle in the particle array of that segment. For each segment I, the first step is to calculate the number of particles that can move in during the time step. The variable NC shows this number and is the minimum of the total number in the INTOOI array and the available particle capacity of the segment. Whether NC particles will move in or not depends on the

---

48 The rate is specified in a variable IQDISP. A rate of 3000 vehicles per lane per hour was assumed for the simulations in this research.

49 The flux limits are stored in an array LIMFLUX(K,L), where I is the segment of concern and K stands for the K-th segment appearing in the INLINK connectivity array for segment I. LIMFLUX array will have a large value during time steps when the flux limits are not active.
Fig 3.12 Movement of Particles (Segment-to-Segment)
flux limits for movements from each of the upstream segments. The algorithm loops over the INTOOI array NC times selecting at most one particle at a time, based on how long it has been waiting\(^{50}\). Once a vehicle is selected, it is moved into the segment I if the flux limit for movement from that upstream segment hasn’t been exceeded. Otherwise, the vehicle stays in the upstream segment in the queue\(^{51}\). If the vehicle can move in, then it is also moved to a new location in segment I based on the time remaining for the vehicle. This time is stored in TLEFT(J) which has a value equal to the simulation time step if particle J has been waiting at the upstream segment-end from the last time step, or a value less than that if it reached the upstream segment-end during this time step. Some of the other variables changed during these operations are shown in figure 3.12.

3.5.2.5 Segment condition updating

At the end of the time step, the new speeds and densities in the segments are calculated for the next time step. This is done using the NPAR array that stores the number of particles in each segment after all the particle moving operations are over. The speeds are calculated based on the modified non-linear Greenshield’s equation explained in chapter 2 (section 2.2.1).

Another significant updating operation is the compressing of the particle array, MTXJ, which has zero values at the locations previously occupied by particles which moved into other segments during this time step. The numbers of all the vehicles currently present in the segment are moved up to the top positions in the MTXJ array as a contiguous list, so that in the next time step only those

---

\(^{50}\) Stored in IQWAIT(J) if there is a queue or TLEFT(J) when IQWAIT(J) is zero.

\(^{51}\) In either case, the position in INTOOI that the vehicle occupies is tagged as selected, i.e., ISEL(spot in INTOOI) = 1.
positions are looped over to move particles. As the MTEX array may have fewer particles than its size, which is set to accommodate the maximum number vehicles in any segment in the network, these array compression operations could be quite time-consuming. An efficient vector code was written for this part, which is discussed along with a comparison of a sequential code, as part of computational results in chapter 4 (section 4.4.3.1)

3.5.3. Segment trip time and segment-queue delay calculations

The segment trip times including the delay in the queue at the segment-end have to be calculated for use in modelling the supply of route travel time information to the motorists. The ADDCHAIN routine adds up these segment trip times over different arcs. These arc trip times are used by KSHORT or ROUTETM to calculate the route trip times, which in turn are needed for the GETLINK subroutine to model the driver selection of routes when current route trip times are shown. So these segment trip times are calculated at the end of each time step.

The segment trip time is composed of the time required for moving through the length of the segment and the queue waiting time. The moving time can be calculated based on the current speed in the segment, but the current queue delay calculation requires certain assumptions. The assumption here is that the current queue delay in the segment is equal to the time required to clear the existing queue at the queue service rate experienced in the past over a certain period (say 5 minutes). Thus, for segment \( i \) at time \( t \),

\[
t_i = t_{sq} + t_{sw}.
\]
\[ = \frac{Q_u \cdot T}{N_T} + \frac{S_t}{v_i'} \]

where,

\( t_u \) = Current queue waiting time

\( t_m \) = Current moving time (to cover the segment length)

\( T \) = min(assumed period for queue service rate calculations, the length of time when the current queue existed)

\( N_T \) = Number of vehicles cleared in time \( T \).

\( Q_u \) = Current queue length in vehicles

\( S_t \) = Length of the segment

\( v_i' \) = Current speed in the segment

The period over which the average queue service rate is calculated is specified by the user\(^2\). For each segment, the number of time steps over which a queue of at least 2 particles per lane existed is stored as well as the actual number of vehicles cleared for each of those time steps\(^3\). Once the period with a queue is over the specified service rate calculation period, the queue service value for the earliest step is removed from the array, the remaining numbers are moved up and the number of vehicles cleared in the current step is added as the

\(^2\) As NQS (in simulation time steps) in the DATA statements in PARTCO.

\(^3\) Arrays NQACCUM and NDQR respectively.
last value. Thus it is possible to find a moving average of queue service rates. To prevent the possibility of unrealistically small queue service rates because the queues that existed in the previous time steps were smaller than could be cleared during those steps, a minimum rate is also used, assuming a saturation flow rate of 1500 vehicles per hour per lane.

3.5.4. Incident Modelling

This part of the simulation program is essentially unchanged from the original single-highway simulation program developed to study the effect of incident-perturbations and the day-to-day dynamics under commuter trip time decisions (see Jayakrishnan, 1987; Mahmassani and Jayakrishnan 1988). The perturbations are modelled completely based on external data provided. The program looks for such data and activates the capacity perturbations modelling block if so specified by the user\(^4\).

Incidents can be specified to occur at any time during the simulation on any number of segments. More than one incident can also be specified on any number of segments. For each incident, the segment on which it occurs, the starting and ending time and the factor by which the segment capacity is reduced (in terms of a reduction in the total lane-miles in the segment) needs to be specified.

The capacity reductions and restorations are handled by the subroutines CHANGE and RESTORE which are called from the MAIN program during the particular time steps when they are needed, before calling PARTCO to move the

\(^4\) The IPERT in the DATA statements in PARTCO is set to 1, for incident simulations.
particles. The logic behind the incident modelling approach is straightforward, as shown in figure 3.13.

3.6. BEHAVIOR MODELLING

The two subroutines that handle the modelling of driver behavior are BEGINRT and GETLINK. The first subroutine decides the initial path of the particles when they are generated, and copies the node numbers along that path into the vehicle's current path array. The latter subroutine models the decisions of each driver to switch to an alternative route or stay on the current route, at each node along his/her journey. If the driver is not equipped for information GETLINK just returns the next segment in the driver's current path, which would be the original path assigned to the driver. It should be noted that while the traffic is composed of macroparticles only, these two decision modelling routines assume each macroparticle to be an individual decision unit, just like an individual driver. Of course, when the macroparticle size is one, as is the case in the simulations reported in this study, this is not an issue.

The BEGINRT routine currently determines the initial route according to one of two options selected externally. One is to consider the current best path from the downstream node of the particle's generation segment to be the initial path selected. The other is to randomly pick from among the best $k'$ paths (where $k' \leq k$, the number of k-shortest paths available). These options are specified with a value 1 or 0 for the indicator variable IPINIT in the DATA statements in the
Fig. 3.13 Incident Simulation logic
BEGINRT subroutine. If IPINIT is 0, then a value should be specified\textsuperscript{55} for $k'$. Another feature of the BEGINRT routine is that it uses the set of paths stored at the end of the simulation start-up time rather than the current $k$-shortest paths. This facility is useful for certain controlled simulations where regardless of the information scenario, the drivers get the same initial routes, which means that only the enroute decisions influence the system\textsuperscript{56}. It is very simple to modify this part to simulate the case of drivers starting with initial routes based on information on the current conditions.

The GETLINK routine is called from the particle movement routine, PARTCO, whenever a particle reaches a node and its next segment has to be found out. If the particle does not receive information, the GETLINK just looks at the next node in the particle's current-path list\textsuperscript{57} to find the next node. The arc from the current node to that node is found from the reverse star of that node after which the segment to move to is found easily\textsuperscript{58}.

If the particle receives information, then GETLINK first calculates the trip time on the current path by adding up the trip times on the arcs constituting the current path of the particle, which may no longer be a path existing in the $k$-shortest path list. The trip time on the best displayed path is found by sorting the trip times on the $k$ paths to the destination from the current node. These two trip

\textsuperscript{55} Specified as the variable NR in BEGINRT.

\textsuperscript{56} The predecessor path array LASTNOD is copied to LASTN01 at the end of the start-up period and is used by BEGINRT.

\textsuperscript{57} i.e., at the ICURRNT(I)th position the JPATH array of particle J.

\textsuperscript{58} From the LINK1 array location for that arc.
time values are compared to decide if the particle switches routes. The decision is based on the behavior model described in section 2.3.3 which involves the particle's indifference towards minimal route trip time advantages and the absolute minimum advantage that it seeks\textsuperscript{39}.

If the particle is switching routes, then the set of nodes in the best path is copied to the particle's path array. The pointer showing how far along the sequence of nodes in this array the particle has travelled, is set to one. The first node in this path array is the current node and the second node is the downstream node of the arc that the particle has decided to move to\textsuperscript{40}. The first segment of the next arc is found by looking at the reverse star of the second node the path array. If the particle is not making a switch, the path array and the pointers are unchanged and the next segment is found just as in the case of the particles not receiving information.

This concludes the discussion on the details of the behavior modelling component, as well as this chapter describing the important programming and implementation details of the various component modules of the general network simulation program. The next chapter discusses the experimental network simulations carried out with this framework.

\textsuperscript{39} RIBF(J) and BOUND respectively.

\textsuperscript{40} It is important that the current node is still pointed to, because the vehicle may not be able to move into the arc that is selected due to traffic capacity problems, in which case the decision routine is called again in the next time step and a new path may be selected. Thus ICURRRT(J) will point to the downstream node of the next arc only after the particle moves into it.