

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

**Development of a Dynamic Shortest Path Algorithm and a Traffic
Simulation Model**

Arun Shankar Bhowmick

A Thesis

In

The Department

of

Building, Civil and Environmental Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Masters of Applied Science at

Concordia University

Montreal, Quebec, Canada

June 2005

© Arun Shankar Bhowmick, 2005



Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

0-494-16246-5

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*

ISBN:

Our file *Notre référence*

ISBN:

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.


Canada

ABSTRACT

Development of a Dynamic Shortest Path Algorithm and a Traffic Simulation Model

Arun Shankar Bhowmick

Shortest path determination in a dynamic transportation network has been a real challenge where network traffic scenario changes continuously. The shortest path between any two nodes in the network depends not only on the travel time of links that constitutes the path but also on the time spent at the intersection which is a major part of the trip time. Existing algorithms found in the literature consider only the link travel time for the shortest path calculation between two nodes. A new algorithm is introduced in this study that calculates the dynamic minimum trip time path between two nodes by considering both link travel time and intersection turning delay. A macroscopic traffic simulator is developed to simulate vehicles that use the proposed algorithm to calculate the minimum trip time path. The trip time of guided vehicles which determine the travel path based on en-route traffic information is compared to the trip time of unguided vehicles which determine the travel path based on pre-trip traffic information. The guided vehicles are assumed to be a part of a decentralized traffic management system where on-board route guidance system allows the driver to calculate the minimum trip time path each time they cross an intersection. A part of Montréal city road network is also simulated to show the application of proposed path finding algorithm. From the results it is seen that for a long distance trip the amount of trip time savings is higher because the guided vehicles usually choose the path with less number of left turning delays.

ACKNOWLEDGEMENTS

I would like to extend my sincere gratitude to Dr. B. Ashtakala, for being my supervisor and helping me through the development of this thesis. I am very much thankful to him for his guidance and the encouragement during my graduate studies in this university. He has timely extended his vast experience and advice to encounter different problems and guided me towards the successful completion of the thesis. I am greatly indebted to him for the financial assistance that he provided to me for completing my graduate studies.

I would like to dedicate this work to my beloved parents. Their blessings and firm belief in my abilities have made my studies possible.

TABLE OF CONTENTS

	Page No:
LIST OF FIGURES	ix
LIST OF TABLES	xi
LIST OF SYMBOLS	xii
CHAPTER 1 Introduction	
1.1 Background	1
1.2 Problem Definition	2
1.3 Objectives and Structure of the Thesis	2
CHAPTER 2 Literature Review	
2.1 Dynamic Route Guidance	4
2.1.1 Past and Present Approach to Routing	4
2.1.2 Comparison of Computer Networks and Transportation Networks	5
2.1.3 Deterministic and Adaptive Route Guidance Architecture	5
2.2 Shortest Path Approaches in Transportation Networks	7
2.2.1 Basic Assumptions of Shortest Path Problems	7
2.2.2 Static Shortest Path Algorithms	8
2.2.3 Dynamic Shortest Path (DSP) Algorithms	8
2.2.3.1 Programming Formulation for DSP Algorithms	9
2.2.4 Path Search Techniques	13
2.2.4.1 Labeling Technique	13
2.2.4.2 Heuristic Search	14
2.2.4.3 Bidirectional Search	14

2.2.5	Multi-Objective Shortest Paths	15
2.2.6	A Comparative Study of Popular Shortest Path Algorithms	16
2.3	Implementation Issues for Shortest Path Algorithms	17
2.3.1	Network Representation	18
2.3.2	Different Data Structure	19
2.3.3	First-in, First-out(FIFO) Network Properties	20
2.4	Discussion	20
CHAPTER 3 Dynamic Shortest Path Search Algorithm		
3.1	Introduction	21
3.2	Theoretical Concept	21
3.3	Minimum Trip Time Path Algorithm	22
3.3.1	Formulation of the Algorithm	22
3.3.2	Assumptions	23
3.3.3	Search Strategy	23
3.3.4	Description of the Algorithm	29
3.3.5	Implementation of the Algorithm	33
3.4	Discussion	38
CHAPTER 4 Traffic Simulation Model		
4.1	Introduction	39
4.2	Simulator Scenarios	39
4.2.1	Description of the Network	39
4.2.2	Modeling Traffic Dynamics	40
4.2.2.1	Modeling Concept	40

4.2.2.2	Traffic Stream Characteristics	43
4.2.3	Signal Timing Cycles	44
4.2.4	Travel Demand Characteristics	44
4.3	Simulator Structure	45
4.3.1	Loading Phase	47
4.3.1.1	Network Generation Sub-Model	47
4.3.1.2	Input Data Sub-Model	50
4.3.2	Update Phase	50
4.3.2.1	Event Scheduler Sub-Model	50
4.3.2.2	Traffic Flow Propagation Sub-Model	53
4.3.3	Advance Phase	56
4.3.3.1	Left Turn Movement	57
4.4	Discussion	58
CHAPTER 5	Computer Programs	
5.1	Introduction	60
5.2	Correlation of the Problem Domain to an Object-Oriented Data Model	60
5.3	Object-Oriented Approach	61
5.3.1	JAVA Implementation	62
5.4	Class Definitions and Relationships	62
5.5	Summary of the Computer Programs	76
CHAPTER 6	Results of Traffic Simulation Model	
6.1.	Introduction	77
6.2.	Simulation Results for Hypothetical Network	77

6.2.1. Estimated Trip Time Savings	84
6.2.2. Estimated Average Travel Speed	89
6.3. Discussion	92
CHAPTER 7	Application of Traffic Simulation Model to Montreal Road Network
7.1. Introduction	93
7.2. Selected Road Network in Montréal	93
7.3. Results of the Simulation	94
7.3.1. Trip Time Savings	95
7.3.2. Minimum Trip Time Path Comparisons	98
7.4. Discussion	105
CHAPTER 8	Conclusions
8.1. Introduction	107
8.2. Conclusions	107
8.3. Application to Intelligent Transportation System (ITS)	108
8.4. Future Research	109
REFERENCE	
APPENDIX A: Network Generation Data File	
APPENDIX B: Printout of Computer Program	
APPENDIX C: Full Size Montreal Map Showing Selected Road Network	

LIST OF FIGURES

Figure No.	Title of the Figure	Page No.
3.1	Coded Transportation Network	25
3.2	Modified Coded Network	26
3.3	Priority Queue Data Structure	27
4.1	Coded Hypothetical Network	39
4.2	Fundamental Diagram of Traffic Flow	41
4.3	A Generic Cell Structure	42
4.4	Traffic Simulator Flow Chart	46
4.5	Four Leg Intersection with Incoming and Outgoing Links	47
4.6	Link Connectivity Flow Chart	49
4.7	Event Scheduler Flow Chart	51
4.8	Traffic Light Scheduler Flow Chart	52
4.9	TrafficGenerator Flow Chart	53
4.10	A Typical Link Segment	54
4.11	Time Division for Vehicle Transfer	57
4.12	Flow Chart for Left Turn Movement	58
5.1	UML Diagram of Cell Object Class	64
5.2	UML Diagram for Link Object Class	65
5.3	UML Diagram for Intersection Object Class	67
5.4	UML Diagram for Car Object Class	69
5.5	UML Diagram of TrafficLight Object	70
5.6	UML Diagram of TrafficGenerator Object Class	71
5.7	UML Diagram of MacroModel Object Class	72
5.8	UML Diagram of Path Object Class	73
5.9	UML Diagram of RoadCanvas Object Class	74
5.10	UML Diagram of RouteEngine Object Class	75
6.1	Gradual Variation of Trip Time for mixed Traffic	80-82
6.2	Variation of Trip Time for Guided and Unguided Vehicles	83

6.3	Trip Time Savings (%) variation for guided vehicles	87
6.4	Trip Time Savings (%) variation for Unguided vehicles	88
6.5	Average Travel Speed Variation for Guided Vehicles	90
6.6	Average Travel Speed Variation for Unguided Vehicles	91
7.1	Coded Montreal City Network	94
7.2	Variation of Trip Time savings according to Level of RG(%)	97

LIST OF TABLES

Table No.	Title of the Table	Page No.
4.1	Saturation Flow Rate	44
4.2	Origin-Destination Trip Matrix	44
4.3	Node Neighbour Information	48
4.4	Node Outgoing and Incoming Link Information	48
4.5	Link Connectivity Table	49
5.1	UML Diagram Definitions	62
6.1	Traffic Scenario Vehicle Composition	78
6.2	Simulation Results for Travel Time/km	79
6.3	Simulation Results for Trip Time	84
6.4(a)	Trip Times Savings for Guided vehicle	85
6.4(b)	Trip Times Savings (%) for Guided vehicle	85
6.5(a)	Trip Times Savings for Unguided vehicle	86
6.5(b)	Trip Times Savings(%) for Unguided vehicle	86
6.6	Simulation Results for Average Travel Speed	89
7.1	Intersection ID for Montreal Road Network	93
7.2	Origin Destination for Selected Path	95
7.3	Simulation Results for Trip Time	95
7.4(a)	Simulation Results for Trip Time Savings	96
7.4(b)	Simulation Results for Trip Time Savings (%)	96
7.5	Origin Destination for Selected Path	98
7.6	Trip Information for Unguided Vehicle for Path P1	99
7.7	Trip Information for guided Vehicle for Path P1	100
7.8	Trip Information for Unguided Vehicle for Path P2	102
7.9	Trip Information for Guided Vehicle for Path P2	102
7.10	Trip Information for Unguided Vehicle for Path P3	103
7.11	Trip Information for Guided Vehicle for Path P3	104

LIST OF SYMBOLS

Symbol	Description
ITS	Intelligent Transportation System
G	Basic Network
N	Total Number of Nodes In The Network
A	Total Number of Arcs In The Network
S	Source Node of The Network
D	Destination Node of The Network
i, j, k	Any Node In The Network
P_i	Permanent Label of Travel Time on Shortest Path of any Node i
T_i	Temporary Label of Travel Time on Shortest Path of any Node i
t, m_t	Discrete Time Step
$d_{ij}(t)$	Time-Dependent Link Delays or Travel Time
f_j	Minimum Travel Time From Origin Node to Any Node J
$w_{ij}(t)$	Maximum Waiting Time Allowed at Any Node i Departing At t
$\pi_{ij}(t)$	The Fastest Travel Time To Destination Node D Departing Node i at time t
$\lambda_i(t)$	The Total Travel Time on the Shortest Path from Node i to Destination Node at time t .
A_s	Total Number of Links on Shortest Path
δ	Small Time Interval when Some Predictable Change in Traffic Condition Occurs.
T	Sampling Period
Λ_i	A Vector Label that Stores all the $\lambda_i(t)$ Values for All Time Step.
SE	Scan Eligible
b	Branching Factor
d	Depth of the Shortest Path
FIFO	First-in-First-out
LIFO	Last-in-First-out
TAZ	Traffic Analysis Zone
O-D	Origin-Destination

ID	Identification Number
$d(i)$	Upper Level of the Travel Time on Shortest Path of any Node i
$pred(i)$	Predecessor Node of any Node i on Shortest Path
l_{ij}	Length of the Link between Node i and Node j
c_{ij}	Travel Time between Node i and Node j
$m_{ik}(j)$	Turning Delay Between Node i and Node k Through Node j
x_i, y_i	Coordinate of Node i
$\lambda(i)$	Perceived travel time from origin to destination node through node i
$g(i)$	Travel Time Experienced form Origin Node to Node i
$h(i)$	Estimated Travel Time from Node i to Destination Node
L_{ij}	Link Between Node i and Node j
Q	Traffic Flow Rate
u_f	Free Flow Speed
m	meter
km	Kilometer
H	Hour
sec	Seconds
Veh	Vehicle
q_{max}	Maximum Link Flow or Link Capacity
c	Link Segment
N_c	Total Number of Link Segment
Δ_c	Length of Link Segment
M	Total number of Time Steps
r	Red Time is Traffic Signal Cycle
C	Traffic Signal Cycle Duration
T_i	Trip time on shortest path
T_l	Link travel time
$k_c(t)$	Cell Density at any Time Step t
$u_c(t)$	Cell Mean Speed at any Time Step t
$q_c(t)$	Cell Traffic Flow at any Time Step t
k_j	Jam Density

$n_c(t)$	Number of Vehicles in Cell c at Time t
$N_c(t)$	Maximum Number of Vehicles in Cell c at Time t
$O_c(t)$	Number of Vehicles Leaving Cell c at Time t
$I_c(t)$	Number of Vehicles Entering into Cell c at Time t
$m_c(t)$	Number of Vehicles can Cross Segment c
UML	Unified Modeling Language
BBHPSA	Bi-objective Bidirectional Heuristic Path Search Algorithm
RG	Route Guidance
TMC	Traffic Management Center

CHAPTER 1

INTRODUCTION

1.1. Background

Almost all networks have characteristics which vary with time. Networks like transportation networks experience predictable rising and falling trends of utilization over the course of a day, typically at morning and evening “peak hour”. This is particularly true for highway networks but in urban road networks the traffic conditions are constantly changing. One of the primary problems in a dynamic transportation network is the determination of shortest paths. Because the state of the network is constantly changing, the shortest path between a given origin and destination also changes with time. This information is used not only for pre-trip route planning (based on the assumption that the traveler is seeking the shortest path), but increasingly today in a real-time context of route-guidance while en-route to the destination. With the scope of Intelligent Transportation Systems (ITS) along with improved logistics and hardware capabilities an efficient system should respond to continuously changing network conditions and thus model the driver’s path selection for the desired destination. The real challenge of an efficient route guidance system is to respond to the driver’s actual travel preference and at the same time to predict the route with a realistic estimation of total travel time required to reach to the destination. To make a reasonable estimation of trip time several factors that affect trip time into the network might need to be considered. Past studies indicate that minimizing turn in the travel path is an important choice for drivers because intersection turning maneuver induces the maneuvering complexity added to the trip and delays at intersection increases the trip time as well.

1.2. Problem Definition

For a road network, travel time on a particular link depends on the link dynamics such as traffic flow, individual vehicle speed etc. and delays at intersection depend on the intersection traffic signal timing cycles. Depending on the approach volume, priority is set to the different turning volume. Due to the dynamic characteristics of the traffic, often the vehicles experience unusual delay at the intersection which affects their trip time and that is why shortest paths based on link travel time do not always resemble the best trip time path as well as most preferred path of drivers. So, dynamic path information has to be evaluated a repeated number of times with changing traffic condition along with drivers route choice preference to determine the best trip time path between an origin-destination pair. Existing algorithms that calculate the dynamic shortest path do not implement these requirements at the same time whereas the combination of some well established path search techniques can be an efficient solution for this problem domain. This research study is intended to develop a new technique for dynamic shortest path calculation by combining the heuristic search with bi-directional search technique and considering intersection turning delays with link travel time.

1.3. Objectives and Structure of the Thesis

The objectives of this study are manifold. They are as follows:

- Develop a bi-objective bi directional heuristic dynamic shortest path algorithm.
- Develop a macroscopic traffic simulator using cell transmission model.
- Model the presence of guided vehicles following the developed path search technique on the road network and demonstrate its capability to model the corresponding traffic dynamics.

- Simulate different traffic scenarios to evaluate the improvement of using the new shortest path algorithm in terms of trip time savings.
- Demonstrate the practical application of dynamic shortest path algorithm and simulation model using Montreal road network.

The rest of the thesis is structured as follows,

Chapter 2 discusses the shortest path approach in transportation network and various routing algorithm for static and dynamic shortest path with their implementation issues.

Chapter 3 presents the formulation of the new algorithm for dynamic shortest path calculation with the necessary assumptions. It also presents the node search strategy and details implementation of the algorithm.

Chapter 4 discusses the details explanation of macroscopic traffic simulation model developed in this thesis with traffic flow dynamics.

Chapter 5 discusses the computer programs used in this thesis with the details object oriented implementation of the traffic simulator. UML diagrams of objects used in the simulator are also presented.

Chapter 6 discusses the results of the traffic simulator under various traffic scenarios for hypothetical network.

Chapter 7 demonstrates the application of new algorithm and traffic simulation model to Montreal City road network.

Chapter 8 summarizes the major conclusions of this thesis with future research directions.

CHAPTER 2

LITERATURE REVIEW

2.1. Dynamic Route Guidance

Factors such as time of day, street geometry, drivers' experience, drivers' behavior and people's daily activities affect vehicle distribution on a street network. Congestion occurs when people rush into the street at the same time and share the same routes particularly at bottleneck locations. By providing trip makers with information about travel options and so allowing them to make better travel decisions, dynamic route guidance system promise to enhance the utilization of existing network infrastructure and to help manage congestion. Drivers with real time traffic information will be able to avoid congested roads and therefore will help to improve traffic flow throughout the entire road network. In addition, by coordinating the paths of the various vehicles, the route guidance system can inform road users about traffic network conditions (link travel time, waiting time at intersections etc.) and suggest a path to follow from their current position to their ultimate destination that will minimize travel time for all drivers, not just those using the system.

2.1.1. Past and Present Approach to Routing

As the mobility of goods and peoples has been increased in the last few decades, traffic routing approach becomes a complicated issue to be dealt with. The major challenge for today's traffic network is to guide the vehicle to the destination with reduced trip travel time. In the past, travelers had to rely only on the expected normal network condition when planning for a particular trip. The recent development of dynamic traffic routing [7, 27, 31] are oriented to achieve both the 'Local' optimum and the 'Global' optimum

solution of this problem. The ‘Local’ optimum solution that diverts the traffic around the point of congestion works well for normal traffic condition like off peak hour but during the peak hours when traffic volume is very high the congestion might be shifted to downstream [42]. The ‘Global’ optimum solution employs both static and dynamic shortest path algorithm that helps drivers to take the anticipatory action regarding the route selection decision based on the detection of upstream congestion and optimal congestion control policies over both time and space [42].

2.1.2. Comparison of Computer Networks and Transportation Networks

Transportation networks are the fundamental structures of transport geography associated with the movement and storage of goods and people. Through the generalization, the spatial distribution of a transportation network can be compared with a computer network where the vehicular flow of transportation networks closely exhibit the behavior of “packets” in computer networks [5, 42]. As the extent of the computer networks increasing rapidly and recent developments in ITS [27], reflect a propensity for increased use of sophisticated routing algorithms to optimize the mass movement of internet and vehicular traffic. Different factors affecting the complexity of the particular routing algorithms for a computer network can be pointed as coordination between all nodes of the network, possibility of link and node failures and congestion [5]. It is interesting to note that the possibility of link failures and congestion is very similar to the incidents and traffic jams in transportation networks.

2.1.3. Deterministic and Adaptive Route Guidance Architecture

Extensive research [14, 23, 29, 43] has been done in the last two decades in the study of network routing architectures. Two kinds of routing architectures have been studied

particularly: the deterministic routing architecture and the adaptive routing architecture. A deterministic routing architecture uses a fixed path for each pair of source and destination nodes, while an adaptive routing architecture may use network information and select an alternative path for a given pair of source and destination nodes. It has been observed that the deterministic routing architectures, though simple and efficient, are extremely vulnerable to the dynamic network condition. For this reason, adaptive routing architectures have been preferred and extensively studied in the literature [14, 23]. Adaptive routing schemes can be further classified into *centralized routing* and *distributed or decentralized routing* [14, 29, 43]. In the centralized adaptive routing architecture, route selection function is performed at some central site like traffic management center. In the distributed routing architecture, route selection function is located in-vehicle or on-passenger in the delivery of multi-modal route guidance. Decentralized or distributed scheme can be further classified into autonomous and dynamic architecture. In the autonomous form of decentralized architecture, all route guidance functionality is in-vehicle. Here route guidance is relegated to primarily static route guidance applications, since there is no mechanism to systematically update the network map database with current link travel times. In the decentralized dynamic architecture, link-travel times are broadcast to vehicles to provide real-time estimates of network congestion. The management center does not directly track route requests or location/destination information from its clients (vehicles in the network). In this study, decentralized dynamic architecture has been considered to develop the new shortest path approach.

2.2. Shortest Path Approaches in Transportation Networks

Network flows is a problem domain that arise in several fields including applied mathematics, computer science, engineering, management, operation research. Shortest path problems are among the most studied network flow optimization problems [2, 40]. The most obvious applications arise in transportation or communications, such as finding the minimum travel time route between two nodes or between origin and destination. Due to the nature of applications the path finding procedures in transportation network need to be very flexible and efficient both from running time point of view and memory requirements. Since no particular algorithm can solve all kinds of transportation network path search problems, extensive research can be found for different implementation of shortest path finding procedures [2, 20, 40].

2.2.1. Basic Assumptions for Shortest Path Problems

To solve shortest path problems for a transportation network several assumptions have to be imposed [2]. Considering a basic network $G=(N, A)$ having $|N|$ nodes and $|A|$ arcs where each arc is $(i, j) \in A$ associated with an arc length (or arc cost) c_{ij} . The network has two distinguished set of nodes, node S called Source and node D called Destination. The length of the directed path is defined as sum of the lengths of arcs in the path. The shortest path problem is to determine for every non source node $i \in N$, a shortest length directed path from node S to node D . The necessary assumptions are given below:

- All arc lengths are integers.
- The network contains a directed path from node S to every other node in the network.
- The network does not contain a negative cycle.

- The network is directed.

2.2.2. Static Shortest Path Algorithms

Network where link costs remain constant over time is treated as static network. The most fundamental form of static shortest path algorithm found in the literature is Dijkstra's [19] algorithm that finds shortest path from the source node to all other nodes in a network. Most of the researches [11, 12, 13, 20, 28, 35, 36, 47] on shortest path of the network are based on this algorithm. The algorithm follows the forward star node arrangements on its way to find the shortest path for a given source node. The algorithm has been modified in various researches from its initial formulation to improve the run time complexity using different data structure [2, 11].

2.2.3. Dynamic Shortest Path (DSP) Algorithms

So far, discussions about the shortest path algorithms mostly dealt with the static shortest path problems. Since the main concern is the dynamic transportation network where link cost or specifically travel time changes along with time, the focus is on dynamic shortest path finding techniques. In transportation application, especially in the urban network where congestion is a huge concern, dynamic network condition must be considered to calculate the shortest path. The dynamic shortest path problems are categorized into two types [17]. In the first type, the static shortest paths must be recomputed due to frequent, instantaneous and unpredictable changes in the network. The second type is the time-dependent shortest path problem in which network characteristics change in a predictable fashion. Such problems arise frequently in vehicular transportation, where shortest paths are computed from anticipated future characteristics of the network for different departure time as one travels through morning or evening "Peak Hour". In the subsequent section different dynamic shortest path algorithms that have been found in the literature

are presented for the network given earlier. The network is said to be dynamic if the value of network data, such as arc travel times or arc travel costs, depend upon the time at which travel along those arcs takes place.

2.2.3.1. Programming Formulation for the DSP Algorithms

Dynamic path algorithms have been extensively investigated in the literature. Some of the major works done can be found in [11, 12, 13, 20, 35, 36, 47]. The label setting algorithm was proposed [20]. This generalizes the Dijkstra's [19] algorithm for the shortest path from s to D described earlier. It is to be noted that this algorithm is based upon an implicit assumption of first-in-first-out (FIFO) property also stated by [28, 35, 36]. [42] has formulated the algorithm as given below:

Let P_i be the permanent label and T_i be the temporary label of the node i . P_i represents the shortest travel time from origin node to i and T_i represents the upper bound of the shortest travel time from origin to same node. The procedure is as follows:

Step 1: $i = s$ start from the origin node s

Set $P_i = 0$ and $P_j = \infty \forall j$ for all temporary nodes

Step 2: $T_j = \min\{T_j, P_i + d_{ij}(P_i)\}$

Step 3: Find $\min T_j \forall j$ and set $P_j = T_j$

Step 4: if $j = D$ then stop, make the shortest path tree rooted at i otherwise, $i = j$

Step 5: go to step 2

The algorithm converges at most $N - 1$ steps.

Several models considered the shortest path problems where link delay varies with time [35, 36]. They investigated the minimum delay path finding algorithms under various waiting constraints. Given a bi-directional network $G(N, A, D)$ with $N = \{1, 2, 3, \dots, n\}$ being

the set of nodes, $A \subseteq N \times N$ the set of links (with $(i, k) \in A$) implying $(k, i) \in A$), and $D = \{d_{ik}(t) \forall (i, k) \in A\}$ a set of time dependent link delays. $d_{ik}(t)$ is a strictly positive function for $[0, \infty]$ time bound which describes the delay of flow over link (i, k) at time t . Links are referred to as *frozen link* model considering the link delay is fixed at a time when a vehicle starts traversing the link referred to as the *elastic link* model when link delay varies with time. Because there is a possibility of non-FIFO characteristics in the elastic link model so, both the FIFO and non-FIFO conditions have been considered in this research.

Three different network traversal policies are considered.

- *Unrestricted waiting* (UW) in which unlimited waiting is allowed everywhere along the travel path through the network.
- *Forbidden waiting* (FW) in which waiting is disallowed everywhere along the travel path through the network
- *Source waiting* (SW) in which waiting is disallowed everywhere along the travel path through the network except at the source node which permits unlimited waiting.

The dynamic shortest path problems were analyzed in [11, 12, 13] according to the following category:

- Fastest versus minimum cost path problems.
- Discrete versus continuous representation of time.
- First-in-first-out (FIFO) networks versus non-FIFO networks.
- Waiting is allowed versus waiting is not allowed at nodes.
- Expected types of shortest path: One-to-all for a given departure time or all departure times and all-to-one for all departure times.

- Integer versus real valued link travel times.

In these studies, the author investigated one-to-all version of the shortest path problem by using alternative arguments and also proposed an efficient all-to-one discrete dynamic shortest paths algorithm. They are summarized below:

Given a discrete dynamic network $G(N, A, D, C)$ where, $N = \{1, \dots, n\}$ is the set of nodes and $A = \{(i, j) \in N \times N\}$ is the set of links. The number of links is denoted by m . $D = \{d_{ij}(t) \forall (i, j) \in A\}$ is the set of time-dependent travel times and $d_{ij}(t)$ is a discrete time-dependent function which takes a static value after a finite number of intervals M . $S = \{0, \dots, M-1\}$ is the set of departure time intervals, $O(i)$ is the set of nodes having an outgoing link to node i and $I(i)$ is the set of nodes having an incoming link from node i .

Formulation of the optimum functions are given below:

- One-to-all fastest path problem

Waiting at nodes is not allowed

If f_j denotes the minimum travel time from origin node S to node j leaving the origin node at time 0. Then,

$$f_j = \begin{cases} \min_{i \in O(j)} \min_{t \geq f_i} (t + d_{ij}(t)), & j \neq S \\ 0 & ; j = S \end{cases}$$

For node j only one path is considered that visits the previous node i at a time greater than equal to f_i .

- All-to-one for all departure times fastest path problem

A backward star formulation when no waiting is allowed was presented when, $\pi_i(t)$ is the fastest travel time to destination D departing node i at time t . So, the minimum travel times can be formulated in the following functional form

$$\pi_i(t) = \begin{cases} \min_{j \in A(i)} d_{ij}(t) + \pi_j(t + d_{ij}(t)) & i \neq q \\ 0 & i = q \end{cases}$$

The author has given a proposition that $\pi_i(t)$ can be set in a decreasing order of departure time intervals.

The general time-dependent shortest path algorithm proposed in [47] is based on Bellman's [4] principle of optimality, which had also been followed in other studies as discussed earlier. Considering the directed network given earlier, $\lambda_i(t)$ denotes the total travel time of the current shortest path from current node i to node N at time t . A vector label $\Lambda_i = [\lambda_i(t_0), \lambda_i(t_0 + \delta), \dots, \lambda_i(t_0 + M\delta)]$ is considered to store all the labels $\lambda_i(t)$ for every time step from the set $S_M = \{t_0, t_0 + \delta, t_0 + 2\delta, t_0 + 3\delta, \dots, t_0 + M\delta\}$. Instead of scanning all the nodes in each iteration, a list of scan eligible (SE) nodes is maintained, containing the nodes with some potential to improve the node labels of at least one other node. The algorithm is summarized below

Step 1:

- Create SE list and initialize it with destination node N .
- Initialize the vector labels $\Lambda_N = (0, 0, \dots, 0)$, $\Lambda_i = (\infty, \infty, \dots, \infty)$ for $i = 1, 2, 3, \dots, N-1$

Step 2:

- If $SE = \emptyset$, go to step 4

Otherwise, Select first node from SE list [current node i], delete it from the list.

Scan each node $j \in \Gamma^{-1}\{i\}$ for each time step $t \in S$

$$\lambda_j(t) = \min\{\lambda_j(t), d_{ij}(t) + \lambda_i[t + d_{ij}(t)]\}$$

If at least one of the labels of node j has been improved, insert node j into SE list.

Step 3:

- Repeat step 2.

Step 4:

- Terminate the algorithm.

On termination of the algorithm, every element of the vector label is either an infinite number, means no valid path exists between this node and destination node at the corresponding time step, or a finite number that represents the shortest path for the same.

2.2.4. Path Search Techniques

Algorithm run time depends mainly on how nodes of the network are expanded for shortest path determination. Some of the major techniques [2, 24, 34, 39] are discussed below:

2.2.4.1. Labeling Techniques

Labeling algorithms are most popular and efficient algorithms for solving the shortest path problems. The network flow studies [2] typically classify shortest path algorithms into two groups. These are label-Setting and label-Correcting algorithms. Both the algorithms are iterative. They assign tentative minimum cost labels to nodes at each step of the algorithm. Both of the algorithms proceed in a way such that these labels are improved until the shortest path is found. But they vary in a way they update the cost labels from step to step and how they converge toward the shortest path for the node pair. The two problem approaches also differ in the way they implement the data structure for execution of the algorithm. Label-Setting algorithms are much more efficient and have much better worst case complexity bounds; on the other hand label-Correcting algorithms

are applicable to more general class of problems. In fact, label-setting algorithms are special cases of label-correcting algorithms.

2.2.4.2. Heuristic Search

The running time complexity and finding the optimal solution are two fundamental goals that every algorithm has to satisfy and there is a significant trade-off between these two. So, sometimes it is suitable to choose heuristic algorithm that gives up one or both of the goals. The heuristic search is an informed search strategy that uses the globally available information to reduce the search scope and at the same finds a near optimum solution. The most popular heuristic search used in path finding has been reported in the literature [1, 24, 46] is A* algorithm which incorporates the available network information in directing the search effort towards destination. To ensure that the above function will yield an optimum solution, one must set the lower bound of the estimating function. To apply A* algorithm for finding shortest travel time path in the network, an admissible estimating function is the Euclidian distance between two nodes divided by the maximum speed (free flow speed of the link). This has been reported in previous studies [1, 24]. The heuristic search was furthered by looking ahead one extra node towards destination and defined as “Best Neighbor heuristic search” [46]. Sparser networks that exhibited less grid like topologies benefited more from this modified version of heuristic technique.

2.2.4.3. Bi-directional Search

The basic procedure of the bidirectional search is to compute the objective path from both forward and backward search direction at the same time [18, 34, 39]. Running time of this technique is much faster than the unidirectional search and two conditions must be satisfied for the successful implementation of this search [46]. They are

1. A criterion for stopping the search action.
2. A criterion for altering between the forward and backward searches.

The search technique works by altering the forward and reverse version of Dijkstra's [19] algorithm. During initialization algorithm associate zero cost with Origin s and Destination t for forward and reverse search option respectively. In addition the algorithm maintains the value of the shortest path λ and initialize it with $\lambda = \infty$. When the head node i and tail node j of link (i, j) is scanned by the forward and reverse search then λ is updated with $d_s(i) + l_{ij} + d_r(t)$ if λ has the higher value earlier. Similar update is done for reverse search also. The algorithm terminates when the node chosen by the forward search is already scanned by the reverse search. The worst case complexity of the algorithm is $O(b^{d/2})$ where b is the branching factor and d is the depth of the shortest path [46].

2.2.5. Multi-objective Shortest Paths

Due to the multiobjective nature of dynamic transportation problems, the multiobjective shortest paths considering various constraints have been studied extensively in previous studies [3, 6, 26, 32, 33]. The multimodal urban transportation system has been modeled for multiobjective path planning problems [33]. They have proposed a utility measure to take into account the propensity of the trip planning satisfying multiple objectives of minimizing total travel time, travel cost and discommodity. Various route selecting strategies link has been given [6] (1) most direct path (2) path that minimizes use of highway (3) path that avoids local street and (4) path that avoids highway. In the bi-objective path search model, they have proposed a path finding algorithm which gives the path with best trip quality considering two attributes when selecting a path. Travel time is

taken as primary search attribute and trip complexity is taken as secondary attribute to break the ties in the objective function. They have considered the trip complexity as turning movement complexity cost based on the angle between the links involved in the turning maneuver and defined the solution path that produces good travel time and good trip complexity. Trade-offs between the two objectives (travel time and trip complexity) have been demonstrated to show route choice decision sensitivity to the different level of demand volume.

2.2.6. A Comparative Study of Popular Shortest Path Algorithms

From the literature of the dynamic shortest path discussed so far, several conclusions can be made:

- Most of the algorithms consider only the link travel time as for the objective function to get the optimal solution.
- Most of the algorithms differ only through the complexity analysis and data structure implementation.
- The bi-objective algorithm developed by [6] considers a fixed complexity cost for turning movement which doesn't mimic the real world scenario.
- The combination of heuristic search and bidirectional search would be an interesting approach as they both reduce the total run time of the algorithm.

As research [22] on driver behavior has indicated that travel time is not the only factor that dominates the route selection process, the path search algorithms need to address the multiple objectives that can influence the driver route choice. Also, to avoid any information bottleneck between the vehicle in the network and the central traffic Management Center while calculating the shortest path using the real time traffic

information, the shortest path algorithm need to be very fast. Considering all these aspects and the limitations of the existing dynamic path search algorithms, this study is motivated to propose a dynamic path search technique which combines the heuristic search with bidirectional search technique as both minimize the algorithm run time from the basic label-setting algorithm as found in the literature. The search algorithm considers the intersection turning delay as bi-objective variant which contributes a major part of total trip travel time in an urban network to mimic the multi-objective behavior the dynamic network.

2.3. Implementation Issues for Shortest Path Algorithm

Dealing shortest paths in a transportation networks carry various implementation issues that can affect both shortest path algorithm performance and analysis of the network itself. Given the increasing size of present digital road network database, complexity of the dynamic network problem seemingly small differences in implementation can make huge differences in practice. The most critical shortest path implementations issues have been found in the literature [2, 45] are:

- Network Representation
- Node selection rule/ node processing structure
- First-in-First-out (FIFO) network

Network storage structure refers to the physical data model for the transportation network. This includes the relative memory locations that allow efficient retrieval of related information. The labeling method keeps track of which nodes are candidates for being scanned. The decision rule/node processing structure determines which candidate

node should be scanned next. Each of this issue is very crucial to the implementation of the particular shortest path algorithm and will be discussed in the following sections.

2.3.1. Network Representation

Network representation or network storage structure refers to physical data model for the transportation network. The transportation network has been defined as a system of linear features connected at intersections and interchanges [45]. These intersections and interchanges are called nodes. The linear feature connecting any given pair of nodes is called an arc or link. Examples of real world networks include road networks, telecommunication networks, and river networks, among others. How a network is represented in a computer is vital to the performance of a particular network analysis or traffic simulation. The trade-off of choosing a particular data structure is often between speed and storage space. In network analysis, commonly used representations of a network include [2, 24]

- Node-Arc Incidence Matrix
- Node-Node Adjacency Matrix
- Adjacency Lists
- Forward and Reverse Star Representation

Previous studies [2, 15, 45] have demonstrated that the Forward and Reverse Star Representation is the most efficient among all existing network data structures for representing a network. In the study, forward star representation has been followed to build the network topology

2.3.2. Different Data Structure

Node selection function is a bottleneck operation for all shortest path algorithms [2]. Efficient node selection can significantly improve the worse case complexity of an algorithm. Different data structures have been found in the literature [2, 10, 45, 47] those were implemented in various algorithms.

Link List: Rather than store the elements in a sequential order, it can store element in an arbitrary fashion. All it needs is the additional information to get access that element in the specified order of the structure. A cell is the building block of linked list. A linked list can be viewed as a collection of cells. Each cell stores the information about the cell itself and also a handle to the next cell in the linked list. When a cell stores handle only for the next cell then it is called “Singly Linked List”; on the other hand if it stores handle for both the next and previous cell then it is called “Doubly Linked List”. In transportation network model a travel path can be seen as a linked list of path segments, where each segment acts like a cell and stores the segment traffic information and also the information about the next segment to follow in the particular path. Depending on the implementation issues, travel path can be treated as singly or doubly linked list.

Queues: This storage structure holds an implicit property, while dealing with its stored elements. It handles elements in a first-in-first-out (FIFO) manner with elements inserted at one end (the rear/tail) and deleted from the other end (the front/head). A road segment where overtaking is not allowed resembles the queue structure in a way that a vehicle leaving an intersection enters into the segment at the rear end and leaves the segment from its front end. There is another form of queue structure where insertion can be done at both ends but deletion take place only from the front end. This special type has been

extensively implemented into the execution of shortest path algorithm and traffic simulation [10, 47].

Stacks: This is a special kind of storage structure, which exhibits the last-in-first-out (LIFO) property where all insertions and deletions take place in one end. A transportation network does not truly resemble this structure.

2.3.3. First-in-First-out(FIFO) Network Properties

Link travel time functions sometimes behave such that vehicles exit the link in the order in which they entered. This property is referred to as the FIFO (first in first out) condition [12, 30, 42]. The FIFO condition, also known as the non-overtaking condition in traffic theory [12], may be written mathematically in a variety of ways. One definition is that the FIFO condition is valid if and only if:

$$\forall(i, j, t), t + d_{ij}(t) \leq (t + 1) + d_{ij}(t + 1)$$

From the above relationship it can be interpreted that, as long as FIFO condition is satisfied link exit time function is non-decreasing.

2.4. Discussion

In this chapter, a brief overview of the shortest path approaches for static and dynamic transportation network along with different path search techniques has given. The related works regarding dynamic shortest path calculation and their limitations have been highlighted. The chapter is concluded by providing the various implementation issues in shortest path algorithm regarding network representation and data structure.

CHAPTER 3

DYNAMIC SHORTEST PATH SEARCH ALGORITHM

3.1. Introduction

The theoretical concepts that form the basis for the development of the new minimum trip time path algorithm are explained in this chapter. The chapter discusses the node search strategy and search attributes for the algorithm. The step wise implementation and algorithm pseudo code are also presented.

3.2. Theoretical Concept

In transportation planning process, urban areas are subdivided into smaller sectors of traffic analysis zone (TAZ) [37]. These zones are connected to each other through roads and intersections that constitute the transportation network. Fixed facilities in a transportation network are coded in terms of nodes, links and attributes for individual links. Nodes represent the intersection of multiple road segments while links represent sections of roads between intersections. Link attributes such travel times are based on the segment length and speed of the traffic stream on the particular road segment. Traffic volume on the road network depends on the interzonal trip which can be defined as movement of a passenger or a vehicle between an origin and destination (O-D) zone. A minimum path is a path with the minimum amount of specific impedance between O-D pair. The impedance can be travel time, travel distance, number of turns etc. A minimum distance path between any O-D pair is a path that has a minimum distance between the origin and destination. The minimum trip time path developed in this thesis is the path that needs least time to go from origin to destination cumulating all link travel times and intersection delays on the path.

3.3. Minimum Trip Time Path Algorithm

The static and dynamic shortest path algorithms discussed in chapter 2 that use the link travel as a search attribute implements the basic Dijkstra's [19] algorithm in the following form:

The algorithm maintains a label $d(i)$ associated with each node i that represents the upper bound of the travel time on the shortest path from source node to that node. The algorithm divides the network into two groups as temporary labeled and permanently labeled for node selection option and it maintains a tree for predecessor indices $pred(i)$ for every node using node selection function $d(j) = d(i) + c_{ij}$, where c_{ij} is the travel time of link between node i and node j . Each time a node i is expanded to the adjacent node j the label of node j is updated with the selection function. If the label is found greater than the $d(j)$ already recorded for node then the label is kept unchanged, otherwise label $d(j)$ is updated to reflect the reduction in travel time to reach to the node from the origin node and the predecessor indices $pred(j)$ of node j is updated to node i . The process is continued until all nodes have been expanded. Upon termination, the predecessor index for each node is used to get the shortest path tree rooted at the origin node. The new algorithm follows the same approach to build the path but considering the intersection turning delays along with the link travel time. The node search strategy and details implementation of the new algorithm are given in the subsequent section.

3.3.1. Formulation of the Algorithm

Assuming a finite directed network $G = (N, A)$ having $|N|$ nodes and $|A|$ arcs where each arc $(i, j) \in A$ is associated with an arc length l_{ij} . The network has two distinguished nodes, node O is called origin and node D is called destination. The length of the directed path is

defined as sum of the lengths of individual arcs in the path. Let $d_{ij}(t)$ be the non-negative time required to travel from node i to node j departing at time t and $m_{ik}(j)$ be the turning delay associated between node i and node k through node j . It is important to note that maneuver or turning delay for a particular node is dependent on the orientation between the predecessor and successor of the node towards the direction of path. So, for all the boundary nodes of the network the turning delay is considered as zero. $I(i)$ is defined as the total number of incoming links to node i and $O(i)$ is defined as total number of outgoing links from node i . The time horizon is defined as M .

3.3.2. Assumptions

The following basic assumptions are made for the algorithm.

1. The basic network is sparse grid network.
2. All the nodes resemble signalized intersections.
3. Turning or maneuvering complexity depends on the signal light delay rather than the angle between links involved in the turning
4. Travel cost is taken as real valued link travel time and intersection turning delay.
5. The algorithm addresses one-to-one minimum trip time path problem departing an origin at a given time interval.

Assumption 1 can be justified as in the real world transportation networks each node is connected with only few nodes. Assumption 2, 3, and 4 can be justified for the use case of an urban street network.

3.3.3. Search Strategy

As the computational time is the most important factor for shortest path algorithm, following two most important considerations must be taken care of

1. Search area for next node on the shortest path.
2. The data structure used to retrieve the previous node on the shortest path.

The search area for the next node on the shortest path depends primarily on the objectives of the models and the type of algorithm usages. For a decentralized traffic management systems where on board route guidance system is expected to have the capability to calculate the fastest path for the driver. In this thesis only one-to-one shortest path algorithm is considered where algorithm calculates the path between origin and destination only. In a labeling algorithm, the number of visited nodes during the search is a good measure of the optimal search area [2]. So the less number of visited or scanned nodes indicate higher efficiency of the algorithm in terms of processing time. The number of visited nodes depends entirely on depth d in the shortest path between two nodes and branching factor b . For “Best First Search” link Dijkstra’s [19] algorithm the number of nodes during search is of order $O(b^d)$ [38]. So for a vast transportation network databases this exponential growth of depth creates the computational hazard. This bottleneck can be lessened by heuristic search technique while reducing the basic network into a simplified one.

To optimize the search area for the algorithm, the geographical orientation of the origin and destination nodes is used. This is similar to the normal approach to the navigation problems or flow of liquid on a slope surface. Depending on the orientation of the nodes the out degree of each node is reduced in a way that no nodes will have any outgoing link directly opposite to both vertical and horizontal direction from origin to destination. For a typical $N * N$ fully connected grid network with maximum out degree of 4 the total number of directed links can be given by $4(N^2 - N)$ [1]. So, the total number of links can

be reduced to $2(N^2 - N)$. The optimization technique can be best visualized from the following example network.

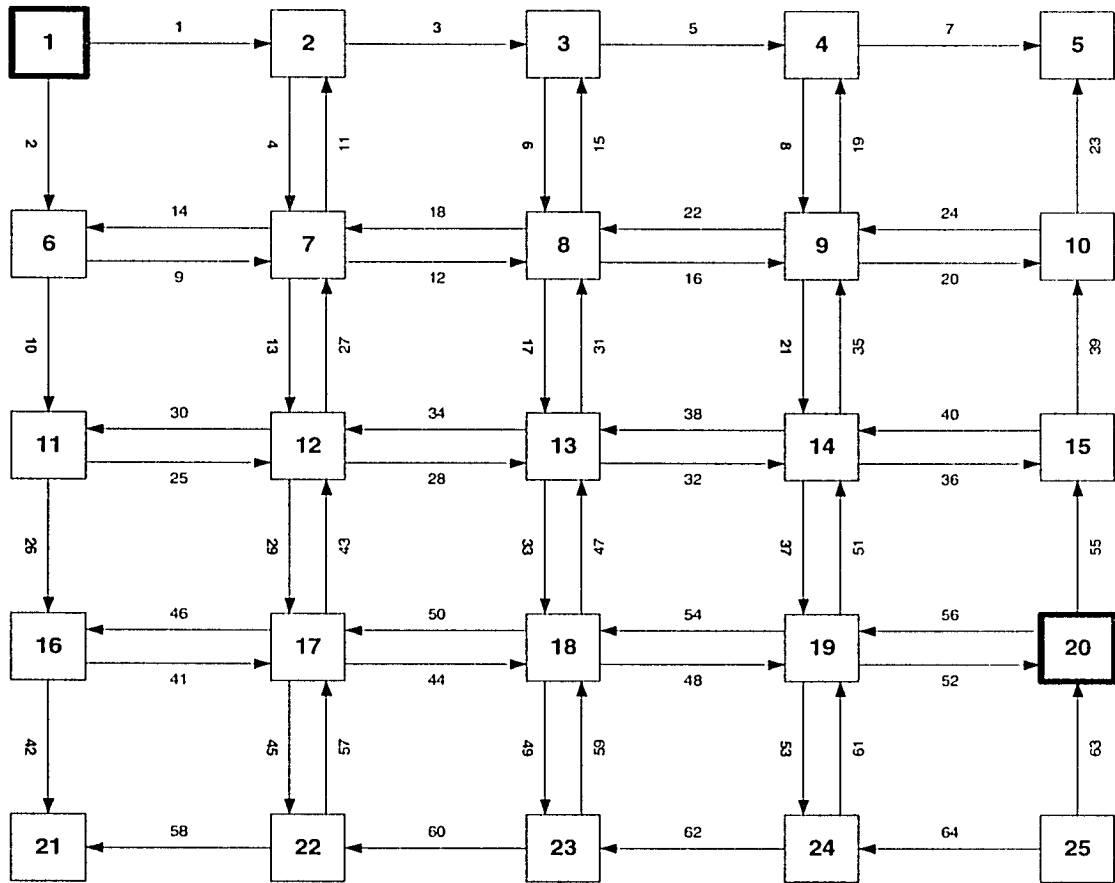


Figure 3.1: Coded Transportation Network

In the above coded transportation network, the total number of nodes are 25 and total number of links are 64. So, average out degree or the branching factor of this network is 2.56. Nodes 1 and 20 are considered as origin and destination respectively. As the destination node is on the south\eastern side of the origin node the base network is modified with all the nodes having outgoing links towards south and east direction only. The modified network is shown below,

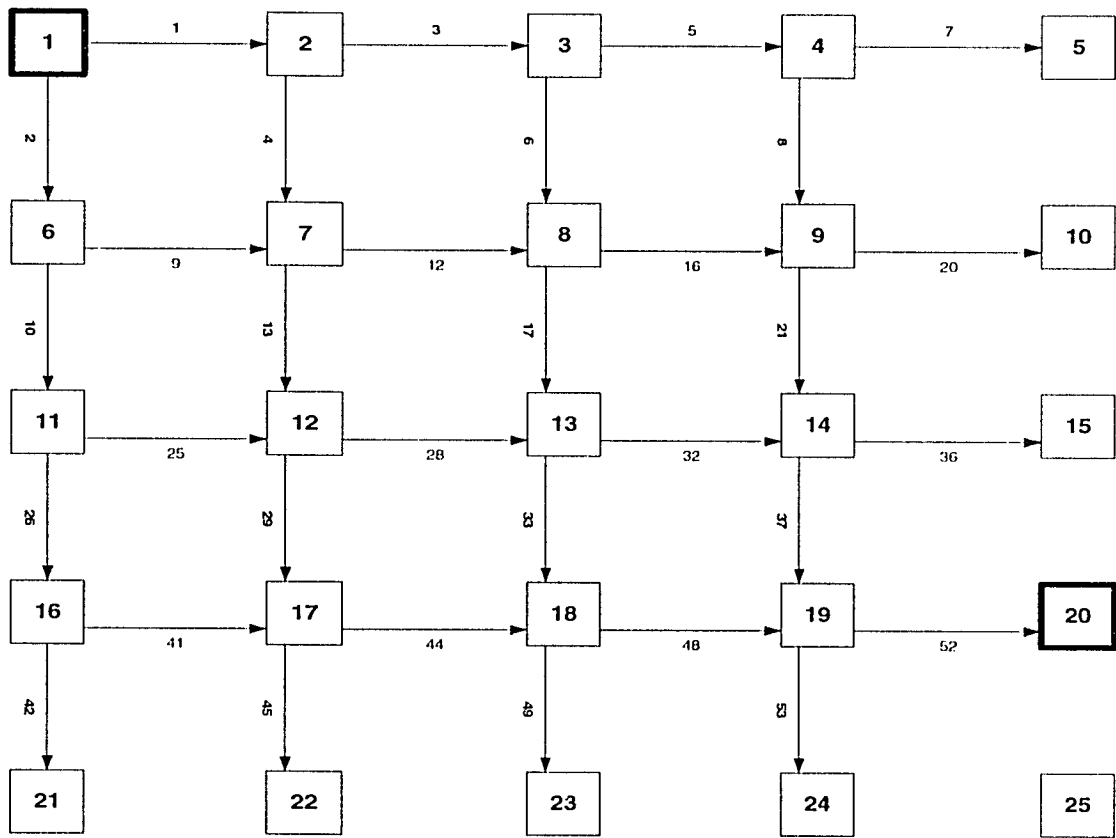


Figure 3.2: Modified Coded Network

In this network, the total number of links are 32. So, the reduction of total number links is 50%. Another important point to note that there will always be a possibility to have some nodes with no incoming links. That means those nodes will not be considered during algorithm run time process.

The priority data structure has been implemented in this study for node selection and path selection strategy during algorithm run time process which is more sophisticated form of the queue data structure discussed in chapter 2. In priority queue data structure elements are selected depending on priority of the particular element, not in the order of their entry into the structure.

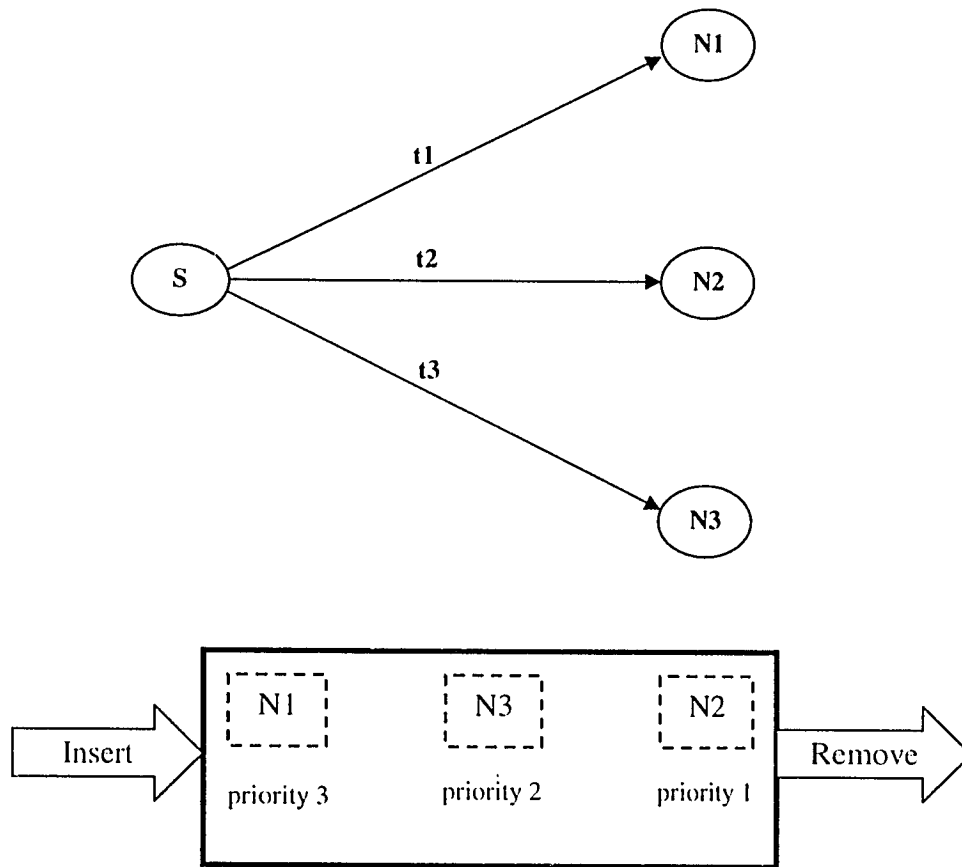


Figure 3.3: Priority Queue Data Structure

From a given node **S** in the above figure, t_1 , t_2 and t_3 are assumed as the required travel times to reach the three neighbor nodes **N1**, **N2** and **N3** and $t_1 > t_3 > t_2$. To retrieve the node with minimum travel time from node **S**, the node **N1**, **N2** and **N3** are stored in a priority queue by giving priority according to travel time. So for this case, upon removal the first element of the priority queue **N2** will be selected as the node requires least time to reach.

As urban street network is a grid like network which is also an assumption for the algorithm the estimation function for the heuristic search has been used in this study as the Euclidian distance between two nodes divided by the free flow speed of the network

as reported in the literature [1, 46]. So, the estimation function between two nodes i and j can be written as:

$$e(i, j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad (3.1)$$

$$\therefore h(i) = \frac{e(i, j)}{u_f} \quad (3.2)$$

where,

x_i, y_i = coordinate of node i

x_j, y_j = coordinate of node j

u_f = free flow speed

This estimation function ensures the lower bound of travel time thus satisfy the admissibility property [1, 46]. So, the node selection function for any given node i can be written as

$$\lambda(i) = g(i) + h(i) \quad (3.3)$$

where,

$\lambda(i)$ = perceived travel time from origin to destination node through node i

$g(i)$ = Travel time experienced from origin node to node i

$h(i)$ = Estimated travel time from node i destination node

The symmetrical bidirectional search combined with heuristic search has been followed in this study. It has been implemented by altering the forward and backward search. Each time a forward search scans an arc (i, j) such that node j has already been scanned by the backward search, it checks if the path O-D between origin and destination is formed by the path O- i from the forward search and path j -D from the backward search has the less

travel time level than the travel time level of path O-D so far, then the path O-D is updated. Same update is done for the backward search if the above condition is satisfied. For the case of optimum solution domain heuristic and bidirectional search technique are not good choices. Bidirectional search needs to explore almost twice as many nodes as a unidirectional search [46]. For a decentralized real time traffic management system when the vehicles look ahead for best path, each time it reaches any intersection, a near optimum solution would be sufficient to reach the destination. So, if the optimality constraint is relaxed a near-optimal route can be found much faster by using the bidirectional heuristic search.

3.3.4. Description of The Algorithm

To calculate minimum “Trip Time” path for any given origin destination nodes the algorithm is implemented through two separate steps. In the first step, a bidirectional heuristic search algorithm calculates a list of paths with minimum link travel time between the given nodes. In the second step, a bidirectional search algorithm calculates a list of paths with minimum turning complexity (intersection turning maneuvering delay) between the same pair of nodes. Then for all the paths total trip time is calculated using simulator flow variable for that time interval and stored in a priority queue data structure. The first path of the priority queue gives the least “Trip Time” path between the particular origin destination pair for that time interval. It is important to note that heuristic search has been only used for least travel time path calculation because it has been assumed earlier that turning complexity is associated with traffic signal timing plans of the intersection rather than the angle between the links. So, the estimation function to turning complexity heuristic algorithm is not applicable.

To calculate the link travel time, the link capacity function as given in [8] has been used in the study. The function is given below,

$$T_l = \frac{l_{ij}}{u_f} \left[1 + 0.15 \left(\frac{q}{q_{\max}} \right)^4 \right] \quad (3.4)$$

where,

T_l = Travel time required to cross the link L_{ij}

l_{ij} = Length of link L_{ij}

u_f = free flow speed

q = Link flow

q_{\max} = Maximum Link flow or capacity

Though this function ignores the interdependencies between conflicting flow streams, the use of this simplified link performance is justifiable as a complicated function could initiate the computational burden to the algorithm.

In making turning maneuver as an objective intersection turning delay has been chosen which is very much similar to the maneuver complexity concept that had been introduced in [6]. The turning delays on urban street network is not associated with the angle between turn but rather controlled by the signal lights at intersection. This turning delay can be extended from static scenario to real time scenario while considering the time dependent turning delays which play an important role in trip planning and obviously depends not only on geometry and control policies of the intersection but also on the flow of the opposing links. The delay function used in this study for the signalized intersection is based on the Webster's model given in [44]. The equation gives the average delay at the intersection. The general Webster's model [44] is given below:

$$d(i) = \frac{r^2}{2C \left(1 - \frac{x}{q}\right)} \quad (3.5)$$

where,

x = Traffic volume on entering link, veh/hr

q = Exiting rate of traffic volume, veh/hr

r = Red time of the traffic light, in minutes

C = Cycle time of the traffic light, in minutes

$d(i)$ = Average delay at the intersection, in minutes

i = Intersection

The equation 3.5 gives the same amount of delay for all the turning movements corresponding to each link but in reality left turning delay is much higher than the straight and right turning delays. In the thesis, this scenario is implemented by reducing the capacity of left turning lane to 1/3 of the straight turning lane capacity. So obviously the paths with less number of left turns are less complex and have less amount of delay. On the other hand straight paths might face more congestion which result in increase in travel time. So drivers may need to sacrifice link travel time if they do not want to wait in the intersection and vice versa. The total travel time from origin to destination is 'Trip Time' which combines the link travel time and intersection turning delay to give the measure of the trip in terms of time scale. Mathematically it can be written as:

$$T_t = \sum_{l=1}^{A_t} T_l + \sum_{l=1}^{A_t} d(l.Head) \quad (3.6)$$

where,

T_t = Trip time on shortest path

T_l = Travel time on link L_{ij} .

$d(l.Head)$ =Turning delays at the intersection between link L_{ij} and next link on shortest path.

A_s = Total number of Links on shortest path.

As already mentioned earlier, the algorithm calculates the fastest 'Trip Time' path in two separate steps and on termination chooses the best path in terms of trip time from a priority queue which is eventually a combination of choice sets. One of the choice sets is denoted for time based paths where paths are generated by bidirectional heuristic search and stored in an ascending order and the other one is denoted for the complexity based paths where paths are generated by bidirectional search and also stored in an ascending order. One way is to develop the choice set by determining k-paths between the origin and destination node. But for a real world implementation it would be unrealistic to determine all the possible paths between nodes because it takes larger computational effort with literally no effect for the higher ranked path in the decision making process. The number of path would be calculated depends upon the initial best path as it is expected that the next best path will be somewhere close to the previous best one. The k-path is calculated by removing each link of the initial best path one by one from the network. It is important to note that once the k^{th} path is found after removing a link, that link has to be restored immediately before going to remove the next link in order to keep network integrity. So, once the two choice sets get all the k-paths then total trip time is calculated for all the paths and are added to a priority queue in an ascending order. The first path of the priority queue gives the fastest path in terms of trip time.

3.3.5. Implementation of the Algorithm

The following data structures have been used to implement the algorithm.

$OPEN_f$: Priority queue for unsettled or unvisited nodes with least cost [time or complexity] from origin node used for forward search. This set does not allow duplication of node entry. As the set is sorted so, it acts like a priority queue to determine which node to choose for the next step.

$OPEN_b$: Priority queue for unsettled or unvisited nodes with least cost [time or complexity] from destination node used for backward search. This set does not allow duplication of node entry. As the set is sorted so, it acts like a priority queue to determine which node to choose for the next step.

$CLOSED_f$: Queue used in the forward search for settled or visited nodes whose least cost [time or complexity] path is already found from the origin node.

$CLOSED_b$: The Queue used in the backward search for settled or visited nodes whose least cost [time or complexity] path is already found from the destination node.

$Predecessors_f$: Stores the predecessor node for the current node on the least cost [time or complexity] path during forward search.

$Predecessors_b$: Stores the predecessor node for the current node on the least cost [time or complexity] path during backward search.

$Best$: Stores the node where the algorithm terminates.

$Path_Bank_Time$: Priority queue for Least travel time path.

$Path_Bank_Complexity$: Priority queue for Least complex path

$Path_Evaluate$: Priority queue used during algorithm run time for K-path determination.

Road_Bank: Priority queue for trip time path. Priority queue contains the paths in an ascending order from above mentioned path vector by comparing the trip time.

The algorithm can be summarized by the following steps:

Step 1: Optimize network according to the orientation of the origin and destination nodes.

Step 2: Run BHSPA to compute the least travel time path between origin and destination.

Step 3: Run BHSPA by removing each link from least time path and store the path in an ascending order.

Step 4: Run the BSPA to calculate the least complexity path between origin and destination.

Step 5: Run BSPA by removing each link from least path and store the new path in an ascending order.

Step 6: For all the paths in the Travel time choice set calculate the complexity and add it with travel time to get the total trip time.

Step 7: For all the paths in the Complexity choice set calculate the travel time and add it with complexity to get the total trip time.

Step 8: Compare all the paths in the two choice set and order them in ascending order according to trip time.

The network storage structure will be discussed in details in the following chapter. Here are the details of the algorithm according to the steps pointed earlier:

Bi-objective Bidirectional Heuristic Path Search Algorithm [BBHPSA]:

Path_Bank_Time.Add = Run BHSPA(Origin, Destination)

1. Path_Evaluate .ADD = Path_Bank_Time.FirstElement()

2. For all the Links in Path_Evaluate Link vector, do
3. Remove the i^{th} link from the path and from the network.
4. Path_Bank_Time.Add = Run BHSPA(Origin, Destination)
5. Restore_Network()
6. Path_Bank_Complexity.Add = Run BSPA(Origin, Destination)
7. Path_Evaluate .ADD = Path_Bank_Complexity.FirstElement()
8. For all the nodes in the Path_Evaluate Node vector, do
9. Remove the i^{th} link from the path and from the network.
10. Path_Bank_Complexity = Run BSPA(Origin, Destination)
11. Restore the network
12. For all the paths in Path_Bank_Time Path vector, do
13. Calculate Complexity for i^{th} path
14. Determine 'Trip Time' for i^{th} path by adding $\text{Time}_i + \text{Complexity}_i$
15. For all the paths in Path_Bank_Complexity Path vector, do
16. Calculate Travel Time for i^{th} path.
17. Determine 'Trip Time' for i^{th} path by adding $\text{Complexity}_i + \text{Time}_i$
18. For all the paths in the two path vector,do
19. Road_Bank.Add = i^{th} path
20. Best Path = Road_Bank.FirstElement

Bidirectional heuristic algorithm for time based path (**BHSPA**) is the following:

- **Initialization:** Forward Search: Place origin node O as $\text{OPEN}_r.\text{First} = O$ and $\text{CLOSED}_r = \varnothing$; set $g(O)=0$, $\lambda(O)=g(O)+h(O)$, set $\lambda(n)=\infty$ for $n \in N - O$

Forward Search: Place destination node D as $OPEN_b.First = D$ and $CLOSED_b = \varnothing$; set

$g(D)=0$, $\lambda(D)=g(D)+h(D)$, set $\lambda(n)=\infty$ for $n \in N - D$

▪ **Iteration:**

forwardLoop :

while($OPEN_f \neq \varnothing$)

begin

Let $i = OPEN_f.FirstNode$

$CLOSED_f = CLOSED_f \cup i$

$OPEN_f = OPEN_f \cap i$

for each $j \in B(i)$ *do*

if $\lambda(j) > g(i) + d_{ij}(t) + h(j)$ *then* $\lambda(j) = g(i) + d_{ij}(t) + h(j)$ *and* $predecessor_f(j) = i$;

if ($i \in CLOSED_b$) *Best* = i ; *break : forwardLoop;*

backwardLoop :

while($OPEN_b \neq \varnothing$)

begin

Let $i = OPEN_b.FirstNode$

$CLOSED_b = CLOSED_b \cup i$

$OPEN_b = OPEN_b \cap i$

for each $j \in A(i)$

if $\lambda(j) > g(i) + d_{ij}(t) + h(j)$ *then* $\lambda(j) = g(i) + d_{ij}(t) + h(j)$ *and* $predecessor_b(j) = i$;

if ($i \in CLOSED_f$) *break : forwardLoop;*

break : backwardLoop;

end;

end;

▪ **Path Construction:** For node i get path to origin node O from forward search through $predecessor_f(i)$ and get path to destination node D from backward search through $predecessor_b(i)$. The resulting path is solution path.

Bidirectional algorithm for complexity based path (BSPA) is the following:

▪ **Initialization:** Forward Search: Place origin node O as $OPEN_f.First = O$ and $CLOSED_f = \varnothing$; set $m(O)=0$, set $m(n)=\infty$ for $n \in N - O$

Backward Search: Place destination node D as $OPEN_b.First = D$ and $CLOSED_b = \varnothing$; set

$m(D)=0$, set $m(n)=\infty$ for $n \in N - D$

▪ **Iteration:**

forwardLoop:

while($OPEN_f \neq \emptyset$)

begin

Let $i = OPEN_f.FirstNode$

$CLOSED_f = CLOSED_f \cup i$

$OPEN_f = OPEN_f \cap i$

for each $j \in B(i)$ *do*

if ($predecessor_f(i) = Null$) $m(i) = 0$;

if $d(j) > d(i) + m(i)$ *then* $d(j) = d(i) + m(i)$ *and* $predecessor_f(j) = i$;

else

$k = predecessor_f(i)$

if $d(j) > d(i) + m_{kj}(i)$ *then* $d(j) = d(i) + m_{kj}(i)$ *and* $predecessor_f(j) = i$;

end if

if ($i \in CLOSED_b$) $Best = i$; *break* : *forwardLoop*;

backwardLoop:

while($OPEN_b \neq \emptyset$)

begin

Let $i = OPEN_b.FirstNode$

$CLOSED_b = CLOSED_b \cup i$

$OPEN_b = OPEN_b \cap i$

for each $j \in A(i)$

if ($predecessor_f(i) = Null$) $m(i) = 0$;

if $d(j) > d(i) + m(i)$ *then* $d(j) = d(i) + m(i)$ *and* $predecessor_f(j) = i$;

else

$k = predecessor_f(i)$

if $d(j) > d(i) + m_{kj}(i)$ *then* $d(j) = d(i) + m_{kj}(i)$ *and* $predecessor_f(j) = i$;

end if

if ($i \in CLOSED_f$) *break* : *forwardLoop*;

break : *backwardLoop*;

end;

end;

- **Path Construction:** For node i get path to origin node O from forward search through $predecessor_j(i)$ and get path to destination node D from backward search through $predecessor_i(i)$. The resulting path is solution path.

3.4. Discussion

From the implementation of the algorithm it is evident that the total time required to reach to the destination calculated by the proposed algorithm is always more, compared to the other algorithms for the shortest path calculation found in the literature based on the link travel time only and also the minimum path found by those algorithms might be different depending on the network traffic condition during travel period. But for a same O-D pair total travel time on shortest path from other algorithms can still be compared to the trip time on the minimum path calculated by the proposed algorithm. Traffic simulation results on travel time comparisons are discussed in Chapters 6 and 7.

CHAPTER 4

TRAFFIC SIMULATION MODEL

4.1. Introduction

The traffic simulator developed in this thesis is based on the macroscopic simulation model. Fundamental diagram of traffic flow is assumed to model network traffic dynamics. The simulator is made up of three distinct phases and each phase deals with a different aspect of the simulation model.

4.2. Simulator Scenarios

General considerations of road network, macroscopic traffic flow dynamics for the simulator are given below.

4.2.1. Description of the Network

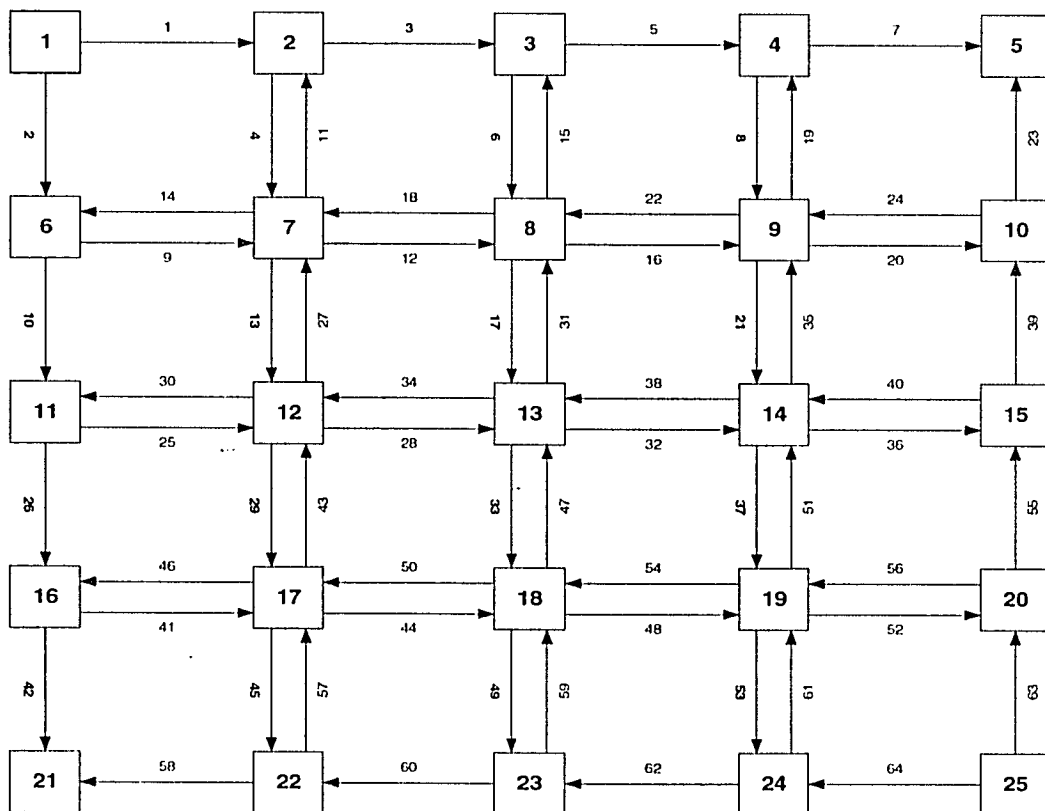


Figure: 4.1: Coded Hypothetical Network

The test network considered for this simulator is a small urban network of 25 nodes and 64 links. All the nodes are considered as four-leg intersection and all the links are directed towards either North-South or East-West direction. Each direction link is assigned with a unique identification (ID) even though they connect same nodes. In this manner the network coded for this simulator differs from the normal coded network where both one way and two way roads between the nodes are given the same road name instead of giving separate names for each direction. The length taken is 1000 meter for each link. The link cell length is calculated from the product of free flow of the traffic stream and simulation time step. Simulation time step is considered as 6 sec for this study. Free flow speed 60 km/h gives the distance traveled by a vehicle moving with free flow speed in a single time step is 100 meter. So, the cell length is taken as 100 meter means a vehicle can cross at most one single cell in one time step. So, for all 1 km links the total number of cells is 10.

4.2.2. Modeling Traffic Dynamics

The macroscopic traffic model used in this study captures the evolution of traffic over each link of the road network without any intermediate entrances or exits, so that those vehicles enter at one end and leave at the other end. This model is time-driven, in which current network traffic conditions are updated every time step of the simulation step.

4.2.2.1. Modeling Concept

The relationship between the density, speed and the corresponding flow of traffic on the roadway is assumed to follow the fundamental diagram of traffic flow [21]. The relationships between these flow variables are given in Figure 4.2.

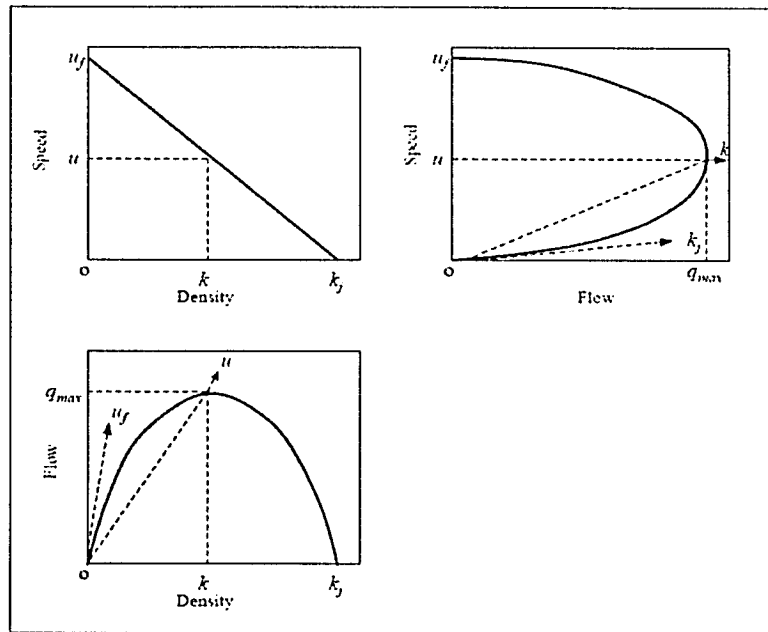


Figure 4.2: Fundamental diagram of traffic flow [21]

The following interpretation can be made from the fundamental diagram:

- When the density is zero, the flow of traffic is zero as there is no vehicle on the road
- The flow increases as the density increases.
- When the traffic density starts to increase, the flow also increases. The traffic flow reaches maximum at certain density, after that it decreases with further increase in density.
- The flow becomes zero at a maximum density generally referred as jam density.

Macroscopic traffic flow variables are considered as the following:

- Traffic density $k(x, t)$, number of vehicles per unit length [veh/km]
- Traffic volume $q(x, t)$, number of vehicles per unit of time [veh/h]
- Mean traffic speed $u(x, t)$, [km/h]

The linear relationship between the density and speed of the traffic satisfy the continuity equation for flow conservation [16] that defines the relationship between flow (q) and density (k) over time and space in the following form:

$$\frac{\partial k}{\partial t} + \frac{\partial q}{\partial x} = 0 \quad (4.1)$$

As macroscopic models are usually based on a discretization, in both space and time each of the cell object is assumed according to Figure 4.3.

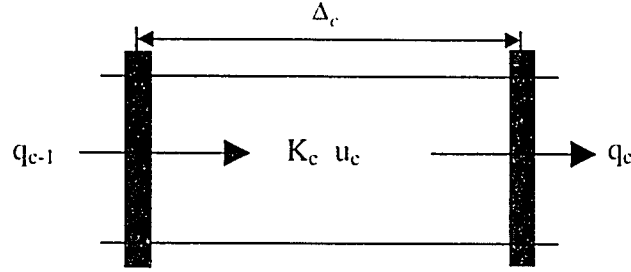


Figure 4.3: A generic Cell Structure

Where , Δ_c denotes the length of cell c and q_{c-1} and q_c denotes the inflow and outflow of the cell. If the road is assumed to have N_c number of cells so, the link length is given by,

$$l = \sum_{c=1}^{N_c} \Delta_c \quad (4.2)$$

So, for a sampling period T and for $m_t = 0, 1, 2, 3, \dots, M$ where $m_t =$ discrete time step and $M =$ total number of discrete time step in the simulation, the flow variables for the macroscopic can be defined as follows:

$k_c(t)$ = number of vehicles in cell c at time $t = m_t T$ divided by Δ_c

$u_c(t)$ = mean speed of vehicles in cell c at time $t = m_t T$

$q_c(t)$ = numbers of vehicles exiting cell c in the period $m_t T, (m_t + 1)T$, divided by Δ_c

Considering time discretization and for the homogenous traffic equation 4.1 can be written as

$$q_c(t) = u_c(t) * k_c(t) \quad (4.3)$$

The parabolic relationship between the speed and flow of traffic and also between the flow and density of the traffic is assumed according to Greenshild's[21] model that is also consistent with fundamental traffic flow diagram given in figure 4.2. The relationship can be given by the following equation:

$$u = u_f \left(1 - \frac{k}{k_j} \right) \quad (4.4)$$

where,

u_f = free flow speed (km/h)

k_j = jam density (veh/km)

4.2.2.2. Traffic Stream Characteristics

The traffic stream characteristics are assumed to follow the fundamental speed-flow relationship. The link capacity is 1800 vph and jam density is 150 vpkm and free flow speed is taken as 60 kmph. It is also assumed that during any time step only discrete number of vehicle can advance from one cell to another cell. The fraction part of the outflow number is carried over the next time step to make the number discrete. The discrete equivalent of capacity or maximum outflow from a cell is based on the roadway capacity 1800 vph is 3 vehicles for each time step. Considering jam density 150 vpkm the discrete number of maximum vehicle can be present in any cell is 15 vehicles during any time step. The minimum time headway is taken 2 seconds for all the vehicles. All cells in the network are considered to be homogenous with respect to assumed traffic stream

characteristics. The saturation flow rate considered for all the turning movements are given below:

Table: 4.1: Saturation Flow Rate

Turning Movement	Saturation Flow Rate (veh/h)
Straight Turn	1900
Left Turn	600
Right Turn	1200

4.2.3. Signal Timing Cycles

All the signals are considered as a two phase signal and to have the fixed signal timing plans. For simplicity no coordination between these are considered. The signal timing varies from 60 sec to 80 sec. No exclusive amber time is considered so it is included in the green phase of the signal. To avoid all the signal starts at the same time, signals are activated during the loading phase of the simulation with and random initial delay.

4.2.4. Travel Demand Characteristics

All the peripheral nodes of the network are assumed as traffic zone centroids and vehicles are assumed to get into the network through these nodes. The following trip table has been prepared with assumed traffic volumes between origins and destinations.

Table: 4.2: Origin-Destination Trip Matrix

Node	1	2	3	4	5	6	11	16	21	10	15	20	22	23	24	25	Production
1	0	0	0	0	0	40	0	0	0	20	50	40	20	35	60	0	265
2	0	0	0	0	0	20	30	20	30	20	30	40	50	15	20	0	275
3	0	0	0	0	0	20	30	15	50	20	20	50	40	20	60	0	325
4	0	0	0	0	0	20	30	40	25	10	20	20	30	40	20	0	255
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	10	20	10	20	0	0	0	0	20	25	30	35	20	20	0	210
11	0	15	25	35	45	0	0	0	0	25	20	25	15	30	40	0	275
16	0	35	30	50	60	0	0	0	0	25	25	20	10	40	35	0	330
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	45	25	10	0	15	25	40	25	0	0	0	30	25	30	0	270
15	0	40	20	10	0	10	25	45	40	0	0	0	55	20	15	0	280
20	0	20	70	30	0	40	25	10	60	0	0	0	25	15	10	0	305
22	0	10	15	45	30	25	20	10	0	20	40	30	0	0	0	0	245
23	0	30	10	35	25	45	25	20	0	20	40	15	0	0	0	0	265
24	0	60	50	20	25	55	45	20	0	35	30	10	0	0	0	0	350
25	0	40	30	60	0	20	40	30	0	0	0	0	0	0	0	0	220
Attraction	0	305	295	305	205	310	295	250	230	215	300	280	310	260	310	0	3870

The sample Origin-Destination trip matrix shown in Table 4.2 is used to generate traffic into the network and all the trips are one hour trips between origin and destination nodes. As found in Table 4.2, node 1 and node 25 the total attraction is equal to 0 denotes that these node have incoming links and for node 5 and node 21 the total production is equal to 0 denotes that these nodes have no outgoing links. The last row of the table represents total trip attraction to the node and the last column of the table represents total trip production from the node.

4.3. Simulation Structure

A macroscopic traffic simulator has been developed to model the traffic flow in networks for performance evaluation of guided vehicles following the proposed path search technique to demonstrate the possible travel time savings. This simulator is based on a typical time update simulation technique and treats vehicles on individual basis. The simulator includes a network generator module to build the network topology according to the predefined road network data structure. Having an existing network, simulation is started with a given initial network state and few number of randomly generated vehicles. Vehicle generation, vehicle movement are periodically updated according to the given travel demand, speed-density relationship and signal settings through an event scheduler. The simulator consists of three main phases: Loading Phase, Update Phase and Advance Phase. In loading phase, network data is loaded from a data file and all simulation variables are initialized. Update phase coordinates between events like traffic generation, intersection traffic lights and vehicle arrival into the network through the event scheduler. Advance phase deals with traffic movement between links depending on the link traffic condition. The interrelation between these phases is given on the flowchart, Figure 4.4.

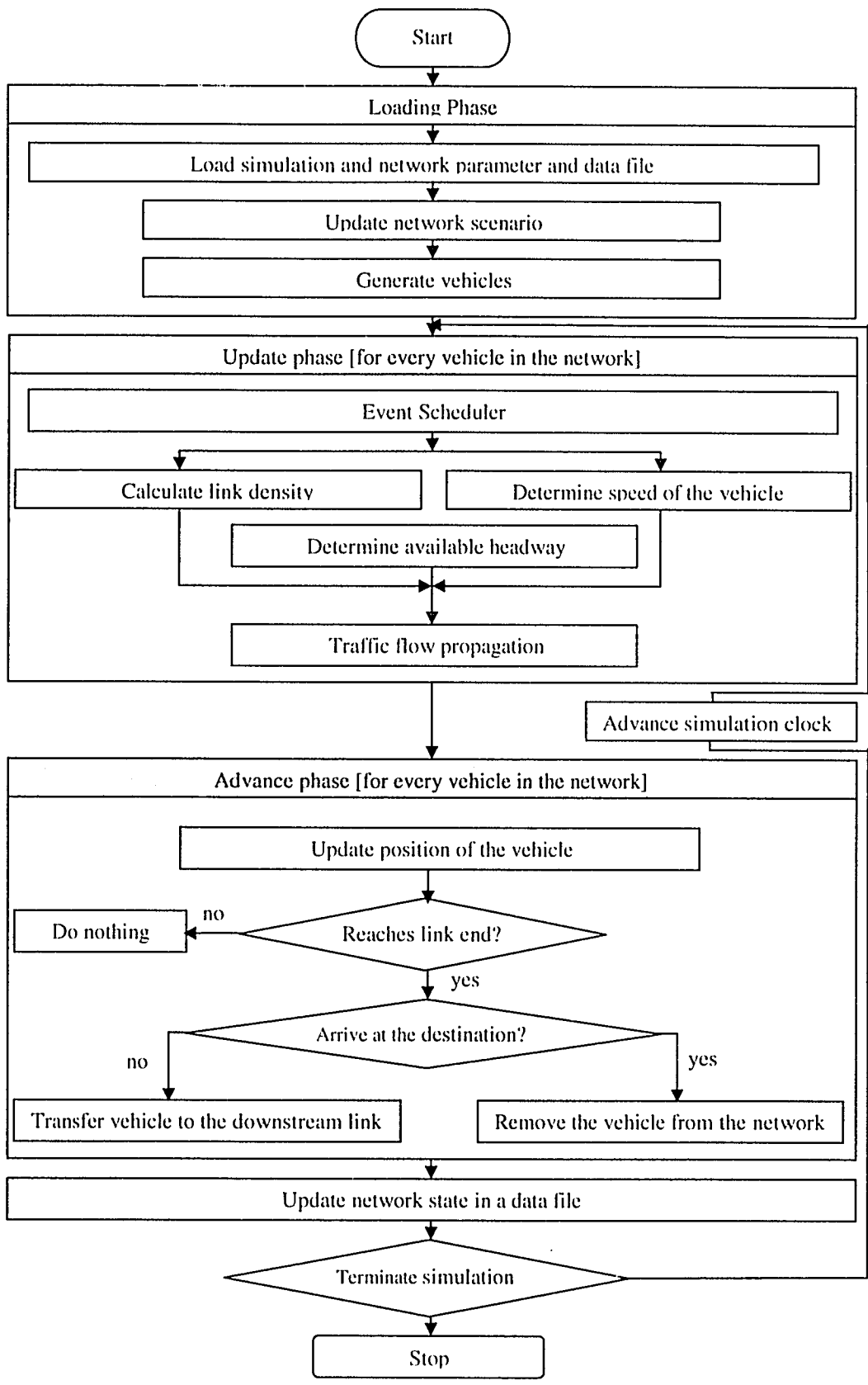


Figure 4.4: Traffic Simulator Flow Chart

In the simulator, each element of the network like node, link, vehicle, traffic light is treated as an individual object. Each of these objects has its own property to interact with each other during the simulation. The detailed implementations of these objects are discussed in the next chapter.

4.3.1. Loading Phase

The loading phase consists of two sub models: Network generator sub model, Input data sub model.

4.3.1.1. Network Generator Sub-Model

Network generator sub model is used to create the network topology on which simulator models the traffic flow. It provides the option to create a coded network and after building the network topology it stores the network data in a data file from where data can be restored for any later use. The following coding approach is followed to identify the relative orientation of all neighbor nodes relative to each node.

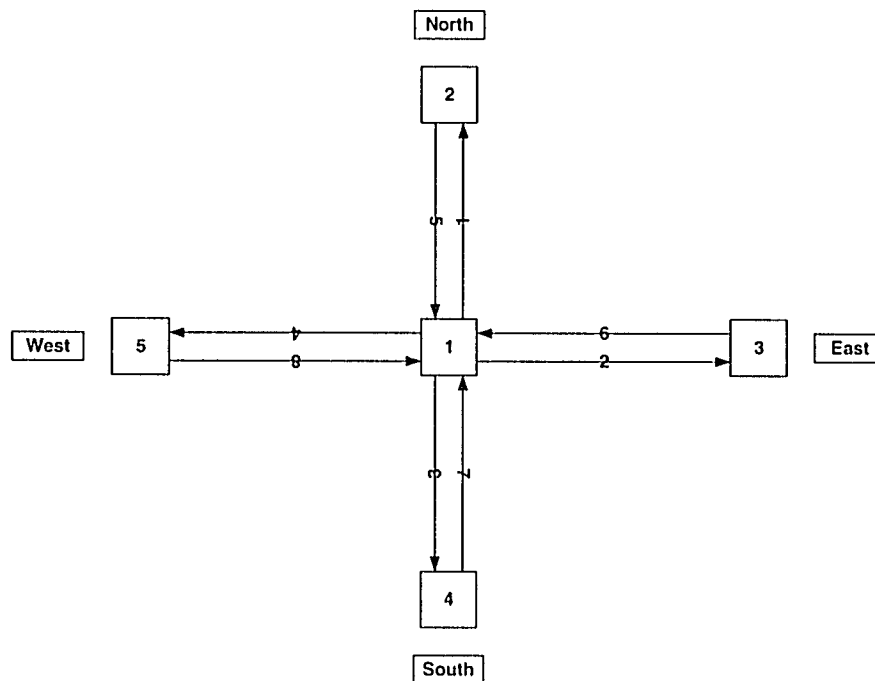


Figure: 4.5: Four Leg Intersection with Incoming and Outgoing Links

In Figure 4.5 a node object is surrounded by four node objects and connected with them by 8 incoming and outgoing link objects. The Node object stores all four surrounding node information regarding their position coordinates and ID, starting from north direction by moving clockwise towards west direction. The outgoing and incoming link information is also stored in the same fashion depending on their relative orientation with respect to base node. In the above figure all the numbers associated with the node and link object denotes their respective ID. So, for node 1 neighbor node information is given in Table 4.3 according to the coding approach.

Table: 4.3: Node Neighbor Information

Direction			
North	East	South	West
2	3	4	5

The outgoing and incoming link information for node 1 is given as follows:

Table: 4.4: Node Outgoing and Incoming Link Information

Link	Direction			
	North	East	South	West
Outgoing	1	2	3	4
Incoming	5	6	7	8

Each node object stores the total number of outgoing links as ‘Out Degree’ and total number of incoming links as ‘In Degree’. If there is no outgoing or incoming link for any direction ‘Null’ value is stored as shown in Table A.1 and Table A.2 in the appendix for the hypothetical network given in Figure 4.1. Once the node object gets all the surrounding node information then each of the link objects associated with the node object is updated for their surrounding link information. Each link object stores the starting node as ‘Head’ node and ending node as ‘Tail’ node. Depending on the orientation of link object with respect to its ‘Head’ node, link object stores the

surrounding link information. In Figure 4.5 for link 7, the 'Head' node is 1 and 'Tail' node is 4. So, with respect to 'Head' node 1, link 7 gets its left link as 4, straight link as 1 and right link as 2. The information is given in Table 4.5.

Table 4.5: Link Connectivity Table

Link	Node				
	Head	Tail	Left Turn	Straight Turn	Right Trun
7	1	4	4	1	2

Link Connectivity approach as discussed is presented in the following flow chart:

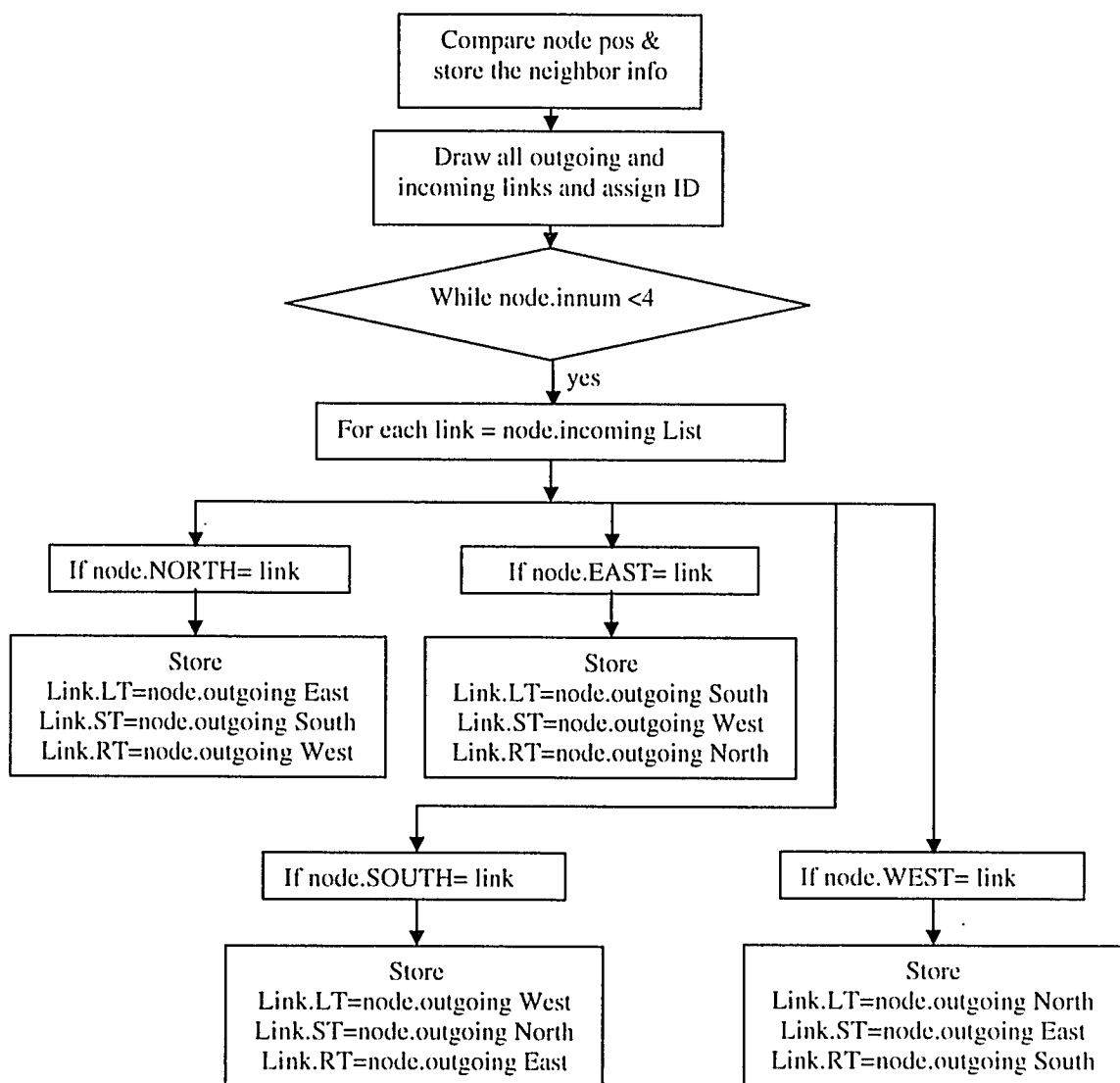


Figure 4.6: Link Connectivity Flow Chart

Once the link connectivity is established the connectivity information is stored in a data file for later use. The link-link connectivity data file for the hypothetical network given in Figure 4.1 is given in appendix A.3. For any direction, if there is no link object found then 'Null' value is stored which implies that no vehicle can take any turn for that particular direction. For link object 1 in the hypothetical network, this can be verified from Table A.3 in the appendix where 'Left Turn' is set to 'Null' as there is no left turn from the link. The link-node connectivity is developed from the link connectivity information and stored in a data file. The link-node connectivity data file for the hypothetical network is given in Table A.4 in the appendix.

4.3.1.2. Input Data Sub-Model

This sub model basically initializes all the variable with the user defined simulation data from the specified data file, initiates traffic generator object according to assumed origin-destination trip matrix to generate vehicle and initiates the simulation clock and traffic lights with random initial delay. The random vehicle objects which have no definite origin and destination are also inserted during the initialization of the network variable.

4.3.2. Update Phase

This phase consists of two important sub-modes, event scheduler sub-model and traffic flow propagation sub-model.

4.3.2.1. Event Scheduler Sub-Model

Event scheduler sub model responds to the events generated by all the running objects in the simulator. In the simulator three independent event generator objects have been used and they are TrafficLight ,TrafficGenerator and Simclock. Each of these objects has their own event generation scheduler. Whenever any event is due by any of these objects,

event scheduler gets the message from the objects and manages the event accordingly. Along with these scheduler objects the sub model also checks the vehicle queue of every link at regular intervals to check any vehicle in the queue waiting to get entry into the link. Also when each of the guided vehicles reaches the new link, they send the message to the scheduler to update the path information according to the current network condition and calculate the shortest path from the current node to the destination. The flow chart of the event scheduler sub model is given in Figure 4.7.

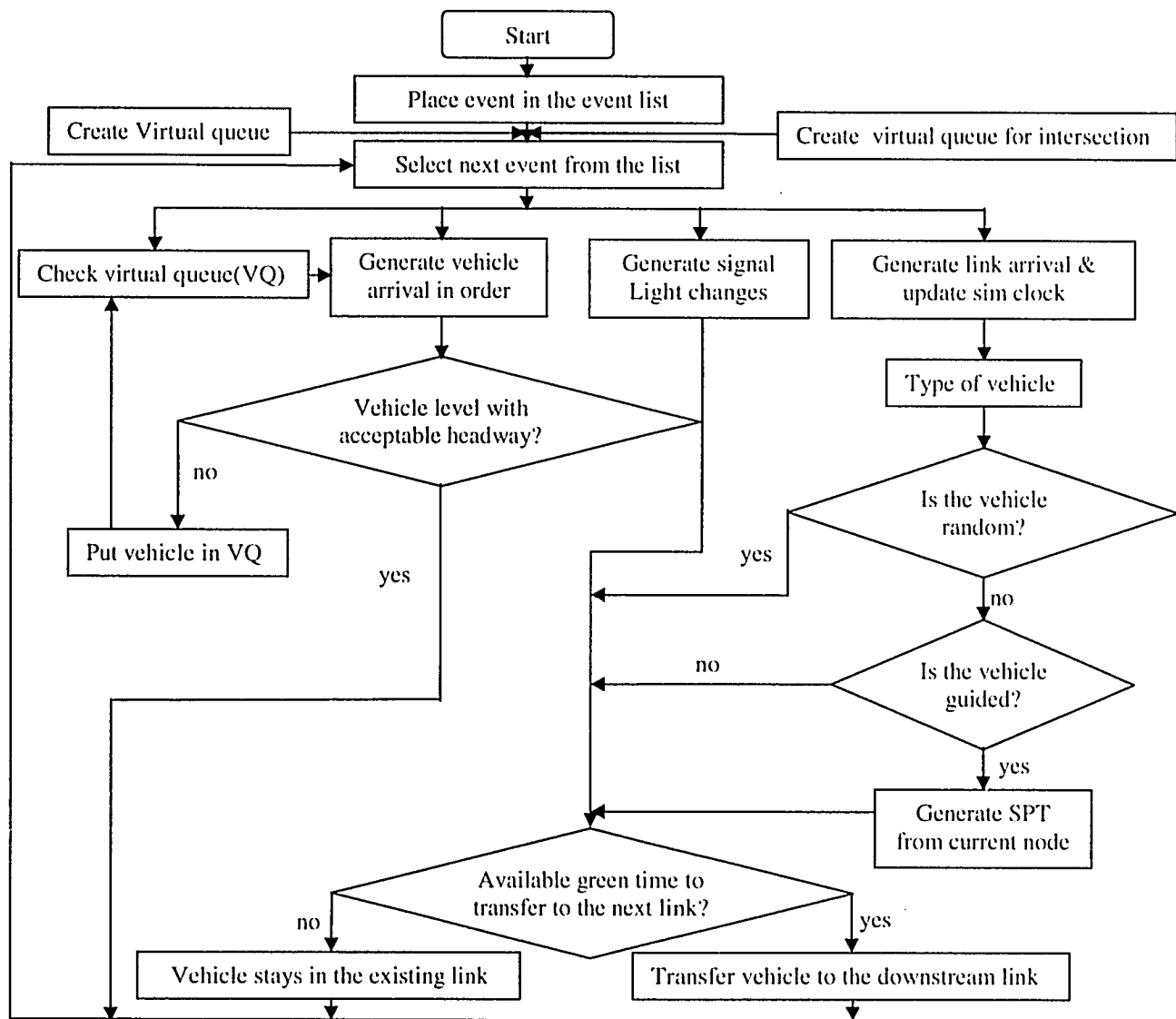


Figure 4.7: Event Scheduler Flow Chart

TrafficLight object switches between the green and red phase according to the preset signal cycle split and associated with each node object and run independently. The flow chart of the TrafficLight object is as follows:

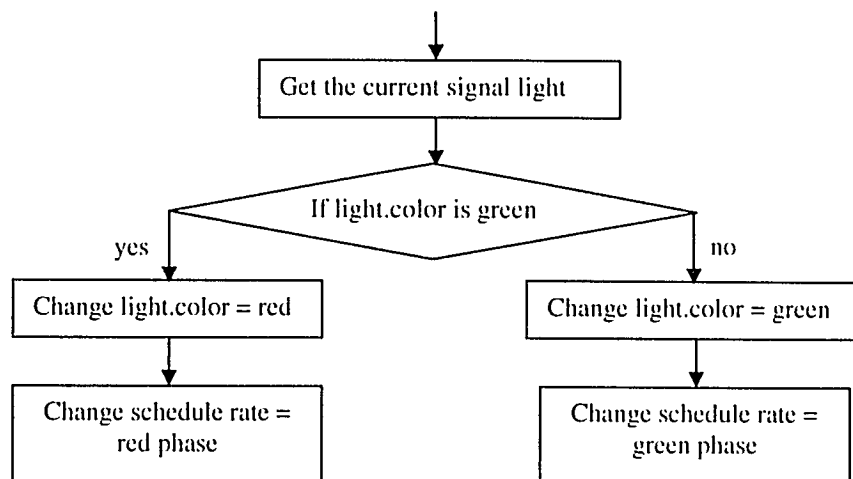


Figure 4.8: Traffic Light Scheduler Flow Chart

TrafficGenerator object generates vehicle and puts them into the link if sufficient gap is found from the last vehicle of the link. In the simulation, vehicles are always considered as a discrete unit so the arrival of the vehicle completes only after the vehicle gets its full length into the link. To insert the vehicle into the link, first vehicle of the queue associated with the origin node is checked with the last vehicle of that link at each time step of the simulation. If there is sufficient headway available, then the insertion takes place otherwise the vehicle remains in queue and is carried to the next time step. The information used to calculate the headway is taken from previous time step. TrafficGenerator flow chart is given in the Figure 4.9.

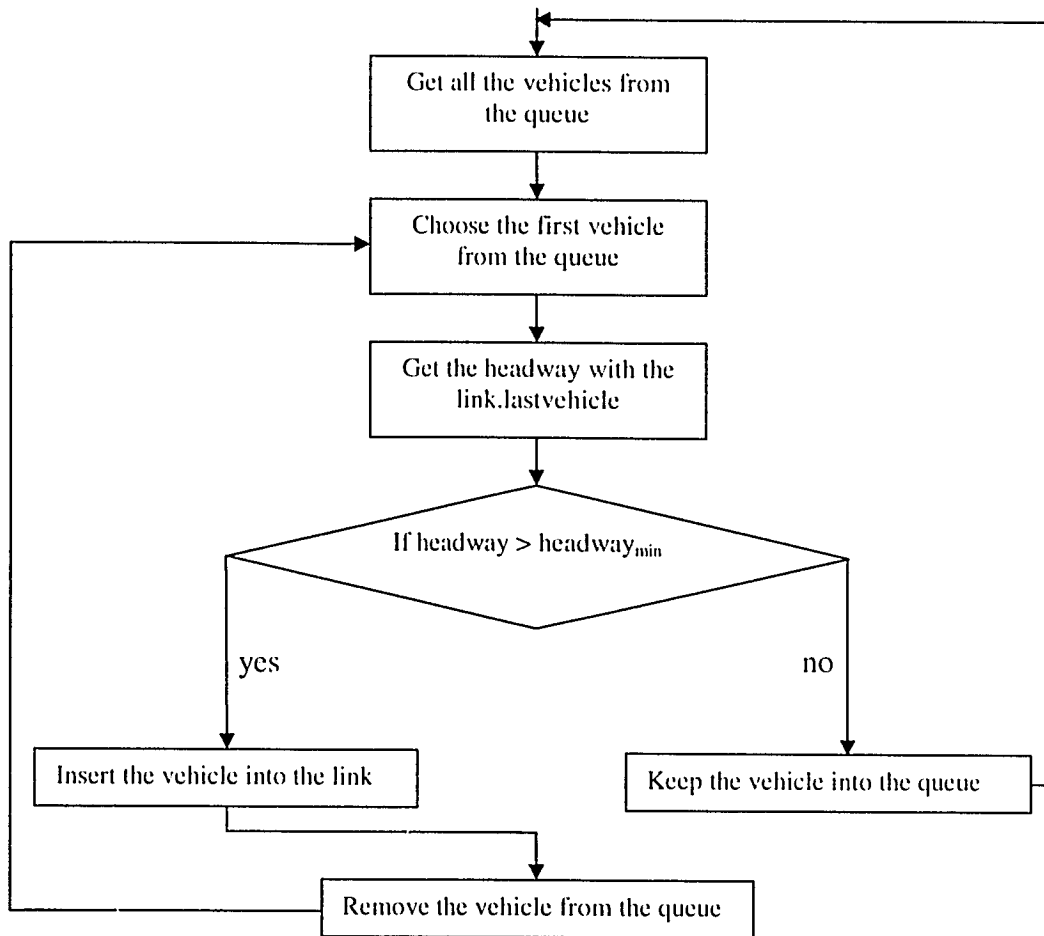


Figure 4.9: TrafficGenerator Flow Chart

4.3.2.2. Traffic flow propagation Sub-Model

The traffic flow propagation sub model of the simulator uses the cell transmission model [9, 16]. According to the cell transmission model each of the roadway is partitioned into discrete segment and time into discrete time steps. The traffic stream characteristics discussed earlier is assumed to satisfy this criterion of cell transmission model. For normal traffic conditions (Uninterrupted flow conditions) all the vehicles moving with free flow speed are expected to cross maximum one cell length in a single time step. For a typical roadway geometry given in Figure 4.10, the equation of the flow conservation is

presented as follows

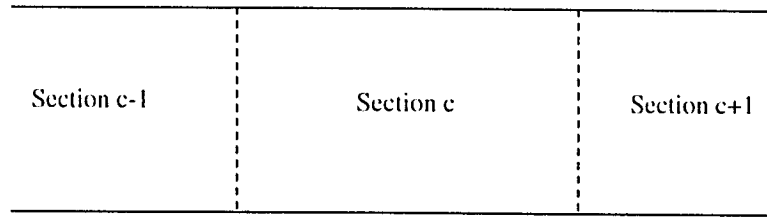


Figure 4.10: A Typical Link Segment

$$n_{c+1}(t + m_t) = n_c(t) \quad (4.5)$$

Where,

$n_c(t)$ = number of vehicles in cell c at time t

$n_{c+1}(t + m_t)$ = number of vehicles in cell $c+1$ (immediate downstream cell of c) at time $t+m_t$

The equation (4.5) holds true if there is no traffic queue in the downstream cell and the downstream cell can accommodate all the vehicles from its immediate upstream cell satisfying the jam density conditions of traffic flow models. To model a realistic traffic flow, congestion in the downstream cell that slows down the traffic in the upstream cell is incorporated using two variables. They are as follows:

$N_c(t)$ = maximum number of vehicles that can be present in segment c at time t

= product of link length and jam density of the link.

$q_c(t)$ = maximum number of vehicles that can flow in segment c at time $(t, t + m_t)$

= product of maximum capacity of the segment and simulation time step.

$$\text{So, } \Delta_c = u_j * m_t \quad (4.6)$$

$$q_c(t) = q_{\max} * m_t \quad (4.7)$$

$$N_c(t) = k_j * \Delta_c \quad (4.8)$$

Where,

Δ_c = length of the cell in meter

m_t = simulation update step in seconds

q_{\max} = maximum flow or capacity in veh/h

k_j = jam density in veh/km

From flow density model discussed earlier, flow propagation and flow conservation equations can be redefined as follows:

$$O_c(t + m_t) = \min[\{N_{c+1}(t) - n_{c+1}(t)\}, q_c(t), q_{c+1}(t), n_c(t), m_t(t)] \quad (4.9)$$

$$I_{c+1}(t + m_t) = O_c(t + m_t) \quad (4.10)$$

Where,

$O_c(t + m_t)$ = number of vehicles leaving cell c at time $(t, t + m_t)$

$N_{c+1}(t) - n_{c+1}(t)$ = maximum number of the vehicles that can be present in segment $c+1$ at time t

$q_c(t)$ = maximum number of vehicles that can flow in segment c at time $(t, t + m_t)$

$q_{c+1}(t)$ = maximum number of vehicles that can flow in segment $c+1$ at time $(t, t + m_t)$

$n_c(t)$ = number of vehicles in the segment c at time t

$m_c(t)$ = number of vehicles can cross segment c at time t using the segment current velocity.

$I_{c+1}(t + m_t)$ = number of vehicles entering segment $c+1$ at time $(t, t + m_t)$

The flow propagation equation (4.9) determines the number of outgoing vehicles for a particular cell under current time step traffic conditions. Flow conservation equation (4.10) assures that number of vehicles entering into a downstream cell is equal to the number of vehicles leaving from the upstream cell. For the signalized intersection, the outflow from a head segment of any link depends on the current state of the traffic signal. The segment outflow under traffic light conditions is given by following equations:

$$q_c(t) = \begin{cases} q_c & \text{if the downstream signal light state is green} \\ 0 & \text{if the downstream signal light state is red} \end{cases}$$

The outflow from links that end at a peripheral node of the network are not restricted to flow propagation equation because the vehicles that reach the destination are automatically removed from the network. When advancing from one segment to another, individual acceleration and deceleration characteristics of the vehicle are not modeled to adjust the speed differential between the segments. This is due to the aggregate nature of the traffic characteristics of all the vehicles in a particular segment. To model the congestion or traffic signal, all vehicles are assumed to be able to stop instantaneously.

4.3.3. Advance Phase

In this phase, vehicles are advanced to the immediate next segment from their current segment using the segment inflow and outflow information received from the update phase. Necessary turning maneuver are also considered for the vehicles that reach the end of the link or waiting for the traffic light. For discrete vehicle movement in this simulator, the last variable in the equation (4.9) is always set to a whole number. In the advance phase this variable is assigned with an integer value using the vehicle count only those getting the full length into the new segment. For each vehicle movement the simulation time step is divided into two parts. First time slice is used to cross the remaining distance

of the segment from the vehicle current position using segment current speed. The second time slice multiplied by the new segment speed gives the new position of the car into the new segment for the current time step. In Figure 4.11, the simulation time step t is divided into two parts t_1 and t_2 for segment A and segment B

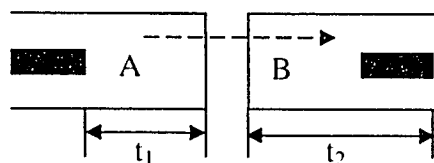


Figure 4.11: Time Division for Vehicle Transfer

respectively. Vehicle position coordinate into any segment is measured from the tail of each segment and is calculated by multiplying the average speed of the segment with the t_1 or t_2 depending on the exit or entry segment.

4.3.3.1. Left Turn Movement

For the turning movements each lane of head segment is considered separately depending on the type of turning maneuver. The straight and right turning maneuvers are same as normal vehicle transfer between the intermediate segments of a link because there is no need to give any priority to any particular vehicle. For the left turning movement, minimum time headway of 2 sec is considered and priority is given to the vehicle taking the straight turn from the oncoming roadway link. The flow chart for left turning movement is given below:

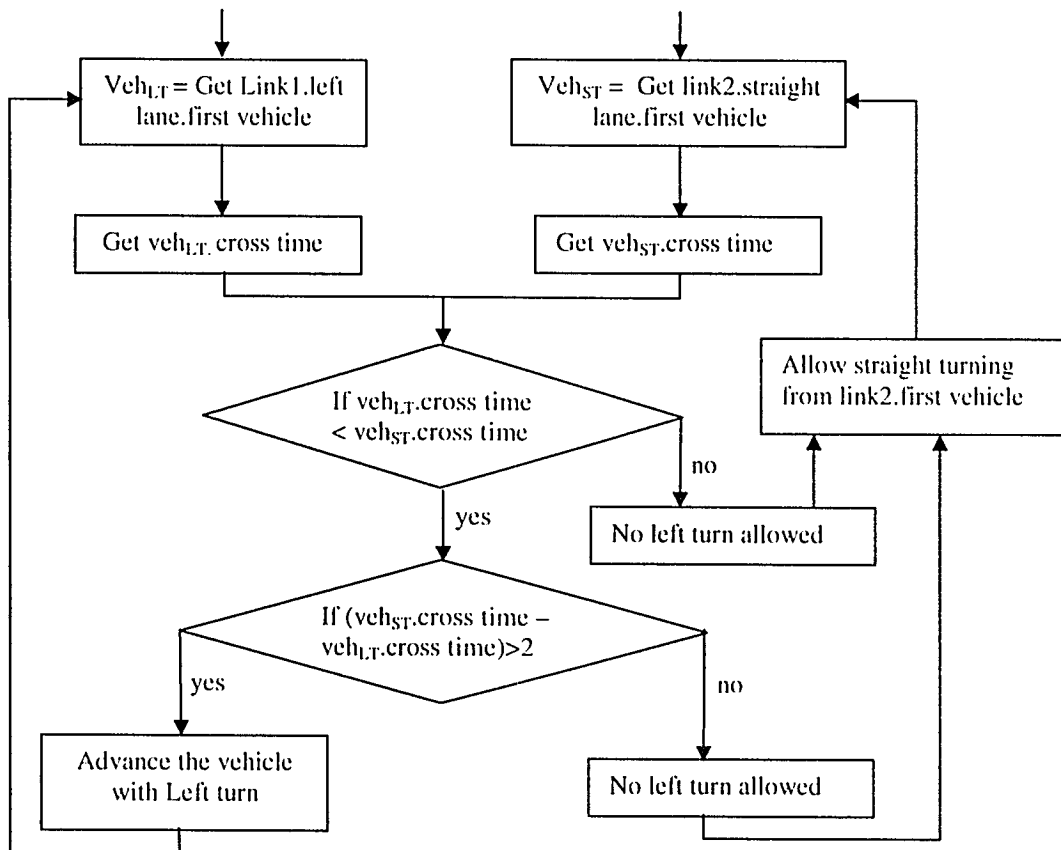


Figure 4.12: Flow Chart for Left Turn Movement

4.4. Discussion

The traffic simulator is intended to simulate the macroscopic traffic behavior of vehicles on a coded network. In each time step during simulation, each of link cells of the network update the cell traffic information like total number of vehicles, vehicle density, flow and average speed of the traffic stream. As mentioned earlier, the simulator treats vehicles on individual basis. Each vehicle updates its position for the next time step using the cell traffic information and all vehicle movements are coordinated with the intersection traffic lights. For each time step, the simulator stores the total number of vehicles running on the network and the total number of vehicles that reached the destination in two separate variables. Also, each vehicle stores the trip start time while entering into the network and

trip end time while exiting from network. Upon termination of the simulation, individual vehicle travel information is released into the data file and vehicles which traveled between the same origin-destination pair are used for travel time savings calculation.

CHAPTER 5

COMPUTER PROGRAMS

5.1. Introduction

The object oriented implementation of the transportation network is used to develop the macroscopic traffic simulator. Each feature of the network is implemented as a separate object. Descriptions of each object, relationships to the other object and the responsibilities to the simulator are discussed in this chapter.

5.2. Correlation of the Problem Domain to an Object-Oriented Data Model

The successful implementation of an efficient traffic management system depends on the capability of dynamic data processing regarding each feature of transportation network along with both the static and dynamic traffic information. A data-model is an abstract representation of some real-world situation or domain of interest. In this thesis, the transportation network is correlated to an object-oriented data model which allows the efficient interaction between transportation network characteristics according to [25]. The key element of an object-oriented data model is to implement structured and efficient data structures referred to as objects. Each class in the data model is an instance of an object. A class represents a kind of object that has the similar data representation and the behavior that mirror the physical reality of that particular object [25, 30]. The three main features of an object are as follows:

- **Abstraction:** the act of representing essential features without including the explanations.
- **Inheritance:** the ability to acquire the properties of objects of other class.

- **Polymorphism:** the ability to take the other form while keeping the original structure.

The implementation of object-oriented approach is justifiable as it removes some of the flaws encountered with the functional or procedural approach. The procedural approach emphasizes the functionality and does not allow the structure of data whereas in the object oriented approach, data is treated as a critical element and also is protected from any outside functions. Moreover in case of any internal changes of the data structure, object identity is not lost and can be implemented by a class as before by using the features of the objects discussed before.

5.3. Object Oriented Approach

A transportation network data model can be represented by the feature of transportation network such as intersection, road section, vehicle, traffic lights etc. The object oriented approach assumes that a feature is an object having a set of properties and a set of relationships with other objects. Topology is defined as arrangements of features of the network and their connectivity which is one of the core concepts in defining transportation network is imbedded in object oriented representation [41]. In the simulator, roads in the network are modeled as link object and the intersections are modeled as node object. Also, each link is considered as linked data structure of several cell objects which is done to implement cell transmission model. Vehicles are considered separate objects which are created at the origin nodes and destroyed upon reaching to the destination node. All intersections are considered signalized where traffic lights are also considered as separate objects which operate through pre-time signal settings. Depending on the flow in the link and available green time, vehicles queue up on the cell and take

appropriate turn upon reaching the end of the link. For simplicity, no overtaking is allowed and all the vehicles are assumed to follow the fundamental traffic flow equations. The attributes and the functions of all the objects have been implemented in such a way that they mimic the real world transportation network representations.

5.3.1. Java Implementation

The object oriented approach of the simulator is implemented in JAVA™. The program is written and compiled against Sun® Microsystem’s J2SE version 1.4 java libraries. The program is developed in Borland® JBuilder® 2005 enterprise edition. Java is chosen because of its object oriented nature and the extensive functionality in the new collection library is included in J2SE version 1.4.

5.4. Class Definitions and Relationship

To show the relationship between the object classes implemented in the simulator, Unified Modeling Language (UML) diagrams of each object class are extracted from the JBuilder® UML browser. The definitions of the UML diagram symbols using the standard Java terminology are given below:

Table 5.1: UML Diagram Definitions

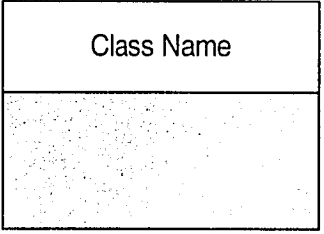

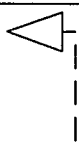
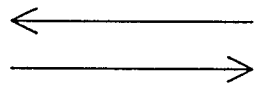
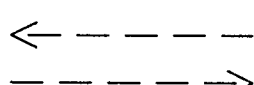
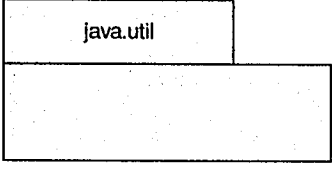
Diagram	Definitions	Diagram Example
Classes	Structures that defines objects. A Class definition defines fields and methods.	
Extended Class	Classes that extend (inherit from) the superclass. Also called subclass	

Diagram	Definitions	Diagram Example
Implementing Interface	Classes that implement the central interface.	
Association/ Reverse association	Specialized dependencies where a reference to another class is stored.	
Dependencies/ Reverse dependencies	Using relationships in which a change to the used object may affect the using object.	
Packages	Collections of related classes	

All the object classes of the simulator is the part of java Package refer to “roadsim”.

The following classes are implemented for the simulator:

Cell: This is the smallest building block of the road network. Each of this object is part of link of the network. It has a fixed capacity, fixed length, link specific unique ID and a queue. Vehicles can enter only through the tail of the cell and leave through the head of the cell.

- Head Cell: Consists of three lanes for left, straight and right turning movements. Each of these cells is linked to the downstream link cell through the network topology. Movements through these lanes are coordinated with traffic signal.
- Transfer Cell: Single lane cell as vehicles only traverse through these cells from upstream to downstream direction towards their destination.

- Tail Cell: Single lane cell. Each of these cells is linked with previous link cell through the network topology.

The UML diagram of the Cell object class is as follows:

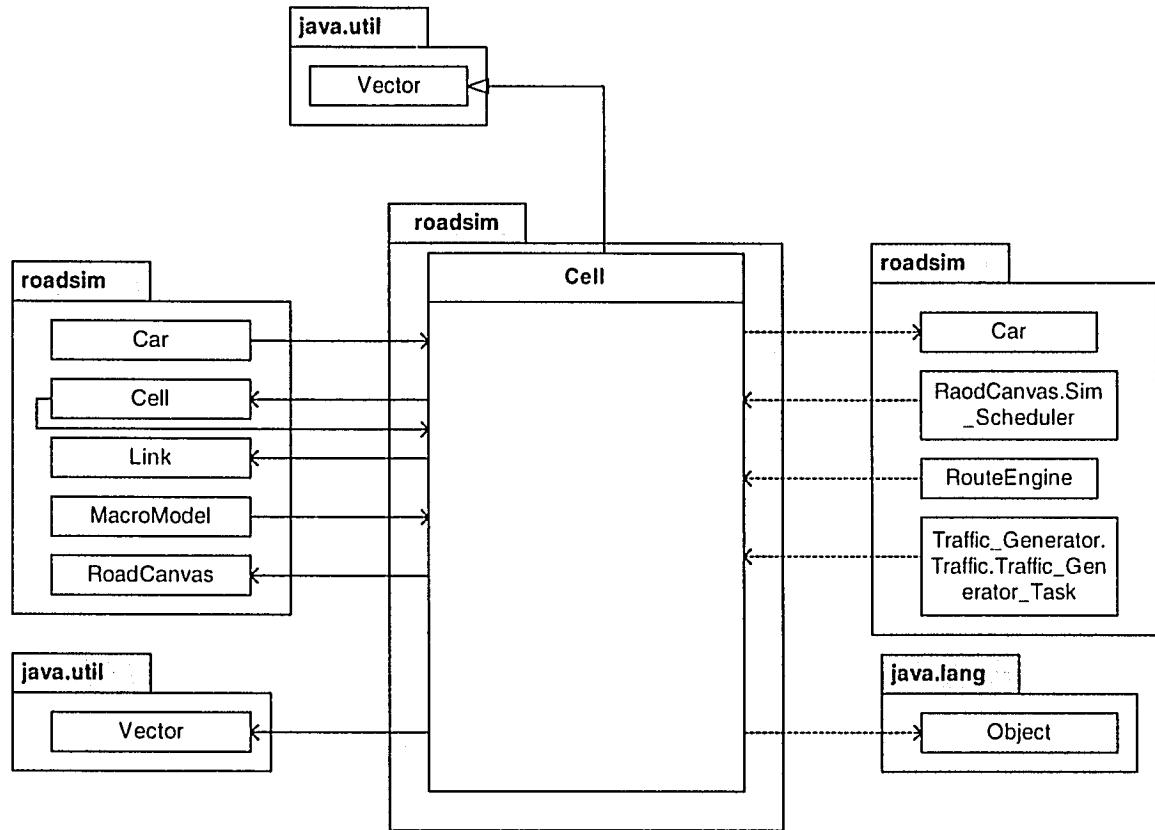


Figure 5.1: UML Diagram of Cell Object Class

The sample program code for Cell object class is given in Appendix B.1

Link: This object consists of several cell objects depending on the length attribute of this object. All the cell objects belong to a particular link object have same traffic characteristics like capacity, speed, density and flow which also represent traffic characteristics of the link. The link object connects two intersection objects as `Head_node` and `Tail_node`. Each object also point to the other link object for left, straight and right movement through `Head_node`. Tail cell of a link object is responsible to take

decision whether to put vehicle in the queue or to insert into the link by checking the link capacity. The UML diagram of the object class is as follows:

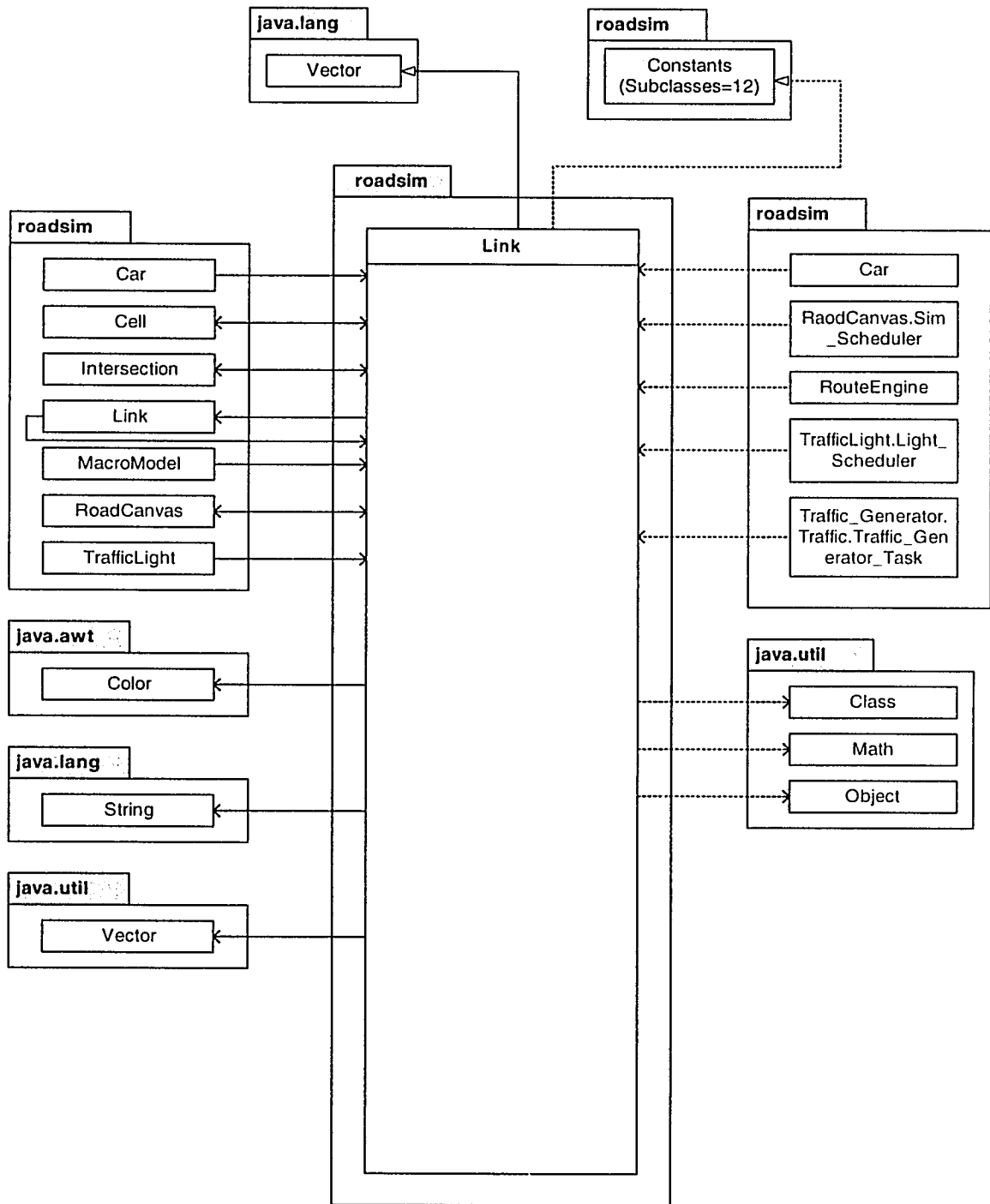


Figure 5.2: UML Diagram for Link Object Class

The sample program code for Link object class is given in Appendix B.2

Intersection: This object acts as a junction of link objects and resembles the intersection of the real-world transportation network. Each of these objects can point at most four intersection and link objects to mimic the real-world four-leg intersection. The movement through any intersection is coordinated according to the TrafficLight object attached to the intersection. Three types of intersections or nodes are considered as follows:

- Entry Node: These are the peripheral nodes through which vehicle enters into the network according to the given travel demand. TrafficGenerator object are attached to these nodes to create the vehicle.
- Exit Node: These are also the peripheral nodes through which vehicle exits from the network.
- Transfer Node: These are the internal nodes of the network to simply removes the vehicle from one link and transfer it to other link depending on the vehicle turning movement.

The UML diagram of the Intersection object is given in the Figure5.3.

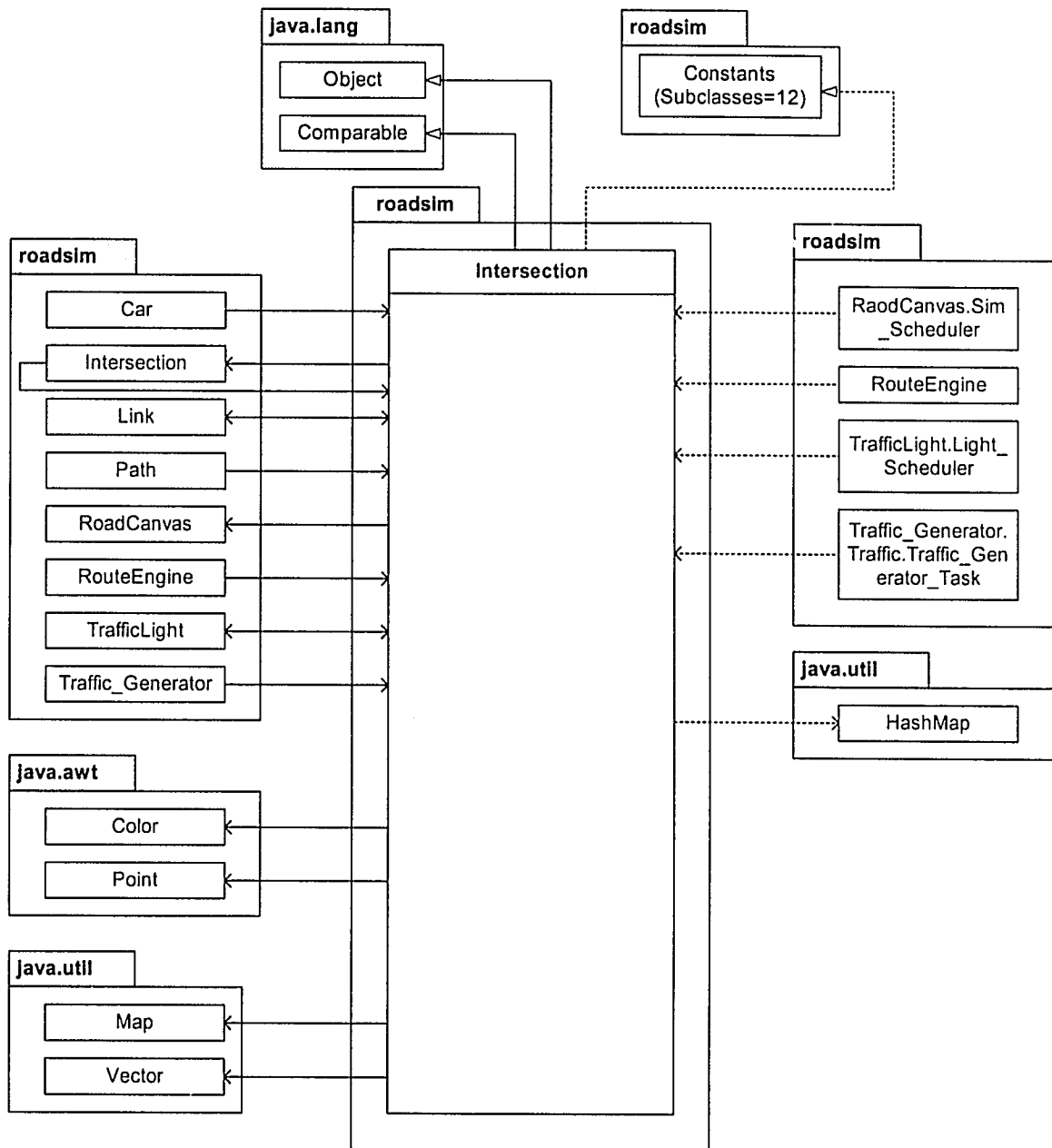


Figure 5.3: UML Diagram for Intersection Object Class

The sample program code for Intersection object class is given in Appendix B.3

Car: This is the most important object of the simulator through which performance of guided vehicles is evaluated. Three types of vehicles are considered as follows:

- Random Vehicle: These vehicles are generated randomly during the initialization state of simulator. They have no specific origin and destination node and are not

considered for the performance evaluation. This type of vehicle represents the drivers who have no definite destination to reach and choose the intersection turning maneuver randomly.

- **Unguided Vehicle:** It originates at the designated origin intersection and exits through the designated destination intersection. Vehicles receive the path information for the particular origin-destination node while entering the network and follow the path for the entire journey. Upon reaching the destination node vehicle release the trip information regarding total travel time, travel speed into a data file. The shortest path calculation for this type of vehicle is performed by Dijkstra's static shortest path algorithm. The vehicle represents the drivers who use pre trip information for their route choice.
- **Guided Vehicle:** It also originates at the designated origin intersection and exits through the designated destination intersection. Vehicles are assumed to be equipped with the route guidance system through which they receive the path information for particular origin-destination each time they reach any intersection and change the path accordingly. Upon reaching the destination intersection, vehicle releases the trip information regarding total travel time, travel speed into a data file. The shortest path calculation for this type of vehicle is performed by proposed dynamic shortest path algorithm. The vehicle represents the drivers who use both pre trip and en-route information for their route choice.

UML diagram of the Car object is as follows:

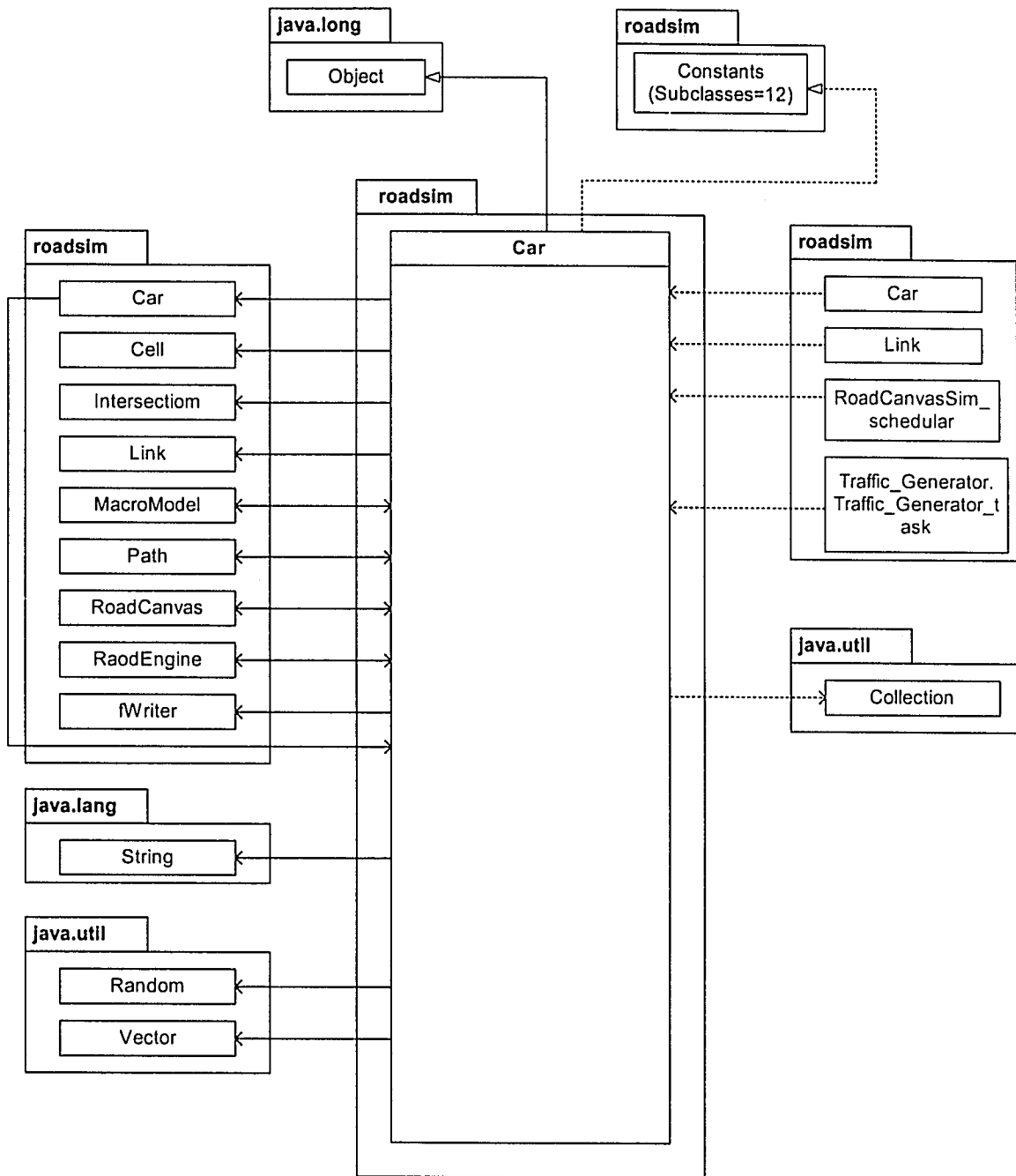


Figure 5.4: UML Diagram for Car Object Class

The sample program code for Car object class is given in Appendix B.4

- **TrafficLight:** This object is attached with the intersection object and activates during the initialization of the simulator. This object changes the green and red state according to the pre-timed signal settings. All the vehicles objects get the trafficleight signal state

through the link object they belong to for any particular simulation time step. UML diagram of TrafficLight object is as follows:

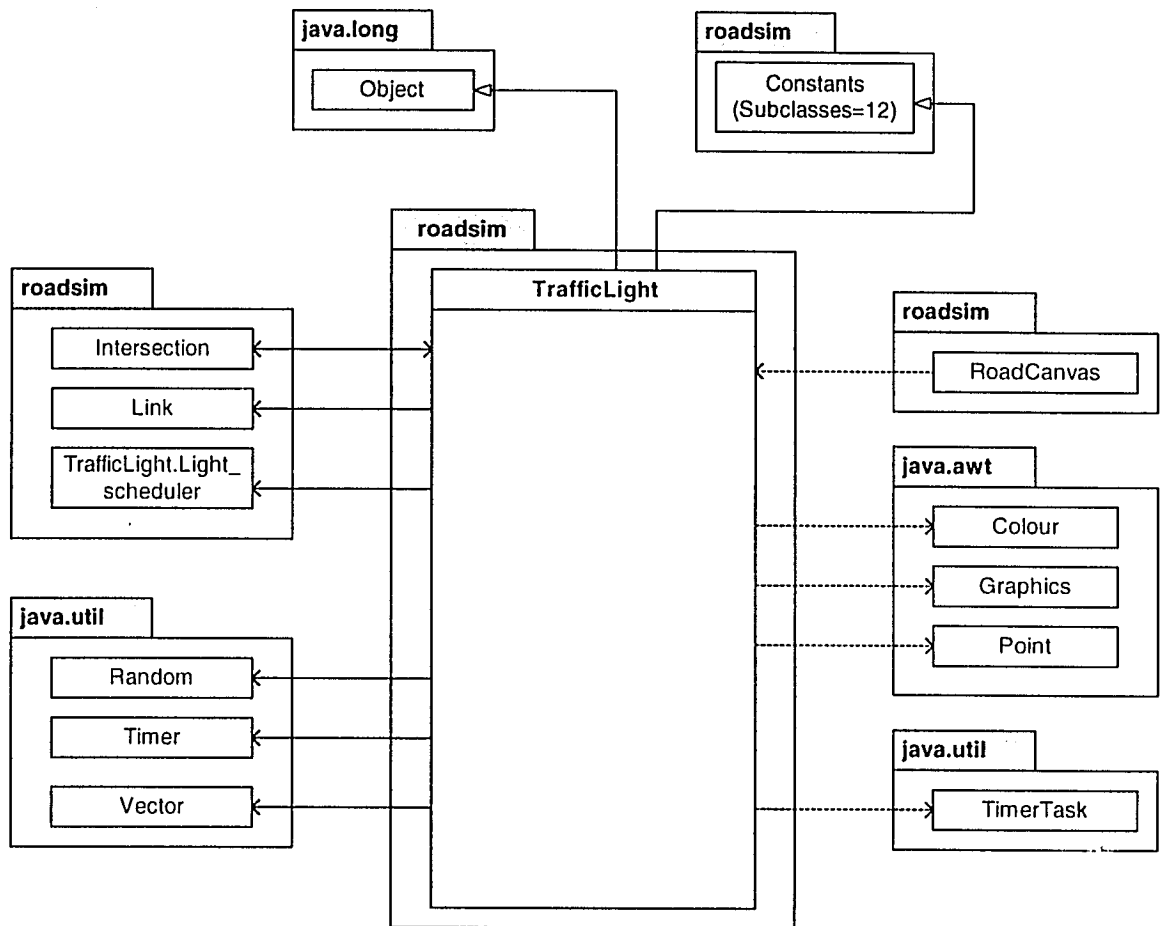


Figure 5.5: UML Diagram of TrafficLight Object

The sample program code for TrafficLight object class is given in Appendix B.5

TrafficGenerator: This object is responsible to generate the vehicle according to the given travel demand by keeping constant time headway between the vehicle. These objects are attached to the designated origin and destination nodes. The object generates vehicles at a uniform rate between the origin destination pair from the according traffic volume data. The object also keeps track of the total number of generated guided and unguided vehicles between origin destination pair to satisfy the vehicle composition according to

the traffic scenario being simulated. UML diagram of Trafficgenerator object class is given as follows:

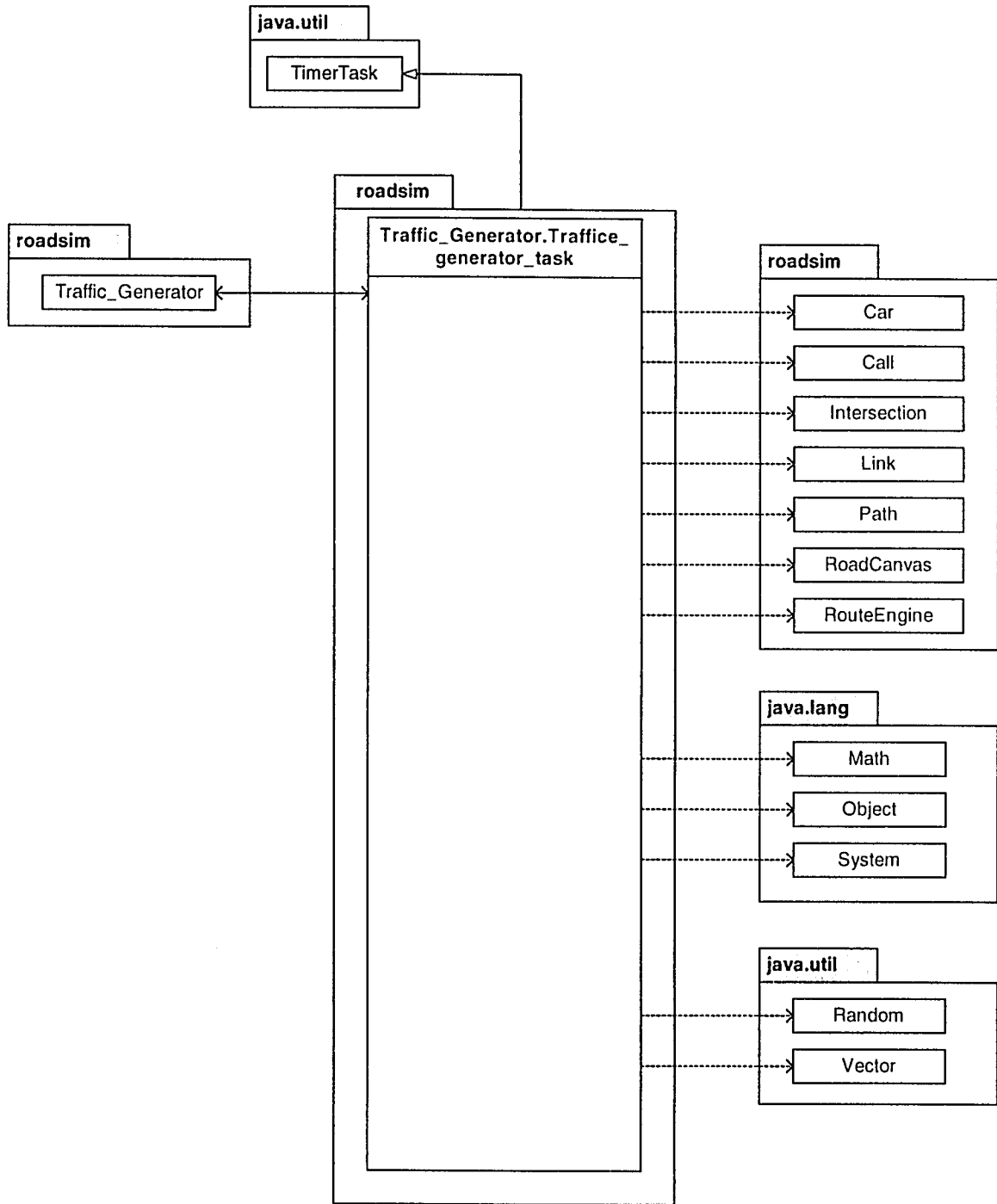


Figure 5.6: UML Diagram of TrafficGenerator Object Class

The sample program code for TrafficGenerator object class is given in Appendix B.6

▪ **MacroModel:** This object implements the macroscopic traffic flow dynamics considered in this study. During each time step, this object determines the maximum number of vehicles that can exit from a particular cell object by taking into consideration the link average travel speed, density and flow for that time period. The object variables store this information and all the vehicle objects use this information to update their individual position into the cell for the next time step. UML diagram of this object is as follows:

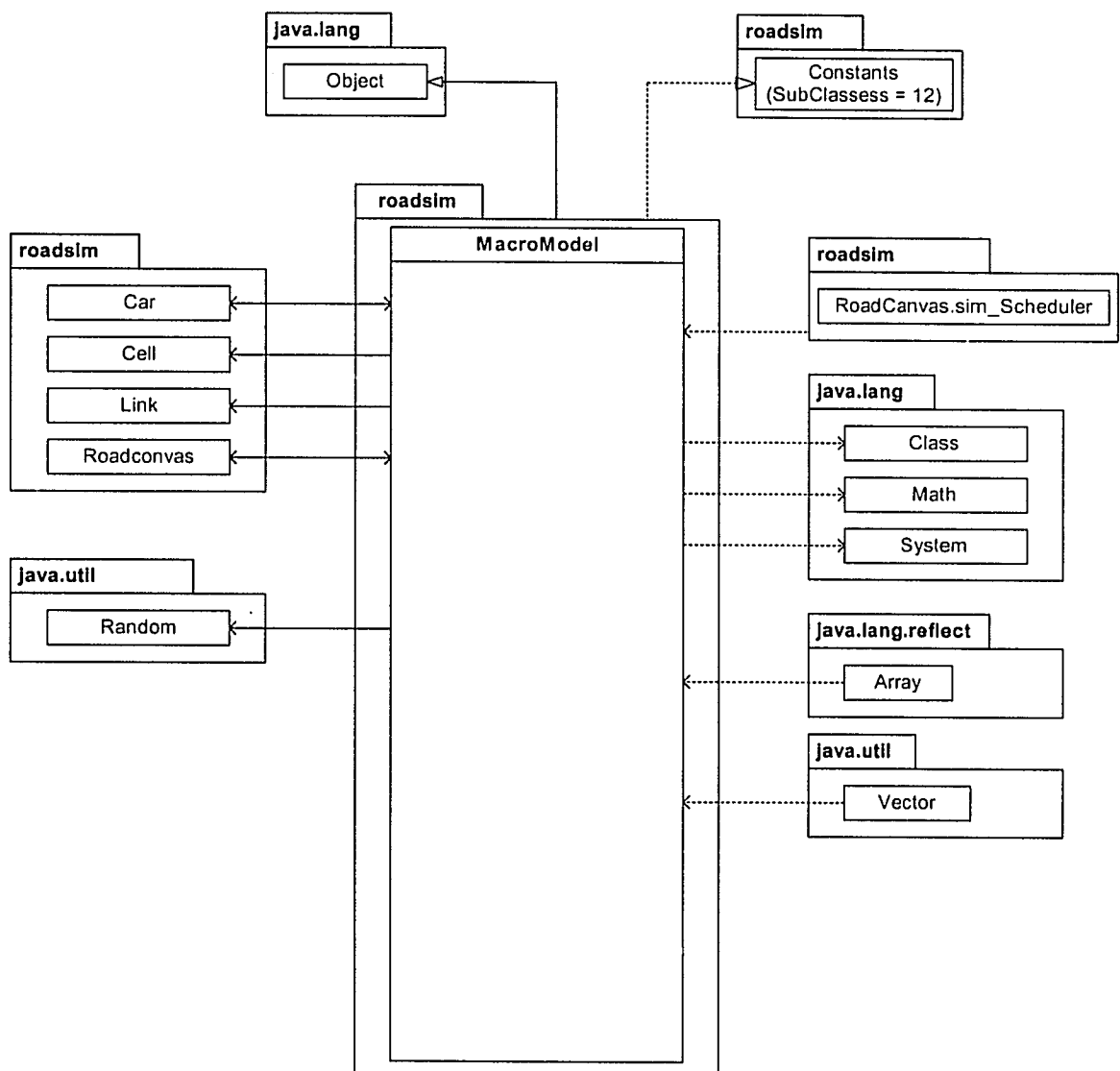


Figure 5.7: UML Diagram of MacroModel Object Class

The sample program code for MacroModel object class is given in Appendix B.7

Path: This object stores the sequence of intersections (Intersection object) between any origin-destination pair. Upon the call to the RouteEngine object by any vehicle object this object stores the sequence of intersections on the shortest path between particular origin-destination pair. UML diagram of the object is as follows:

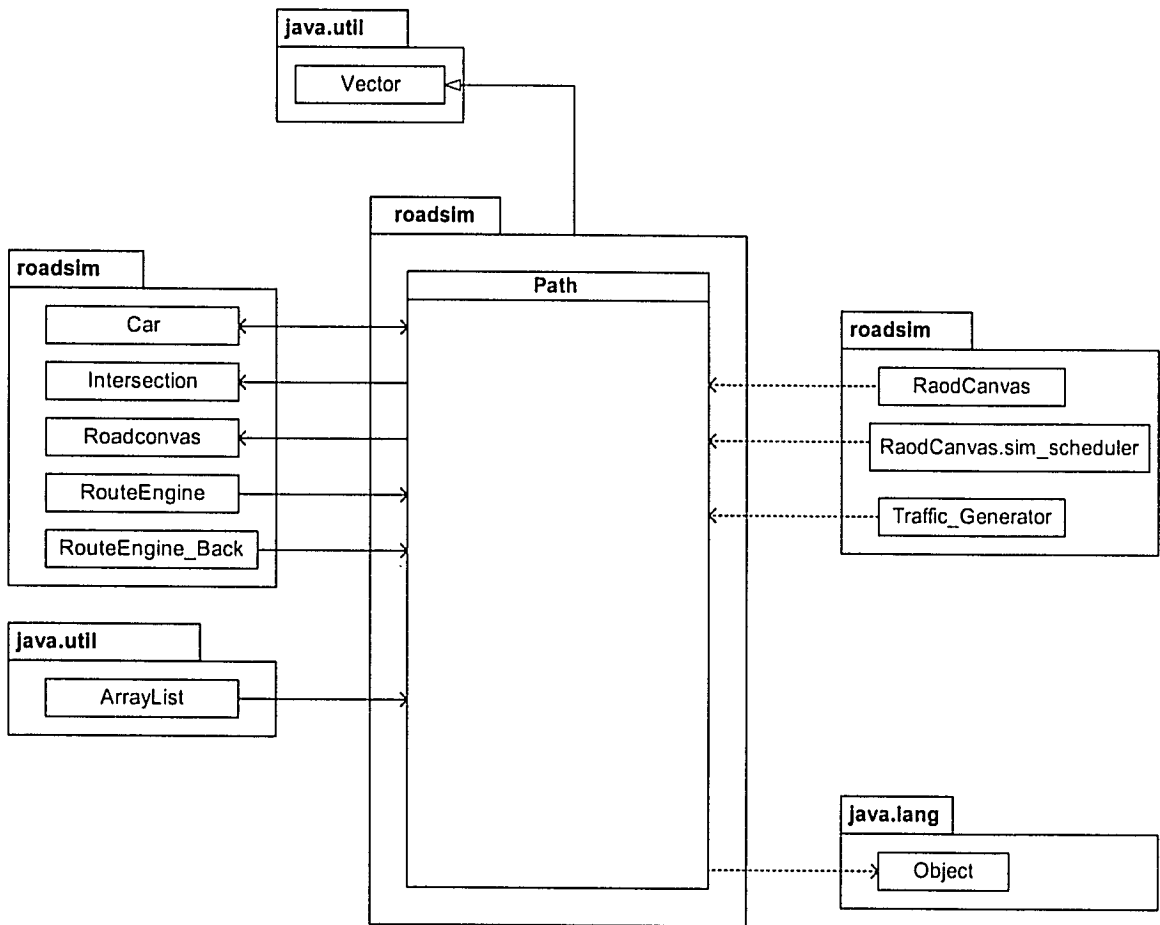


Figure 5.8: UML Diagram of Path Object Class

The sample program code for Path object class is given in Appendix B.8

RouteCanvas: This object is the base object that coordinates all the objects for each time step. This object read all the input data from specified data file and all the objects are initialized through this object. A java Timer object is incorporated as Sim_Scheduler to

control the simulation time frame. The simulation time step is updated according to the preset time interval and after total sampling period of the simulation the vehicle object stores the travel data into a data file upon calling this object. UML diagram of the object is given below:

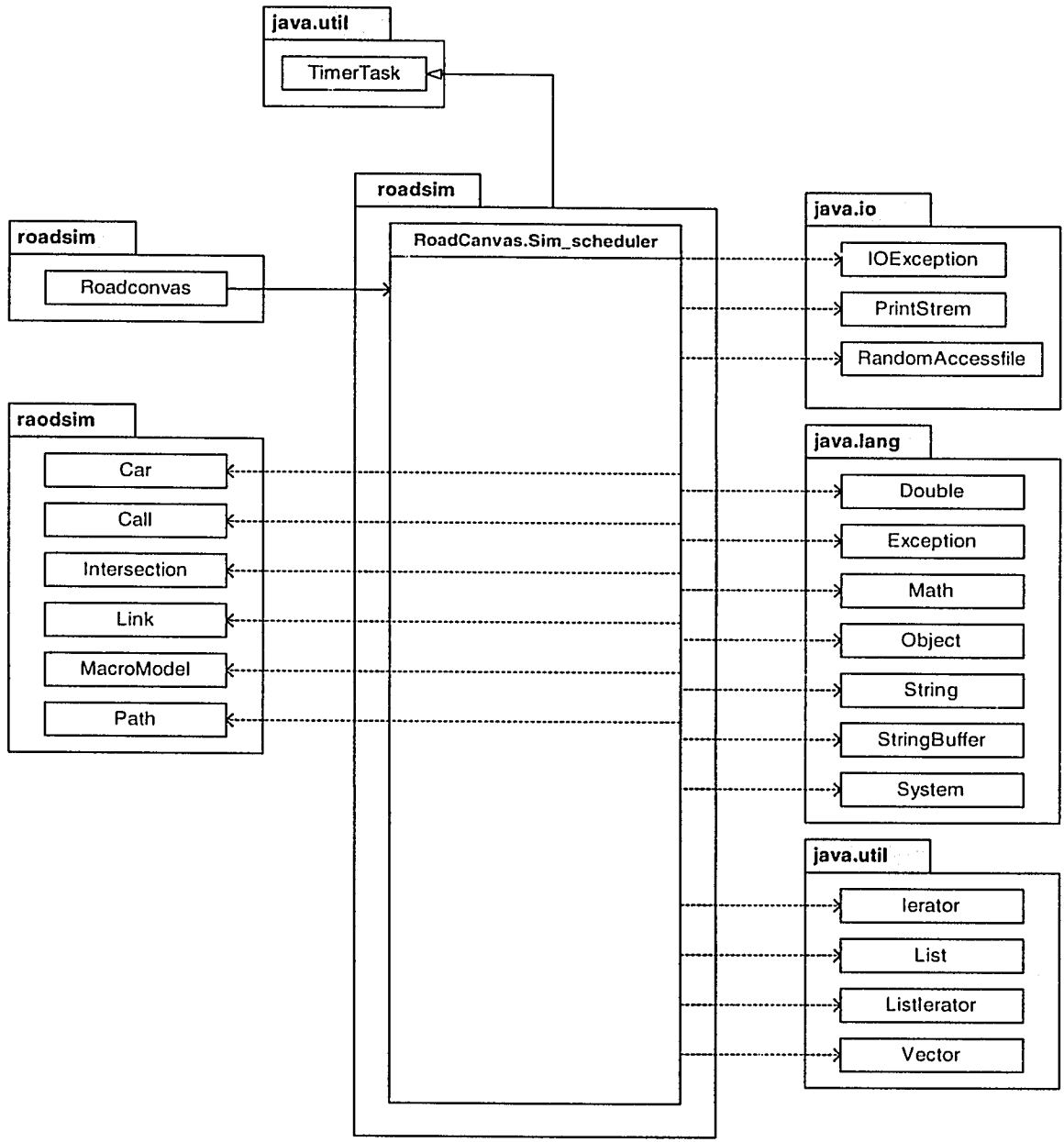


Figure 5.9: UML Diagram of RoadCanvas Object Class

The sample program code for RoadCanvas object class is given in Appendix B.9

RouteEngine: This object is used to calculate shortest path between the given origin and destination intersection for both the guided and unguided vehicle. The unguided vehicle calls this object to give the shortest path using static shortest path algorithm while entering into the network. The guided vehicle calls this object to give the shortest path using the Bi-objective Bidirectional Heuristic Path Search Algorithm (BBHPSA) each time they enter into the new link. UML diagram of the object is given below:

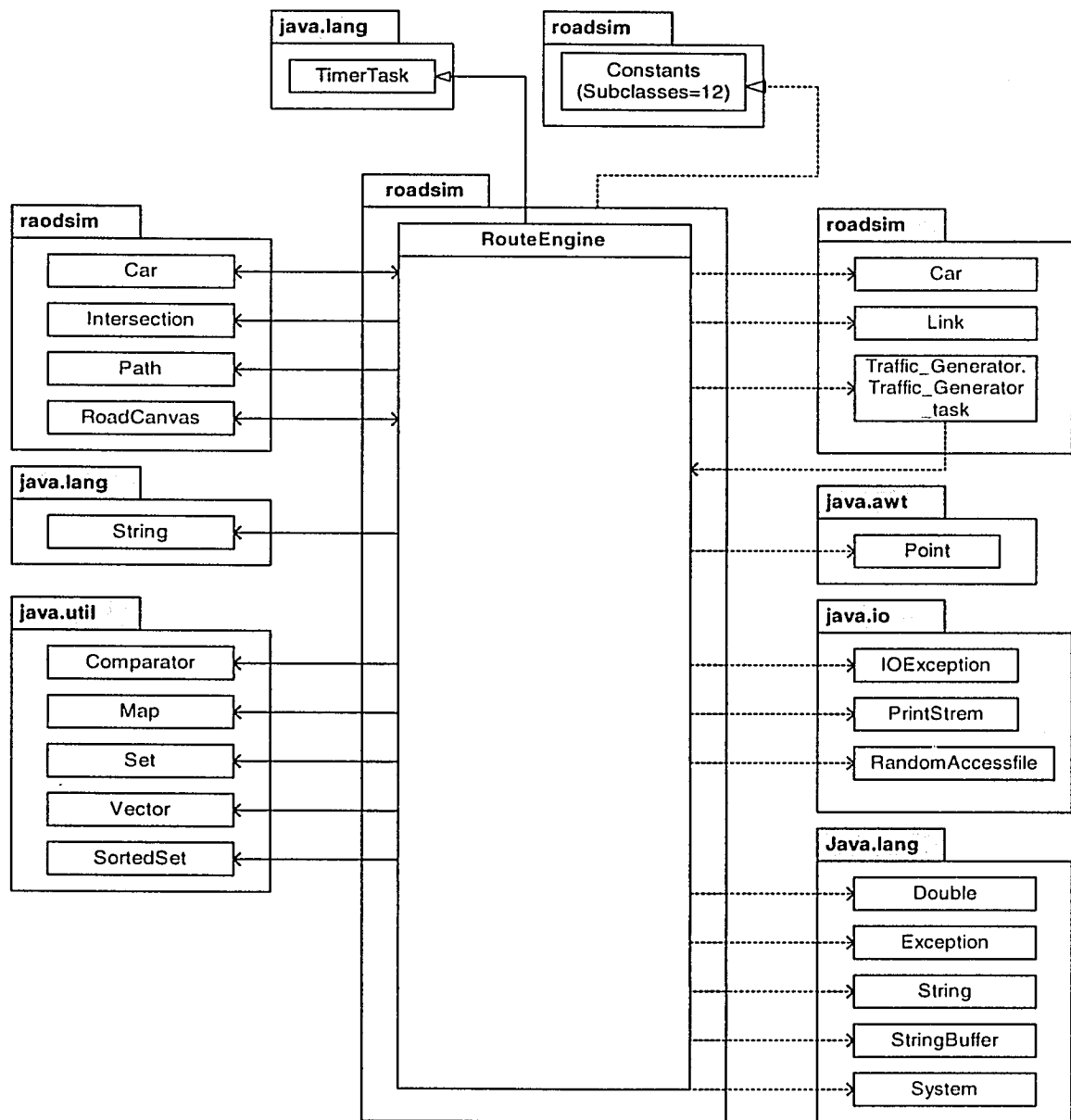


Figure 5.10: UML Diagram of RouteEngine Object Class

The sample program code for RouteEngine object class is given in Appendix B.10

5.5. Summary of the Computer Programs

The object oriented implementation of the traffic simulator discussed in this chapter corresponds to the different phases of the simulator given in previous chapter. During loading phase, all the objects of the simulator are instantiated and variables of each object are initialized. Network topology is built through relationship between the Cell, Link and Intersection object. Upon initialization the simulation timer object, the vehicle object enters in the network according to the travel demand data given earlier. In each time step of the simulation, MacroModel object updates the traffic flow variables of each Cell object as well as the position coordinate of the Car object. RoadCanvas object calculates the outflow from each Cell object through the flow propagation equation given earlier. Traffic flow variables from the previous time step are used to determine the outflow from Cell object for current time step. Upon termination of the simulation, each Car object which reached the destination release the trip information to data files for analytical computation of the simulation results.

CHAPTERS 6

RESULTS OF TRAFFIC SIMULATION MODEL

6.1. Introduction

In this chapter, simulation results from the traffic simulator under various traffic scenarios are reported for the hypothetical network given in the previous chapter. The analytical perspective of the results for each scenario is also discussed.

6.2. Simulation Results of Hypothetical Network

The network is simulated with the assumed travel demand data given in the previous chapter to estimate the possible travel time savings for the 11 different traffic scenarios varied by the presence of certain percentage of guided vehicle in the traffic fleet. The simulation run is started with the traffic scenario considering that all the vehicles in the fleet are unguided vehicle. For each of next run of the simulation, the traffic scenario is varied by increasing the percentage of guided vehicle in the network by 10% from the previous run. So, in the first run only the unguided vehicles and in the last run only the guided vehicles are simulated for the respective origin-destination pair according to the assumed trip table. Trip production and attraction for any of the nodes and total number of trips are kept constant to compare the travel time savings between any particular pair of nodes. For each run, the travel times of each vehicle are cumulated depending on the total number of intersections the vehicle crossed during its course of travel rather than the vehicle traveling between the same origin-destination pair. The accumulation of travel time of each vehicle crossing the equal number of intersections to reach the destination can be justified as link length is same for all the links. So, all the travel path with equal

number of intersections have the same length. The average travel time for the vehicle crossing the same distance is then computed as follows:

Average travel time = Cumulated travel time/ Total number of vehicles.

Route Guidance (RG) level is expressed as “*Level of RG (%)*” according to the percent of guided vehicles in the network. So, Level of RG = 10 means, the amount of guided vehicles is 10 percent of the total number of vehicles in the network. In Table 6.1, the vehicle type composition for different traffic scenarios are given.

Table 6.1: Traffic Scenario Vehicle Composition

Scenario	Level of RG	Vehicle Percentage(%)	
		Guided	Unguided
1	0	0	100
2	10	10	90
3	20	20	80
4	30	30	70
5	40	40	60
6	50	50	50
7	60	60	40
8	70	70	30
9	80	80	20
10	90	90	10
11	100	100	0

The percentage of guided vehicle in each traffic scenario is varied according to the earlier discussion given in this chapter. It is observed from the simulation that the vehicle cross at most 6 intersections to complete the trip between any origin-destination pair in the network. So, maximum 6 intersections on the travel path are considered from the simulation results. In Table 6.2, simulation results of all the traffic scenarios are presented in terms of average travel time/km according to the total number of intersections the vehicle cross during the trip.

Table 6.2: Simulation Results for Travel Time/km

Level of RG(%)	Veh Type	No of Intersections					
		1	2	3	4	5	6
0	Guided						
	Unguided	103.62	105.64	115.69	128.24	138.30	147.58
10	Guided	100.70	103.42	114.07	126.65	135.02	142.18
	Unguided	102.72	105.10	115.31	128.05	137.88	147.04
20	Guided	99.56	102.75	112.72	124.85	132.54	139.75
	Unguided	103.38	105.05	115.32	128.04	137.38	146.47
30	Guided	104.22	102.28	111.92	123.27	130.20	137.90
	Unguided	101.33	104.75	115.07	127.65	137.21	145.93
40	Guided	102.24	101.76	110.65	121.69	127.87	136.89
	Unguided	98.90	104.76	114.56	126.85	136.92	145.61
50	Guided	103.66	101.53	110.29	120.25	125.17	134.03
	Unguided	98.21	104.56	114.54	126.26	136.82	145.04
60	Guided	104.66	101.03	109.84	119.44	123.08	132.47
	Unguided	103.34	104.33	114.32	125.65	136.26	144.61
70	Guided	103.86	100.41	109.49	118.91	121.68	130.75
	Unguided	102.26	104.10	114.07	124.32	135.55	143.61
80	Guided	101.32	100.09	108.57	118.03	121.19	129.32
	Unguided	102.19	103.42	113.72	123.92	134.54	142.61
90	Guided	103.26	99.61	108.31	117.32	121.06	128.32
	Unguided	103.43	103.04	112.82	122.86	133.71	141.61
100	Guided	102.08	98.77	107.81	116.84	120.72	127.03
	Unguided						

All the travel time values in Table 6.2 are in seconds. For each traffic scenario, the average travel times are given for guided and unguided vehicles. The first and last row of the above table is left blank intentionally to show that there is no guided vehicle in the first scenario and there is no unguided vehicle in the last scenario. From this table, at “Level of RG=10%” a guided vehicle that cross 6 intersections to reach the destination requires 142.18 sec to travel one kilometer distance whereas for the same number of intersections an unguided vehicle requires average travel time of 147.04 sec. The variations of average travel times/km given in Table 6.2 have been shown in the following Figures 6.1 (a) to (i) for mixed traffic scenario refer to Gradual Variation of Travel Time for mixed Traffic.

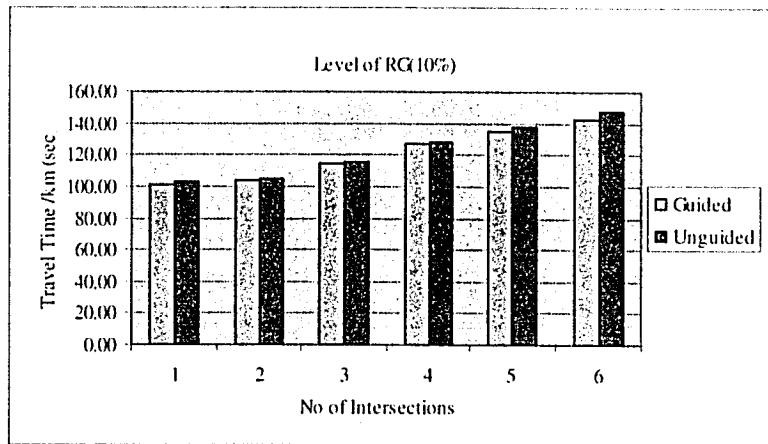


Figure 6.1(a): Scenario 2

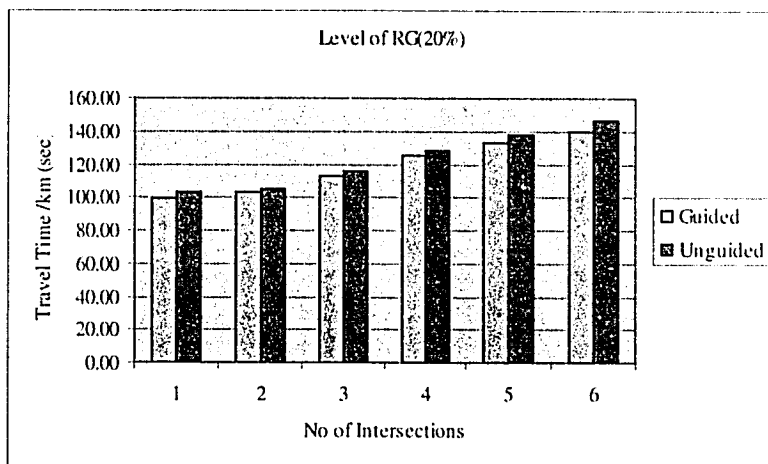


Figure 6.1(b): Scenario 3

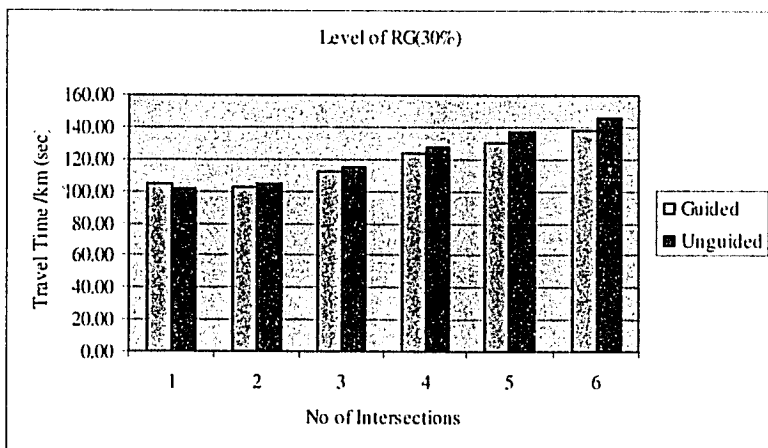


Figure 6.1(c): Scenario 4

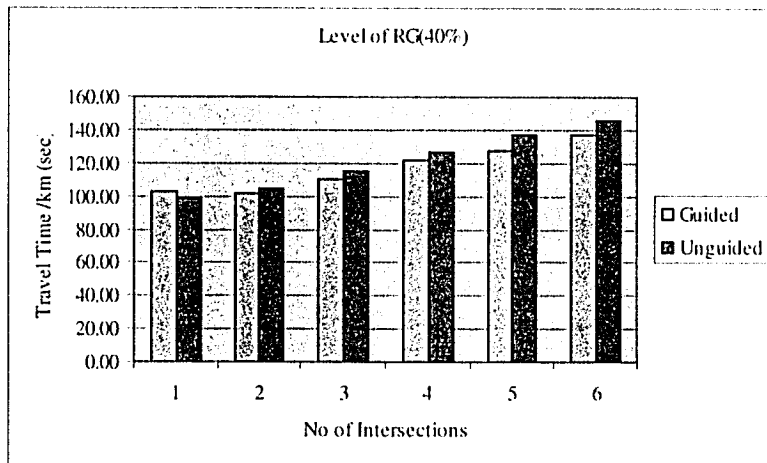


Figure 6.1(d): Scenario 5

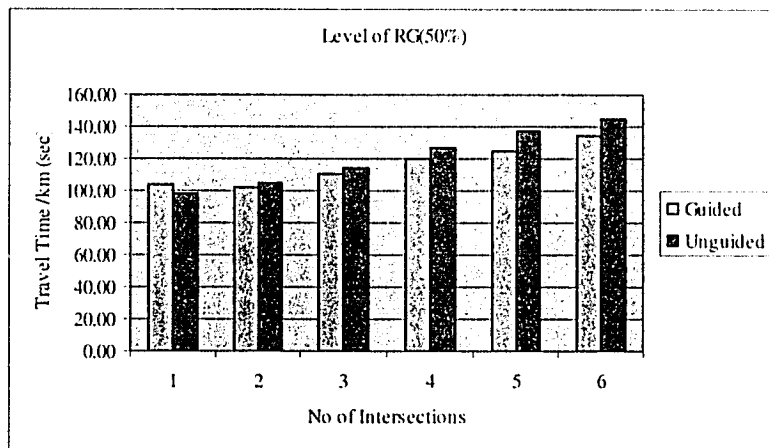


Figure 6.1(e): Scenario 6

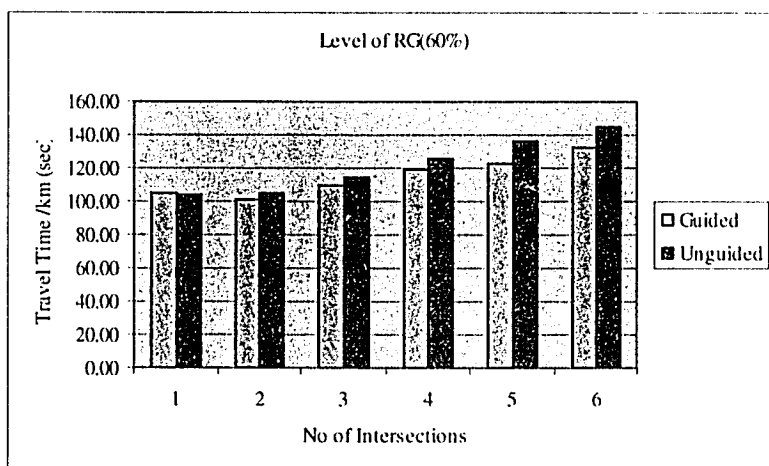


Figure 6.1(f): Scenario 7

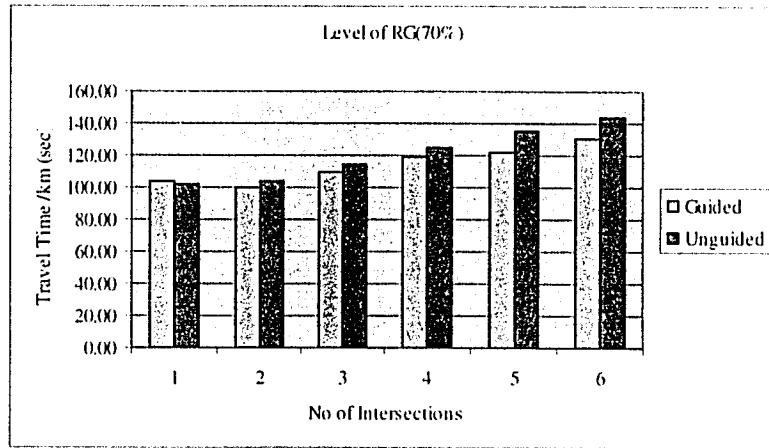


Figure 6.1(g): Scenario 8

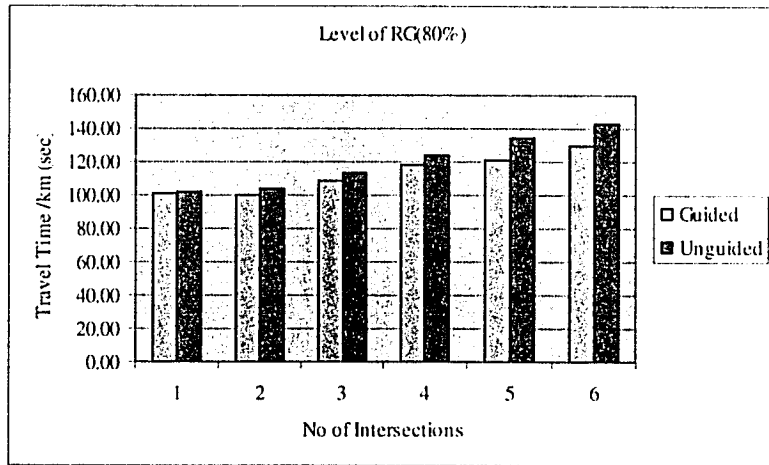


Figure 6.1(h): Scenario 9

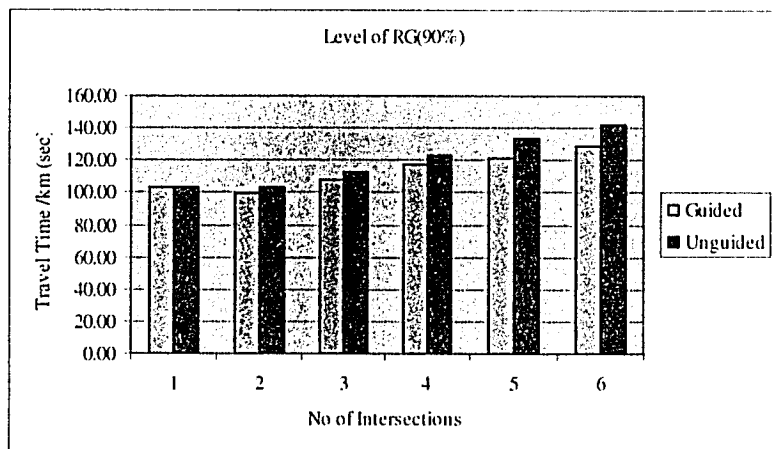


Figure 6.1(i): Scenario 10

In all the figures in 6.1, the gradual variations of the average travel times/km are shown for different composition of guided and unguided vehicles according to the different traffic scenarios. In Figure 6.2, average travel time/km is shown for traffic scenario I and traffic scenario II.

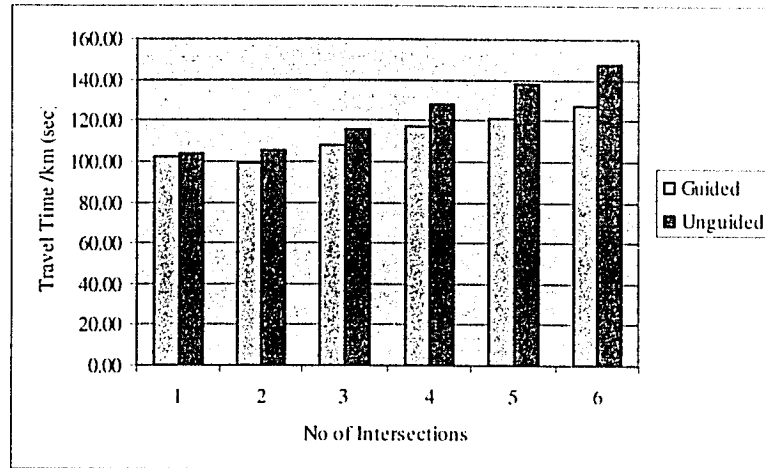


Figure 6.2 Variation of Travel Time for Guided and Unguided Vehicles

In Figures 6.1 and 6.2, average travel time/km are plotted against the number of intersections on the traveled path. The horizontal axis denotes the number of intersections, the vehicle crossed for its trip along the traveled path and vertical axis denotes the average travel time/km in seconds for guided and unguided vehicles into the network. The following interpretations can be made from these figures:

- The average travel time/km for both of guided and unguided vehicles increases with the total trip length.
- Benefits of guided vehicles in the total network traffic are relatively large for the longer trip length or travel path with more number of intersections. This is justifiable as the guided vehicles select their route based on the present traffic conditions upon

crossing the each intersection. So, for the longer routes, vehicles will get more chances for the route selection and to achieve the reduction in travel times.

6.2.1. Estimated Trip Time Savings

Trip time savings for different traffic scenarios are calculated based on the required trip time of the unguided vehicle in traffic scenario 1 (“*Level of RG=0%*”). In Table 6.3, trip times are shown for different traffic scenarios as discussed earlier.

Table 6.3: Simulation Results for Trip Time

Level of RG(%)	Veh Type	No of Intersections					
		1	2	3	4	5	6
0	Guided						
	Unguided	3.45	5.28	7.71	10.69	13.83	17.22
10	Guided	3.36	5.17	7.60	10.55	13.50	16.59
	Unguided	3.42	5.25	7.69	10.67	13.79	17.15
20	Guided	3.32	5.14	7.51	10.40	13.25	16.30
	Unguided	3.45	5.25	7.69	10.67	13.74	17.09
30	Guided	3.47	5.11	7.46	10.27	13.02	16.09
	Unguided	3.38	5.24	7.67	10.64	13.72	17.02
40	Guided	3.41	5.09	7.38	10.14	12.79	15.83
	Unguided	3.30	5.24	7.64	10.57	13.69	16.99
50	Guided	3.46	5.08	7.35	10.02	12.52	15.64
	Unguided	3.27	5.23	7.64	10.52	13.68	16.92
60	Guided	3.49	5.05	7.32	9.95	12.31	15.45
	Unguided	3.44	5.22	7.62	10.47	13.63	16.87
70	Guided	3.46	5.02	7.30	9.91	12.17	15.25
	Unguided	3.41	5.20	7.60	10.36	13.55	16.75
80	Guided	3.38	5.00	7.24	9.84	12.12	15.09
	Unguided	3.41	5.17	7.58	10.33	13.45	16.64
90	Guided	3.44	4.98	7.22	9.78	12.11	14.97
	Unguided	3.45	5.15	7.52	10.24	13.37	16.52
100	Guided	3.40	4.94	7.19	9.74	12.07	14.82
	Unguided						

In Table 6.3, all the time values are expressed in minutes and trip times are shown for both guided and unguided vehicles. From this table, at “*Level of RG=10%*” a guided vehicle that cross 6 intersections and travels 7 km on the travel path requires 16.59 minutes to reach the destination whereas for the same number of intersections an unguided vehicle requires 17.15 minutes to reach the destination. In the following Tables

6.4(a) and 6.4(b), trip time savings from the simulation results given in Table 6.3 are shown for guided vehicles.

Table 6.4 (a): Trip Time Savings for Guided vehicle

Level of RG(%)	Veh Type	No of Intersections					
		1	2	3	4	5	6
0	Guided	0.00	0.00	0.00	0.00	0.00	0.00
10	Guided	5.83	6.67	6.50	7.95	19.66	37.81
20	Guided	8.12	8.67	11.87	16.95	34.59	54.80
30	Guided	-1.20	10.07	15.07	24.84	48.59	67.78
40	Guided	2.77	11.64	20.17	32.73	62.60	74.83
50	Guided	-0.07	12.33	21.61	39.95	78.78	94.82
60	Guided	-2.09	13.84	23.40	43.99	91.35	105.80
70	Guided	-0.48	15.69	24.80	46.66	99.75	117.80
80	Guided	4.60	16.66	28.48	51.06	102.68	127.80
90	Guided	0.72	18.08	29.51	54.62	103.44	134.80
100	Guided	3.09	20.60	31.51	56.99	105.51	143.83

Table 6.4 (b): Trip Time Savings(%) for Guided vehicle

Level of RG(%)	Veh Type	No of Intersections					
		1	2	3	4	5	6
0	Guided	0.00	0.00	0.00	0.00	0.00	0.00
10	Guided	2.81	2.10	1.40	1.24	2.37	3.66
20	Guided	3.92	2.74	2.57	2.64	4.17	5.30
30	Guided	-0.58	3.18	3.26	3.87	5.86	6.56
40	Guided	1.33	3.67	4.36	5.10	7.54	7.24
50	Guided	-0.03	3.89	4.67	6.23	9.49	9.18
60	Guided	-1.01	4.37	5.06	6.86	11.01	10.24
70	Guided	-0.23	4.95	5.36	7.28	12.02	11.40
80	Guided	2.22	5.26	6.15	7.96	12.37	12.37
90	Guided	0.35	5.70	6.38	8.52	12.47	13.05
100	Guided	1.49	6.50	6.81	8.89	12.72	13.92

In Table 6.4(a), the trip time savings with respect to the unguided vehicles as mentioned earlier are shown in seconds and in Table 6.4(b), trip time savings are expressed in percentage. The sample calculations for Table 6.4(a) and Table 6.4(b) are shown below: From table 6.3, at traffic scenario 1, average trip time required for an unguided vehicle that cross 6 intersections on its travel path is 17.22 minutes and at “Level of RG=90%” a guided vehicle that cross same number of intersections requires 14.97 minutes to reach the destination. So, at “Level of RG=90%” the trip time saving is calculated as

$(17.22 - 14.97) * 60 = 134.8$ sec. The value is shown in Table 6.4(a). In Table 6.4(b), the trip time savings for a guided vehicle is expressed as percentage of trip time of unguided vehicle at “*Level of RG=0%*” and is calculated as $(17.22 - 14.97)/17.22 * 100 = 13.05$ which denotes that, at “*Level of RG=90%*” a guided vehicle that cross 6 intersections on its travel path to reach the destination needs 13.05% of less trip time compared to an unguided vehicle at “*Level of RG=0%*”.

In Tables 6.5(a) and 6.5(b), trip time savings from the simulation results given in Table 6.3 are shown for unguided vehicles.

Table 6.5(a): Trip Time Savings for Unguided vehicle

Level of RG(%)	Veh Type	No of Intersections					
		1	2	3	4	5	6
0	Unguided	0.00	0.00	0.00	0.00	0.00	0.00
10	Unguided	1.80	1.63	1.51	0.95	2.55	3.80
20	Unguided	0.47	1.77	1.49	0.99	5.54	7.77
30	Unguided	4.58	2.67	2.50	2.94	6.54	11.58
40	Unguided	9.44	2.63	4.51	6.95	8.31	13.80
50	Unguided	10.82	3.25	4.62	9.91	8.91	17.77
60	Unguided	0.57	3.94	5.50	12.95	12.22	20.77
70	Unguided	2.73	4.63	6.50	19.62	16.51	27.77
80	Unguided	2.86	6.65	7.89	21.61	22.54	34.81
90	Unguided	0.38	7.80	11.50	26.92	27.52	41.81
100	Unguided						

Table 6.5(b): Trip Time Savings (%) for Unguided vehicle

Level of RG(%)	Veh Type	No of Intersections					
		1	2	3	4	5	6
0	Unguided	0.00	0.00	0.00	0.00	0.00	0.00
10	Unguided	0.87	0.51	0.33	0.15	0.31	0.37
20	Unguided	0.23	0.56	0.32	0.15	0.67	0.75
30	Unguided	2.21	0.84	0.54	0.46	0.79	1.12
40	Unguided	4.56	0.83	0.97	1.08	1.00	1.34
50	Unguided	5.22	1.03	1.00	1.55	1.07	1.72
60	Unguided	0.27	1.24	1.19	2.02	1.47	2.01
70	Unguided	1.32	1.46	1.40	3.06	1.99	2.69
80	Unguided	1.38	2.10	1.70	3.37	2.72	3.37
90	Unguided	0.18	2.46	2.49	4.20	3.32	4.05
100	Unguided						

In Table 6.5(a), the trip time savings for unguided vehicles are shown in seconds and in Table 6.5(b), trip time savings are expressed in percentage. The sample calculations for Table 6.5(a) and Table 6.5(b) are shown below:

As mentioned earlier from Table 6.3, at traffic scenario 1 average trip time required for an unguided vehicle that cross 6 intersections on its travel path is 17.22 minutes and at “Level of RG=90%” an unguided vehicle that cross same number of intersections requires 16.52 minutes to reach the destination. So, at “Level of RG=90%” the trip time saving is calculate as $(17.22 - 16.52) * 60 = 41.81$ sec. The value is shown in Table 6.5(a). In Table 6.5(b), the trip time savings for an unguided vehicle is expressed as percentage of trip time of unguided vehicle at “Level of RG=0%” and is calculated as $(17.22 - 16.52)/17.22 * 100 = 4.05$ which denotes that, at “Level of RG=90%” an unguided vehicle that cross 6 intersections on its travel path to reach the destination needs 4.05% of less trip time compared to an unguided vehicle at “Level of RG=0%”.

The variations of trip time savings in percentage for both guided and unguided vehicles using the values from Table 6.4(b) and Table 6.5(b) are shown graphically as follows:

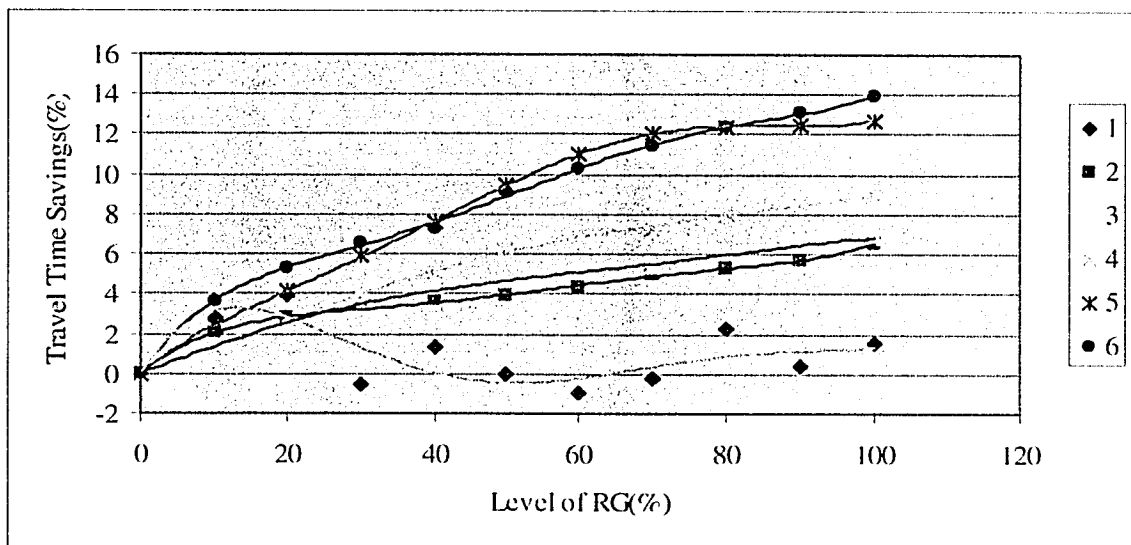


Figure 6.3: Trip Time Savings (%) Variation for Guided vehicles

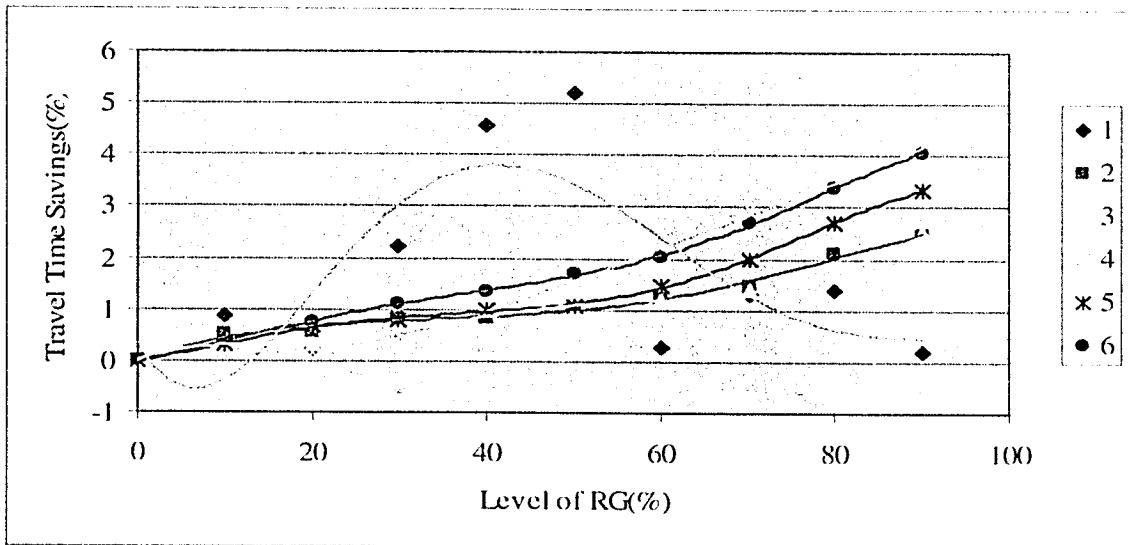


Figure 6.4: Trip Time Savings (%) Variation for Unguided vehicles

In Figures 6.3 and 6.4, the horizontal axis denotes the proportion of guided vehicles and vertical axis denotes savings in trip times expressed in percentage. All the legends shown on the chart denote the number of intersections the vehicle crossed through their travel path. The following interpretations can be made from these figures:

- For the guided vehicles, the savings increase at a higher rate in the initial level of subscriptions of guided vehicles and at a relatively lower rate in the higher level of subscriptions. This happens as in the initial level of RG the few guided vehicles change their routes more frequently to optimize their travel times by avoiding congestion and minimizing the intersection turning delays.
- For the unguided vehicles, the savings increase at a relatively lower rate in the initial level of subscription of guided vehicles and at a relatively higher rate in the higher level of subscriptions. This happens as in the higher level of RG the network traffic flow is optimized to reduce the travel times and unguided vehicles face less congestion and intersection turning delays.

- For a single intersection travel path, there is no influence of guided vehicles as there is a very little chance to improve the trip time for such a short distance. The trip time savings (%) values for single intersection travel path in Table 6.4(b) and Table 6.5(b) and the corresponding plot in Figure 6.3 and Figure 6.4 shows the influence of guided vehicles in trip time savings.
- The trip time savings for unguided vehicles are relatively small compared to the guided vehicles at any level of RG.

6.2.2. Estimated Average Travel Speed

The estimated average travel speeds for all the traffic scenarios are given in Table 6.6.

Table 6.6: Simulation Results for Average Travel Speed

Level of RG (%)	Veh Type	No of Intersections					
		1	2	3	4	5	6
0	Guided	0	0	0	0	0	0
	Unguided	34.74	34.08	31.12	28.07	26.03	24.39
10	Guided	35.75	34.81	31.56	28.42	26.66	25.32
	Unguided	35.05	34.25	31.22	28.11	26.11	24.48
20	Guided	36.16	35.04	31.94	28.83	27.16	25.76
	Unguided	34.82	34.27	31.22	28.12	26.21	24.58
30	Guided	34.54	35.20	32.17	29.20	27.65	26.11
	Unguided	35.53	34.37	31.29	28.20	26.24	24.67
40	Guided	35.21	35.38	32.54	29.58	28.15	26.30
	Unguided	36.40	34.36	31.42	28.38	26.29	24.72
50	Guided	34.73	35.46	32.64	29.94	28.76	26.86
	Unguided	36.66	34.43	31.43	28.51	26.31	24.82
60	Guided	34.40	35.63	32.77	30.14	29.25	27.18
	Unguided	34.84	34.51	31.49	28.65	26.42	24.89
70	Guided	34.66	35.85	32.88	30.28	29.59	27.53
	Unguided	35.21	34.58	31.56	28.96	26.56	25.07
80	Guided	35.53	35.97	33.16	30.50	29.71	27.84
	Unguided	35.23	34.81	31.66	29.05	26.76	25.24
90	Guided	34.86	36.14	33.24	30.69	29.74	28.05
	Unguided	34.81	34.94	31.91	29.30	26.92	25.42
100	Guided	35.27	36.45	33.39	30.81	29.82	28.34
	Unguided	0	0	0	0	0	0

All the values in Table 6.6 are given in km/h. For each traffic scenario, the average travel speed is given for both guided and unguided vehicles. The zero value in the first and last

row of the table denotes the two traffic scenario 1 and traffic scenario 11. The speed values are calculated from the average travel time/km value of Table 6.2. From Table 6.2, at “Level of $RG=10\%$ ” a guided vehicle that cross 6 intersections to reach the destination requires 142.18 seconds to travel one kilometer distance. So, the average speed of the guided vehicle in one link of 1 km is calculated as $(3600/142.58)=25.32$ km/h that is shown in Table 6.6. All other values are in the table are calculated using the same procedure. The variations of the average travel speed are shown graphically as follows:

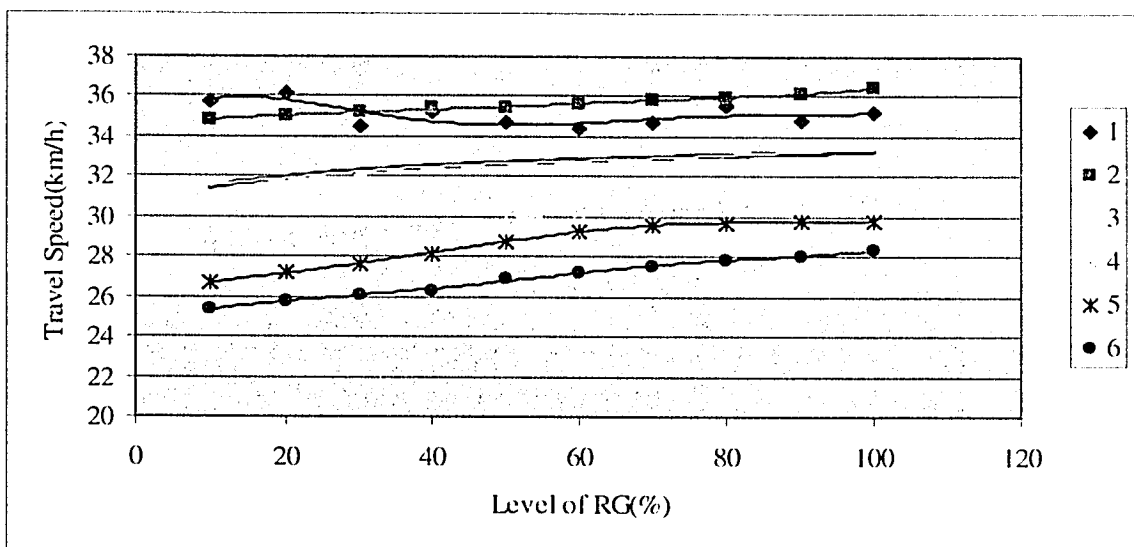


Figure 6.5: Average Travel Speed Variation for Guided Vehicles

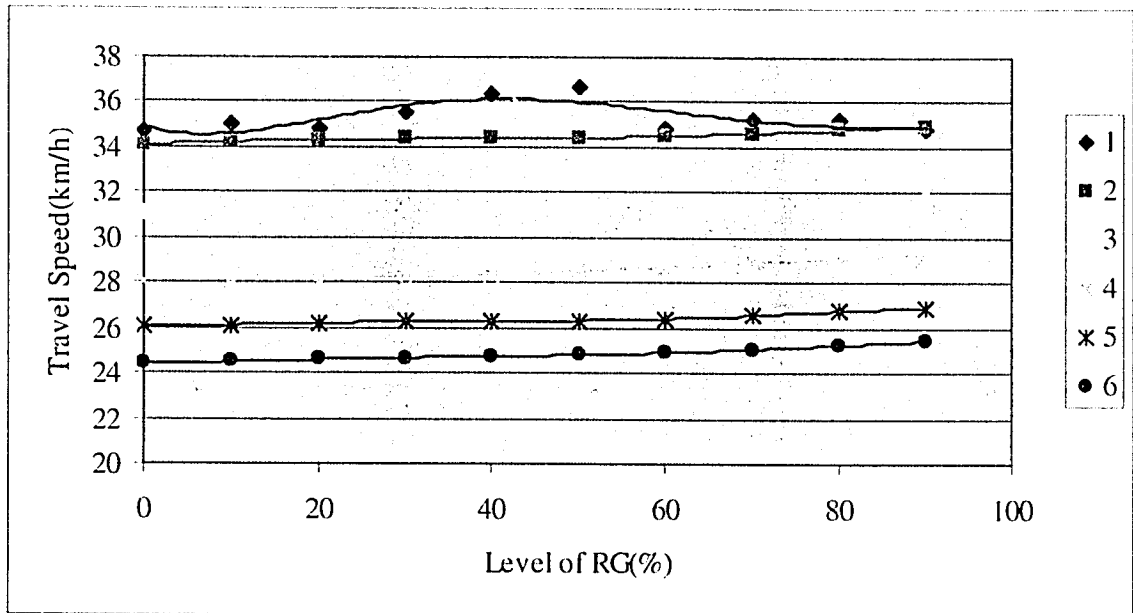


Figure 6.6: Average Travel Speed Variation for Unguided Vehicles

In Figures 6.5 and 6.6, the horizontal axis denotes the proportion of guided vehicles and vertical axis denotes average travel speed expressed in km/h. All the legends shown on the chart denote the number of intersections the vehicle crossed through their travel path. The following interpretations can be made from these figures:

- The average travel speeds for the guided vehicles are always higher than the unguided vehicles for any level of RG.
- For the guided vehicles, the average speed increases at a higher rate in the initial level of RG and at a relatively lower rate in the higher level of RG because of the less congested routes.
- Like the travel time savings plots, for a single intersection travel path there is no influence of guided vehicles as there is a very little chance to improve the travel speed and corresponding travel time for such a short distance.
- For the unguided vehicles, the average travel speed increases at a lower rate in the initial level of RG and at a relatively higher rate in the higher level of RG.

6.3. Discussion

For the analysis, eleven different traffic scenarios are considered with different compositions of guided and unguided vehicles. From the results, it is observed that the benefits of the guided vehicles to reduce the trip time are more for longer trip distance, because for longer distances, vehicles must cross more intersections as the intersections are almost equally spaced in the urban street network. During the initial level of guided vehicle subscription trip time savings rate for guided vehicles is higher as there is a higher possibility to get a new path and thereby improve the total trip time. The percentage of guided vehicles in the vehicle composition also influence the total trip time for the unguided vehicles as the link traffic flow is optimized to increase the overall network traffic flow. So, all vehicles can travel at a higher speed. The link length of the network is larger than the typical link length of an urban network. The larger link length is assumed to get longer travel times, so that trip time savings can be determined as the guided vehicle only checks the network traffic condition while entering into the link. So, the larger link length affects the trip time savings as traffic conditions may change in a way to provide an alternative route with less trip time.

CHAPTER 7

APPLICATION OF TRAFFIC SIMULATION MODEL TO MONTREAL ROAD NETWORK

7.1. Introduction

A selected road network of Montréal city is used for the application of the new algorithm. The selected network is modeled through the traffic simulator developed in this thesis to show the travel time savings of the guided vehicles. The traffic simulator calculates the minimum trip time path with the new algorithm.

7.2. Selected Road Network in Montreal

To simulate the traffic on an urban road network a part of Montréal city road network surrounded by four major arterials has been chosen as a test bed. The selected area surrounded by Rue Jean-Talon, Boulevard Pie-IX, Rue Notre-Dame and Boulevard Decarie, is the central part of city and also a place of major traffic activity. For simplicity, only North-South and East-West major arterials that carry major traffic load have been considered in the simulator. A total of 25 intersections have been identified and each intersection is given a unique ID to distinguish from each other. Intersection identification of the arterials is given in Table 7.1.

Table 7.1: Intersection ID for Montreal Road Network

Road Name	Boul. Decarie	Ch. de la Cote-dés-Neiges	Rue St- Denis	Rue d'Iberville	Boul. Pie-IX
Rue Jean-Talon	1	2	3	4	5
Av. Van Horne	6	7	8	9	10
Ch. De la Cote-Ste-Catherine	11	12	13	14	15
Rue Sherbrooke	16	17	18	19	20
Rue Notre-Dame	21	22	23	24	25

The selected road network area and the surrounding area shown on full size Montréal road map is given in the appendix C. The city road network is coded into links and intersections and the distance of each link is measured. The city road map is used to obtain the distance of each link for the selected network. The coded road network with intersection ID and the individual link distance is shown below:

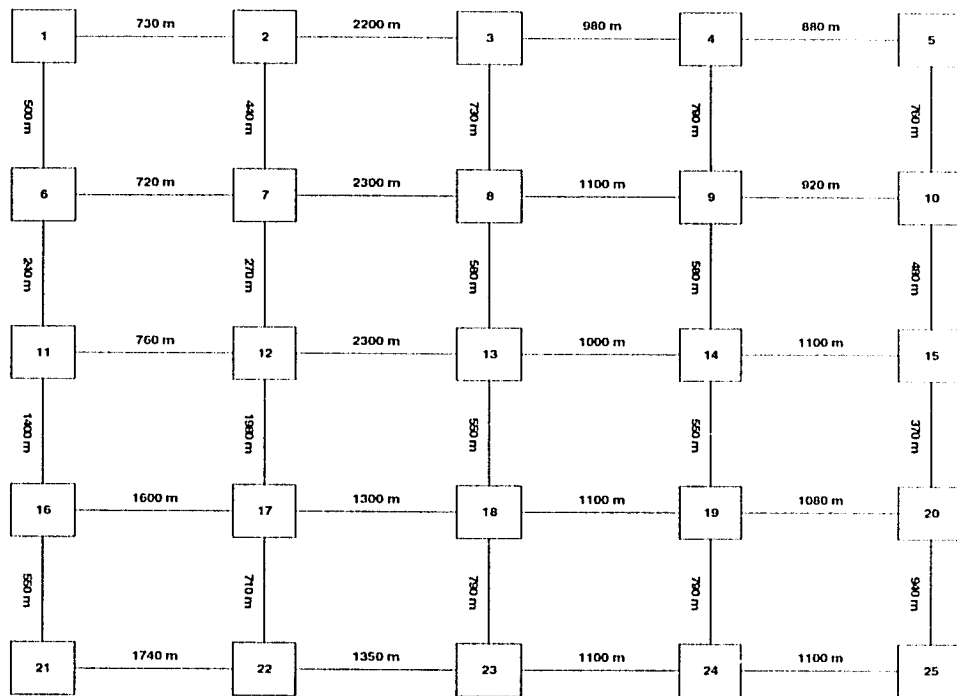


Figure: 7.1: Coded Montreal City Network

Like the hypothetical network, all the peripheral intersections of the network are considered as the traffic zone centriods and vehicles are assumed to enter and exit from the network through these intersections. Since the trip data of Montréal city was not available, the origin destination trip matrix given in chapter 4 is used for trip generation.

7.3. Results of the simulation

From the simulation results for the hypothetical network it is evident that trip time savings for the guided vehicles that use the new dynamic path search algorithm for minimum trip time path calculation is significant for longer trip distance. So, for the

Montréal road network, only the paths having highest number of intersections among all the possible paths of the network are considered to show the simulation results. The same traffic scenario and the same travel demand data as discussed in the previous chapter are used for simulation run.

7.3.1. Trip Time Savings

The origin-destination pair and the intersections of the respective arterials for each path that are considered are shown in Table 7.2.

Table 7.2: Origin Destination for Selected Path

Path	Origin	Destination	Starting Intersection	Ending Intersection
1	1	20	Jean Talon/Decarie	Sherbrooke/Pie-IX
2	1	24	Jean Talon/Decarie	Notre-Dame/d'Iberville
3	4	21	Jean Talon/d'Iberville	Notre-Dame/Decarie
4	10	21	Van Horne/Pie-IX	Notre-Dame/Decarie
5	16	5	Sherbrooke/Decarie	Jean Talon/Pie-IX
6	22	5	Notre-Dame/Cote-des-Neiges	Jean Talon/Pie-IX
7	25	2	Notre-Dame/Pie-IX	Jean Talon/Cote-des-Neiges
8	25	6	Notre-Dame/Pie-IX	Van Horne/Decarie

For each traffic scenario, the average trip time required for a guided vehicle at different “Level of $RG(\%)$ ” between any particular origin-destination pair of Table 7.2 is given in Table 7.3 and all values in the table are expressed in minutes.

Table 7.3: Simulation Results for Trip Time

Path	Origin	Destination	Level of $RG(\%)$										
			0	10	20	30	40	50	60	70	80	90	100
1	1	20	15.61	15.09	14.79	14.49	14.24	14.02	13.97	13.59	13.42	13.31	13.17
2	1	24	16.3	15.97	15.69	15.47	15.2	14.97	14.7	14.57	14.37	14.14	13.93
3	4	21	15.42	15.02	14.73	14.45	14.16	13.95	13.83	13.7	13.6	13.37	13.18
4	10	21	15.8	15.4	15.05	14.7	14.5	14.35	14.26	14.2	14.12	14.08	14.01
5	16	5	16.47	16.06	15.67	15.36	15.02	14.69	14.45	14.27	14.14	14.02	13.9
6	22	5	16.55	16.2	15.8	15.49	15.07	14.82	14.69	14.6	14.47	14.29	14.17
7	25	2	15.97	15.57	15.02	14.82	14.6	14.5	14.42	14.31	14.21	14.1	14.01
8	25	6	15.85	15.42	15.14	14.89	14.69	14.52	14.29	14.04	13.97	13.69	13.59

From Table 7.3, at “Level of $RG=10\%$ ” the trip time required for a guided vehicle to travel between intersection of Rue Jean-Talon/Boul. Decarie and Rue Sherbrooke/Boul. Pie-IX is 15.09 minutes. All other values in the table denote the same interpretation for

different “Level of $RG(\%)$ ”. The trip time savings for a particular path is calculated with respect to trip times found in traffic scenario when there is no guided vehicle in the network at “Level of $RG=0\%$ ” as shown in Table 7.3. The trip time savings results of a guided vehicle for all the paths of Table 7.2 are given in Tables 7.4(a) and 7.4(b).

Table 7.4(a): Simulation Results for Trip Time Savings

Path	Origin	Destination	Level of $RG(\%)$										
			0	10	20	30	40	50	60	70	80	90	100
1	1	20	0	31.5	49.6	67.7	82.6	95.6	98.5	121.5	131.6	138.5	146.4
2	1	24	0	19.58	36.58	49.48	65.88	79.58	95.68	103.5	115.9	129.6	142.2
3	4	21	0	23.9	41.3	57.9	75.8	88.2	95.2	103.2	109.2	123.2	134.2
4	10	21	0	24.03	45	65.99	77.7	86.7	92.6	96.1	100.8	103	107.6
5	16	5	0	24.12	48	66.3	86.7	106.7	120.7	131.7	139.7	146.7	153.8
6	22	5	0	21	45	63.8	88.7	103.7	111.7	116.7	124.8	135.7	142.6
7	25	2	0	23.9	56.9	68.9	81.9	88	92.9	99.8	105.8	111.9	117.3
8	25	6	0	25.25	42.36	57.5	69.4	79.5	93.5	108.6	112.5	129.5	135

Table 7.4(b): Simulation Results for Trip Time Savings(%)

Path	Origin	Destination	Level of $RG(\%)$										
			0	10	20	30	40	50	60	70	80	90	100
1	1	20	0	3.36	5.29	7.23	8.82	10.20	10.51	12.97	14.05	14.78	15.63
2	1	24	0	2.00	3.74	5.06	6.74	8.14	9.79	10.58	11.85	13.25	14.54
3	4	21	0	2.58	4.46	6.26	8.19	9.53	10.29	11.16	11.80	13.32	14.51
4	10	21	0	2.54	4.75	6.96	8.20	9.15	9.77	10.14	10.63	10.87	11.35
5	16	5	0	2.44	4.86	6.71	8.78	10.80	12.22	13.33	14.14	14.85	15.57
6	22	5	0	2.12	4.53	6.43	8.93	10.44	11.25	11.75	12.57	13.67	14.36
7	25	2	0	2.49	5.94	7.19	8.55	9.18	9.70	10.42	11.04	11.68	12.24
8	25	6	0	2.66	4.46	6.05	7.30	8.36	9.83	11.42	11.83	13.62	14.20

The trip time saving values in Table 7.4(a) are shown in seconds and in Table 7.4(b) the savings values are expressed in percentage. An example calculation is shown below:

As found in Table 7.3, average trip time required for an unguided vehicle between intersection of Rue Jean-Talon/Boul. Decarie and Rue Sherbrooke/Boul. Pie-IX is 15.61 minutes. For the same origin-destination pair and at “Level of $RG=90\%$ ” the average trip time required for a guided vehicle is 13.31 minutes. So, at “Level of $RG=90\%$ ” the time savings is calculated as $(15.61 - 13.31) * 60 = 138.5$ seconds. The value is shown in Table 7.4(a). In Table 7.4(b), the trip time savings for a guided vehicle is expressed as

percentage of trip time of unguided vehicle at “Level of $RG=0\%$ ” is calculated as $(15.61 - 13.31)/15.6 * 100 = 14.78\%$ which denotes that, at “Level of $RG=90\%$ ” a guided vehicle traveling between intersection of Rue Jean-Talon/Boul. Decarie and Rue Sherbrooke/Boul. Pie-IX need 14.78% of less trip time compared to an unguided vehicle traveling between the same intersections.

The variation of trip time savings for each of the path with different traffic scenario is shown in Figure 7.2.

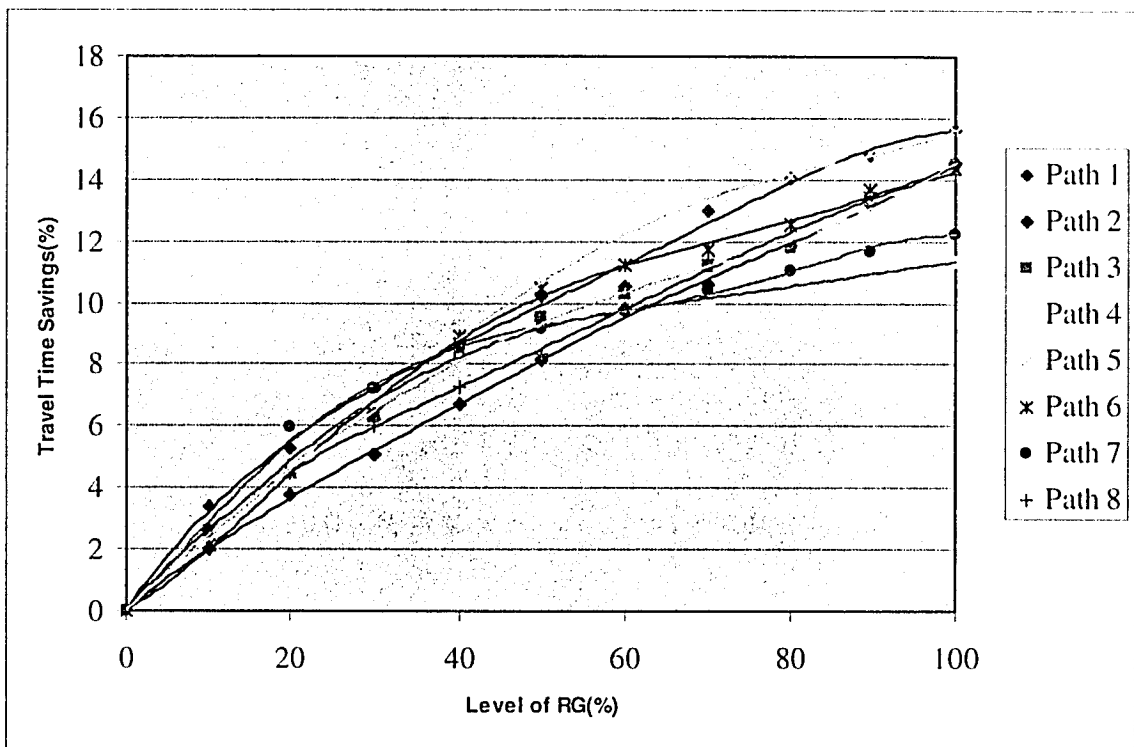


Figure 7.2: Variation of Trip Time savings according to Level of $RG(\%)$

In the above figure, the horizontal axis denotes the proportion of guided vehicles and vertical axis denotes savings in trip times expressed in percentage. All the legends shown on the chart denote different path of Table 7.2. The trip time savings patterns for this network is similar to the pattern found for the hypothetical network and for guided vehicles the trip time savings increases at a higher rate during lower level of subscriptions

of guided vehicles and at a relatively lower rate during the higher level of RG. This happens in the initial level of RG as the few guided vehicles change their routes more frequently to optimize their travel times by avoiding congested routes and minimizing the intersection turning delays.

7.3.2. Minimum Trip Time Path Comparison

To show the improvement of the algorithm in terms of route selection decisions of the guided vehicle, travel path between the following origin-destination pair are considered:

Table 7.5: Origin Destination for Selected Path

Path	Origin		Destination	
	ID	Intersection	ID	Intersection
P1	10	Van Horne/Pie-IX	21	Notre Dame/Decarie
P2	16	Sherbrooke/Decarie	5	Jean Talon/Pie-IX
P3	22	Notre-Dame/Cote-des-neige	5	Jean Talon/Pie-IX

As discussed in the previous chapter vehicles are generated through every origin node by keeping constant time headway, so after a certain simulation time step, total number of vehicles running on the network is same for the same demand data. So, for each of path of Table 7.5 two vehicles that entered into the network approximately at the same traffic volume level and travel between same origin-destination pair are selected for path comparisons. The unguided vehicle select the travel path based on link travel time only and the guided vehicle select the travel path based on link travel time and intersection turning delays. The travel path between the guided and unguided vehicle for each of origin-destination pair of Table 7.5 is compared through predicted and actual trip time. For each vehicle, predicted travel path is calculated during the entrance of the vehicle in the network. The path denotes the shortest travel path to reach the destination based on network traffic condition for that time period. Predicted trip time is the time required to travel on that particular path. Actual travel path is the path that vehicle follows to reach

the destination and actual trip time is time difference of vehicle's trip finish time and trip start time. For unguided vehicle, the predicted and actual travel path is always same as the vehicle does not have the route guidance system to check en-route network traffic information. But actual travel path of the guided vehicle varies considerably from the predicted travel path as the guided vehicle is simulated with the capability to check the en-route network traffic condition. The path comparisons for guided and unguided vehicle are given as follows:

Travel Path P1:

The trip information for guided and unguided vehicle is given in Tables 7.6 and 7.7.

Table 7.6: Trip Information of Unguided Vehicle for Path P1

Link No.	Predicted Travel Path			Actual Travel Path		
	From Node	To Node	Distance(m)	From Node	To Node	Distance(m)
1	10	9	920	10	9	920
2	9	8	1100	9	8	1100
3	8	7	2300	8	7	2300
4	7	12	270	7	12	270
5	12	11	760	12	11	760
6	11	16	1400	11	16	1400
7	16	21	550	16	21	550
Total	-	-	7300	-	-	7300
	Predicted Trip Time(min)			Actual Trip Time(min)		
	10.18			17.08		
	Trip Time Difference(minutes)					
6.9						
Speed(km/h)	43.03			25.64		

Table 7.7: Trip Information of Guided Vehicle for Path P1

Link No.	Predicted Travel Path			Actual Travel Path		
	From Node	To Node	Distance(m)	From Node	To Node	Distance(m)
1	10	9	920	10	9	920
2	9	8	1100	9	14	580
3	8	7	2300	14	19	550
4	7	6	720	19	24	790
5	6	11	240	24	23	1100
6	11	16	1400	23	22	1350
7	16	21	550	22	21	1740
Total	-	-	7230	-	-	7030
	Predicted Trip Time(min)			Actual Trip Time(min)		
	14.21			15.18		
	Trip Time Difference(minutes)					
	0.97					
Speed(km/h)	30.53			27.79		

As shown in Table 7.6, though the predicted path and actual traveled path for the unguided vehicle is same, there is a considerable difference between the predicted and actual trip time as the predicted trip time does not include the intersection delays which is a major part of total trip time. The traveled path contains two left turns, one from Van Horne to Cote-des-Neiges and other one from Cote-Ste-Catherine to Decarie and one right turn from Cote-des-Neiges to Cote-Ste-Catherine. On the other hand, the guided vehicle follows a completely different path with one left turn from Van Horne to Rue d'Iberville and one right turn from Rue d'Iberville to Notre-Dame. The interesting point to be noted for the guided vehicle is that it changes the travel path from the initial predicted path using en-route traffic and intersection turning delay information. Also the predicted trip time is much higher than the unguided vehicle because the guided vehicle always incorporates the intersection turning delays into minimum trip time path calculation. As shown in Table 7.7 the initial path for the guided vehicle contains only one left turn from Van Horne to Decarie but it follows a different path to minimize the trip time and that is why the actual trip time does not vary too much from the predicted

trip time and is also less than the actual trip time of the unguided vehicle. The individual link distance of each path shown in Table 7.6 and Table 7.7 are taken from the measured link distance of Montréal city road network as given in Figure 7.1. The total path distance is the sum of the individual link distances of the path. From Table 7.7, the guided vehicle travels 7230 m and 7030 m on its predicted path and actual path respectively. The average travel speed of the vehicle is calculated from travel path distance and the required trip time to reach the destination. Average travel speed of the guided vehicle on its actual travel path is calculated as follows:

Total distance on actual travel path = 7030 m = 7.03 km

Actual trip time required = 15.18 minutes = 0.252 h

So, average travel speed = $\frac{7.03}{0.252} = 27.79$ km/h

This number is shown in Table 7.7. All other speed values are calculated by following the same procedure. The difference between actual and predicted trip time of the guided vehicle is calculated as follows:

Actual trip time required = 15.18 minutes

Predicted trip time required = 14.21 minutes

So, the difference between the trip time = 15.18 - 14.21 = 0.97 minute = 58.2 seconds.

The number is also shown in Table 7.7. The trip time difference of unguided vehicle is also calculated following the same procedure as shown in Table 7.6.

Travel Path P2:

The trip information for guided and unguided vehicle is given in Tables 7.8 and 7.9.

Table 7.8: Trip Information of Unguided Vehicle for Path P2

Link No.	Predicted Travel Path			Actual Travel Path		
	From Node	To Node	Distance(m)	From Node	To Node	Distance(m)
1	16	17	1600	16	17	1600
2	17	18	1300	17	18	1300
3	18	13	550	18	13	550
4	13	8	580	13	8	580
5	8	9	1100	8	9	1100
6	9	10	920	9	10	920
7	10	5	760	10	5	760
Total	-	-	6810	-	-	6810
	Predicted Trip Time(min)			Actual Trip Time(min)		
	8.17			16		
	Trip Time Difference(minutes)					
	7.83					
Speed(km/h)	50.01			25.54		

Table 7.9: Trip Information of Guided Vehicle for Path P2

Link No.	Predicted Travel Path			Actual Travel Path		
	From Node	To Node	Distance(m)	From Node	To Node	Distance(m)
1	16	17	1600	16	17	1600
2	17	18	1300	17	12	1980
3	18	19	1100	12	7	270
4	19	20	1080	7	2	440
5	20	15	370	2	3	2200
6	15	10	480	3	4	980
7	10	5	760	4	5	880
Total	-	-	6690	-	-	8350
	Predicted Trip Time(min)			Actual Trip Time(min)		
	14.51			14.95		
	Trip Time Difference(minutes)					
	0.44					
Speed(km/h)	27.66			33.51		

As shown in Table 7.8 unguided vehicle follows the path with two left turn, one from Rue Sherbrooke to St-Denis and other one from Av. Van Horne to Boul. Pie-IX. The actual time required to finish the trip is much higher than the predicted trip time. From Table 7.9, the guided vehicle changes the predicted path which consists only of a single left turn between Rue Sherbrooke and Boul. Pie-IX to a path that consists of one left turn

from Rue Sherbrooke to Ch. Cote-des-Neige and one right turn from Ch. Cote-des-Neige to Rue Jean-Talon. The difference between the predicted and actual trip time for the guided vehicle is much less and actual trip time is less than the unguided vehicle. The total path distance in km, average travel speed in km/h and difference between actual and predicted trip time in minutes of the vehicle are as shown in Tables 7.8 and 7.9. They are calculated following the same procedure as discussed earlier for path P1.

Travel Path P3:

The trip information for guided and unguided vehicle is given in Tables 7.10 and 7.11.

Table 7.10: Trip Information of Unguided Vehicle for Path P3

Link No.	Predicted Travel Path			Actual Travel Path		
	From Node	To Node	Distance(m)	From Node	To Node	Distance(m)
1	22	17	710	22	17	710
2	17	18	1300	17	18	1300
3	18	13	550	18	13	550
4	13	14	1000	13	14	1000
5	14	9	580	14	9	580
6	9	10	920	9	10	920
7	10	5	760	10	5	760
Total	-	-	5820	-	-	5820
	Predicted Trip Time(min)			Actual Trip Time(min)		
	8.67			16.01		
	Trip Time Difference(minutes)					
	7.34					
Speed(km/h)	40.28			21.81		

Table 7.11: Trip Information of Guided Vehicle for Path P3

Link No.	Predicted Travel Path			Actual Travel Path		
	From Node	To Node	Distance(m)	From Node	To Node	Distance(m)
1	22	17	710	22	17	710
2	17	12	1980	17	18	1300
3	12	7	270	18	13	550
4	7	2	440	13	8	580
5	2	3	2200	8	3	730
6	3	4	980	3	4	980
7	4	5	880	4	5	880
Total	-	-	7460	-	-	5730
	Predicted Trip Time(min)			Actual Trip Time(min)		
	13.6			14.19		
	Trip Time Difference(minutes)					
	0.59					
Speed(km/h)	32.91			24.23		

As shown in Table 7.10 unguided vehicle follows the path with two right turn one between Ch. Cote-des-Neige and Rue Sherbrooke and other one between Rue d'Iberville and Av. Van Horne and two left turn one between Rue Sherbrooke and St-Denis and other one between Av. Van Horne to Boul. Pie-IX. The actual time required to finish the trip is much higher than the predicted trip time that is similar to path P1 and P2 discussed earlier. From Table 7.11, the guided vehicle changes the initial path that consists only of a single right turn between Ch. Cote-des-Neige and Rue Jean-Talon to a path that consists of one left turn between Rue Sherbrooke and St-Denis and one right turn between St-Denis and Rue Jean-Talon. The required trip time is much less than the actual trip time of the unguided vehicle. The total path distance in km, average travel speed in km/h and difference between actual and predicted trip time in minutes of the vehicle are as shown in Tables 7.10 and 7.11. They are calculated following the same procedure as discussed earlier for path P1 and path P2.

7.4. Discussion

From the simulation results of Montréal road network, it is found that guided vehicles following the travel path using the new algorithm always require less trip time to reach the destination. This reduction in trip time is substantial for a longer trip distance when vehicles need to cross the intersection more frequently. Intersection turning delay varies according to the intersection approach volume and the signal timing cycles. Usually left turning delay is more than the straight and right turning delay. So, the travel path with more left turns requires more time to reach the destination compared to the travel path with more right turns. The new path search algorithm considers the intersection delay along with link travel time while determining the minimum trip time path which is a significant improvement over traditional shortest path algorithm based on travel time on links only. This is very much evident from the three sample paths from Montréal road network where guided vehicles always choose the path with less number of left turns and change their path during the travel period to minimize the trip time if needed. The application of new path search algorithm will be significant if peak hour traffic flow is taken into consideration when the turning delay is much higher due to the residual queue at intersection.

The predicted trip time as defined earlier is based on the travel time and travel speed on the links at the start of the trip. As the trip progresses, the travel time and travel speed on the links vary. So, when the vehicle reaches at destination, the total time on the shortest path is the actual time. The simulation results show that the actual trip times for the guided vehicle are reasonably close to the predicted trip times. The difference between the actual and predicted trip time is less than 1 minute for the guided vehicle whereas the

difference for unguided vehicle is about to 6 to 8 minutes as shown in Tables 7.6 to 7.11. The small difference between actual and predicted trip time of the guided vehicle demonstrates the capability of new algorithm in better trip planning. The actual travel speed for both of the guided and unguided vehicle varies between 20 to 34 km/h as shown in Table 7.6 to Table 7.11 which correspond to the normally expected speeds on urban city road network. As shown in Table 7.9, total distance of the predicted travel path for the guided vehicle is 6690 m whereas the actual distance the vehicle travels on the minimum trip time path is 8350 m. So, minimum trip time path of the guided vehicle is not always the minimum travel distance path. This is because the minimum trip time path is calculated based on traffic volumes on the road links and intersection turning delays.

CHAPTER 8

CONCLUSIONS

8.1. Introduction

This chapter summarizes the major conclusions developed in this thesis, application to ITS and directions of future research in this area.

8.2. Conclusions

The major conclusions of the thesis are as follows:

1. An algorithm to consider both the link travel time and intersection turning delays has been developed to find minimum trip time path between an origin destination pair. The new dynamic shortest path algorithm addresses the algorithm run time issue by combining the heuristic search with bi-directional search that reduce the node search effort and also multi-objective nature of transportation network by considering the intersection turning delays along the link travel time. A macroscopic traffic simulator has been developed to simulate the vehicles that use the new algorithm for route selection. The simulator is based on the cell transmission model that follows a set of state-space equations and is consistent with the fundamental traffic flow theory.
2. The trip time savings for a guided vehicle is more for longer trip distance as the guided vehicle that follows the new algorithm to calculate the minimum time path need to cross more number of intersections to reach the destination. Presence of guided vehicles also influences the unguided vehicle of the network by reducing the trip time as the overall network traffic flow increases. The trip time savings increases at a higher rate for a smaller percentage of guided vehicles because most of the vehicles use the travel path without considering en-route traffic information.

3. Application of the traffic simulation model to Montréal city road network demonstrates the capability of the new algorithm to select the shortest path between an origin-destination pair with minimum intersection turning delays. The simulation results show that the actual trip times for the guided vehicle are reasonably close to the predicted trip time and the difference between the actual and predicted trip is less than 1 minute. The small difference between actual and predicted trip time of the guided vehicle demonstrates the capability of new algorithm in better trip planning. The actual travel speed for both of the guided and unguided vehicle varies between 20 to 34 km/h as shown in Table 7.6 to Table 7.11 in Chapter 7. These travel speed values correspond to normally expected speed on Montréal road. Another interesting finding is to be noted that minimum trip time path of the guided vehicle is not always the minimum travel distance path as shown in the trip information table in Chapter 7. This is because the minimum trip time path is calculated based on traffic volumes on the road links and intersection turning delays.

8.3. Application to Intelligent Transportation System (ITS)

Conventional traffic assignment models are useful as a planning tool but they are not directly applicable in modeling individual routing decision made with the aid of in-vehicle route guidance system. So, the path search algorithm developed in this study has an application in today's dynamic transportation system. Network equipped with on road traffic sensors and GPS tracking for the vehicle position coordinates can supply the real time traffic data to the in-vehicle route guidance system. This Bi-objective Bidirectional Heuristic path search algorithm will generate the dynamic optimum route for the driver for a chosen origin-destination pair. With the minor modifications the algorithm can be

implemented as a part of a centralized traffic management system where based on the network traffic condition Traffic Management Center (TMC) would be able to predict the minimum trip time path between all origin-destination nodes for all departure times. So, the proposed algorithm can assist drivers in better trip planning, decision-making on departure time, route selection and congestion avoidance.

8.4. Future Research

While the runtime of this algorithm was not the primary concern no tests were performed for complexity analysis. But for a dynamic route guidance system all path finding algorithms need to be very fast so, the algorithm needs to be tested on a larger network to measure the computational efficiency. In this thesis the algorithm is proposed only to calculate the one-to-one shortest path for a single departure time. But the algorithm needs to be implemented to calculate one-to-all shortest paths for all departure times. Microscopic traffic characteristics need to be considered into the traffic simulation model to analyze the route guidance efficiency of guided vehicles in case extreme traffic conditions like incidents, queue at intersection, spillback etc.

REFERENCE

1. Adler, J.L.(1998) Best Neighbor Heuristic Search for Finding Minimum Paths in Transportation Networks. Transportation Research Record 1651, Paper no 98-0877, pp. 48-52.
2. Ahuja, R. K., T. L Magnanti, and J. B. Orlin (1993). Network Flows: Theory, Algorithms, and Applications, Prentice Hall, Englewood Cliffs, NJ.
3. Beasley, J.E., and N. Christofides (1989). An Algorithm for the Resource Constrained Shortest Path Problem, Networks 19, pp. 379.
4. Bellman, R (1958). On a Routing Problem. Quart. Appl. Mathematics, Vol. 16, pp. 87-90.
5. Bertsekas, D., and R. Gallager (1992). Data Networks. Second Edition, Prentice Hall. Englewood Cliffs, N. J.
6. Blue, V.J., Adler, J.L. and G.F. List (1997). Real-Time Multiple Objective Path Search for In-Vehicle Route Guidance Systems. Transportation Research Record 1588, pp. 10-17.
7. Bottom, J., et al (1999). Investigation of route guidance generation issues by simulation with DynaMIT, *in* A. Ceder (ed.), Transportation and Traffic Theory. Proceedings of the 14th International Symposium on Transportation and Traffic Theory, Pergamon, pp. 577-600.
8. Bureau of Public Roads (1964). Traffic Assignment Manual, U.S. Department of Commerce, U.S. Government Printing Office, Washington, DC.

9. Casturi, R. (2000), A Macroscopic Model for Evaluating the Impact of Emergency Vehicle Signal Preemption on Traffic. M.Sc. thesis submitted to Virginia Polytechnic Institute and State University.
10. Center for Transportation Research (1993). Development and Testing of Dynamic Traffic Assignment and Simulation Procedures for AITS/ATMS Applications. Technical Report DTFH6 1-90-R-00074-FG, The University of Texas at Austin, Austin, Texas.
11. Chabini, I (1997). A New Short path Algorithm for Discrete Dynamic Networks. Proceedings of the 8th IFAC Symposium on Transport Systems, Chania, Greece, June 16-17, 551-556.
12. Chabini, I (1998). Discrete Dynamic Shortest Path Problems in Transportation Applications: Complexity and Algorithm with Optimal Run Time. Transportation Research Record 1645, Paper no 98-1150, pp. 170-175.
13. Chabini, I., and M. Abou Zeid (2003). The Minimum Cost Flow Problem in Capacitated Dynamic Networks. Submitted to the TRB 82nd Annual Meeting, Washington, D.C.
14. Chen, J., T. Wang, and E. Oh (2003). Adaptive Routing: Centralized versus Distributed, Technical Report, Department of Computer Science, Texas A&M University, College Station.
15. Cherkassky, B. V., A. V. Goldberg,, and T. Radzik (1993). Shortest Paths Algorithms: Theory and Experimental Evaluation. Technical Report 93-1480, Computer Science Department, Stanford University.

16. Daganzo, C.F. (1994) The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory, *Transportation Research*, Vol. 28B, No. 4, pp. 269-287.
17. Dean, B (1999). Continuous-time dynamic shortest path algorithms. Master of Engineering Thesis, Massachusetts Institute of Technology
18. Dechampeaux, D. and L. Sint (1977). An improved Bidirectional Heuristic Search Algorithm. *Journal of ACM*, Volume 24, Issue 2, page 177-191.
19. Dijkstra, E. W (1959). A Note on Two Problems in Connexion with Graphs, *Journal of Numerical Mathematics*, Vol. 1, pp. 269-271.
20. Dreyfus, S. (1969). "An appraisal of some shortest-path algorithms". *Operations Research* 17, 395-412
21. Garber, N. J. and L. A. Hoel (2001), *Traffic and Highway Engineering (Revised 3rd edition)*, PWS Publishing.
22. Golledge, R (1993). Geographical Perspective on Spatial Cognition. In *Behavior and Environment: Psychological and Geographical Approaches* (T. Garling and R. Golledge, eds.), Elsevier Science Publishers, pp. 16-82
23. Gragopoulos, I., E. Papapetrou, F. Pavlidou (2000). Performance Study of Adaptive Routing Algorithms for LEO Satellite Constellations under Self-Similar and Poisson Traffic, *Space Communications*, Vol. 16, pp. 15-22
24. Hart, P., N. Nilsson, and B. Raphael (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transportation System Science and Cybernetics*, Vol. 4, pp. 100-107.

25. Harvey, J. M., and S. Shaw (2001). *Geographic Information Systems for Transportation. Principles and Applications*, Oxford University Press.
26. Henig, I. H (1985). The shortest path problem with two objective functions, *European Journal of Operation Research* 25, pp. 281.
27. Intelligent Vehicle Highway System. A Synopsis, IVHS Roundtable, Transportation Association of Canada, June 1992
28. Kaufmann, D.E. and R.L. Smith (1993). Fastest paths in time-dependent networks for intelligent vehicle highway systems application. *IVHS Journal* 1, pp. 1-11.
29. Kaufmann, D.E., R.L. Smith, and K.E. Wunderlich (1991). An Iterative Routing/Assignment Method for Anticipatory Real-Time Route Guidance, *SAE Vehicle Navigation and Information Systems Conference Proceedings*, P-253, 701-708.
30. Li. L(1998). *Java: Data Structures and Programming*. Springer-Verlag.
31. Maccubbin R. P., Barbara L. Staples, and Michael R. Mercer (2003). *Intelligent Transportation Systems Benefits and Costs*, United States Department of Transportation, Report no: FHWA-OP-03-075
32. Martins, E. Q. V (1984). On a Multicriteria Shortest Path Problem. *European Journal of Operation Research* 16, pp. 236.
33. Modesti. P. and A. Schiomachen (1998). A Utility Measure for Finding Multiobjective Shortest Paths in Urban Multimodal Transportation networks. *European Journal of Operation Research* 111, pp. 495-508.
34. Nelson, P. C. and A. A. Topsis (1992). Unidirectional and Bidirectional Search Algorithms. *IEEE Software*, Volume 9, Issue 2, p.p 77-83

35. Orda, A. and R. Rom (1990). Shortest-path and minimum-delay algorithms in network with time dependent edge length. *Journal of the ACM* 37, pp. 607-625.
36. Orda, A. and R. Rom (1991). Minimum weight paths in time-dependent network. *Networks* 21, pp. 295-320.
37. Papacostas, C. S. and P. D. Prevedouros (2001), *Transportation Engineering and Planning* (3rd edition), Prentice Hall, Upper Saddle River, NJ.
38. Pearsons J. (1998) *Heuristic Search in Route Finding*, Master's Thesis. University of Auckland.
39. Pohl, I (1969). *Bidirectional Heuristic Search in Path Problem*. Stanford University, Stanford, Calif.
40. Pollotino, S., and M. G. Scutella (1998). Shortest Path Algorithms in Transportation Models: Classical and Innovative Aspects. In (P. Marcotte, and S. Nguyen, eds.) *Equilibrium and Advanced Transportation Modelling*, Kluwer, pp. 245-281.
41. Rodrigue, J-P et al. (2004) *Transport Geography on the Web*. Hofstra University, Department of Economics & Geography, <http://people.hofstra.edu/geotrans>
42. Subramanian, S (1997). *Routing Algorithms for Dynamic, Intelligent Transportation Networks*, M.S.C. thesis submitted to Virginia Polytechnic Institute and State University.
43. *The National Intelligent Transportation Systems Architecture* (1998). U.S. Department of Transportation, Version. 2.0, Washington, D.C.
44. Webster, F. V (1958). *Traffic Signal Setting*, Report 39, Road Research Laboratory, Crowthorne, Berkshire, England.

45. Zhan, F.B (1997). Three Fastest Shortest Path Algorithms on Real Road Networks: Data Structures and Procedures. *Journal of Geographic Information and Decision Analysis*, Vol. 1, No. 1, pp. 69-82
46. Zhao, Y (1997). *Vehicle Location and Navigation Systems*. Artech House, Norwood, MA.
47. Zillaskopoulos, A. K., and H. S. Mahmassani (1993). Time dependent Shortest Path Algorithm for Real-time Intelligent Vehicle Highway System Applications. *Transportation Research Record 1408*, TRB, National Research Council, Washington, D.C., pp. 94-104.

APPENDIX A

Network Generation Data File

General Remarks

The following data files from Table A.1 to Table A.4 are used to build the network topology for both of the hypothetical network and selected Montreal city road Network. During the loading phase of the simulation run, these data files are used to generate the connectivity between the Link and the Cell object using network coding approach discussed in chapter 4. In Table A.1, "Out Degree" refers to the total number of outgoing links from the corresponding Node object. In Table A.2, "In Degree" refers to total number of incoming links to the corresponding Node object. Table A.3 and Table A.4 are discussed in chapter 4.

Table A.1. Node Outgoing Link Data file

Node	North	East	South	West	Out Degree
1	Null	2	6	Null	2
2	Null	3	7	Null	2
3	Null	4	8	Null	2
4	Null	5	9	Null	2
5	Null	Null	Null	Null	0
6	Null	7	11	Null	2
7	2	8	12	6	4
8	3	9	13	7	4
9	4	10	14	8	4
10	5	Null	Null	9	2
11	Null	12	16	Null	2
12	7	13	17	11	4
13	8	14	18	12	4
14	9	15	19	13	4
15	10	Null	Null	14	2
16	Null	17	21	Null	2
17	12	18	22	16	4
18	13	19	23	17	4
19	14	20	24	18	4
20	15	Null	Null	19	2
21	Null	Null	Null	Null	0
22	17	Null	Null	21	2
23	18	Null	Null	22	2
24	19	Null	Null	23	2
25	20	Null	Null	24	2

Table A.2. Node Incoming Link Data file

Node	North	East	South	West	In Degree
1	Null	Null	Null	Null	0
2	Null	Null	7	1	2
3	Null	Null	8	2	2
4	Null	Null	9	3	2
5	Null	Null	10	4	2
6	1	7	Null	Null	2
7	2	8	12	6	4
8	3	9	13	7	4
9	4	10	14	8	4
10	Null	Null	15	9	2
11	6	12	Null	Null	2
12	7	13	17	11	4
13	8	14	18	12	4
14	9	15	19	13	4
15	Null	Null	20	14	2
16	11	17	Null	Null	2
17	12	18	22	16	4
18	13	19	23	17	4
19	14	20	24	18	4
20	Null	Null	25	19	2
21	16	22	Null	Null	2
22	17	23	Null	Null	2
23	18	24	Null	Null	2
24	19	25	Null	Null	2
25	Null	Null	Null	Null	0

Table A.3. Link-Link Connectivity Data File

Link	Head	Tail	Left Turn	Straight	Right Turn
1	1	2	Null	3	4
2	1	6	9	10	Null
3	2	3	Null	5	6
4	2	7	12	13	14
5	3	4	Null	7	8
6	3	8	16	17	18
7	4	5	Null	Null	Null
8	4	9	20	21	22
9	6	7	11	12	13
10	6	11	25	26	Null
11	7	2	Null	Null	3
12	7	8	15	16	17
13	7	12	28	29	30
14	7	6	10	Null	Null
15	8	3	Null	Null	5
16	8	9	19	20	21
17	8	13	32	33	34
18	8	7	13	14	11
19	9	4	Null	Null	7
20	9	10	23	Null	Null
21	9	14	36	37	38
22	9	8	17	18	15
23	10	5	Null	Null	Null
24	10	9	21	22	19
25	11	12	27	28	29
26	11	16	41	42	Null
27	12	7	14	11	12
28	12	13	31	32	33
29	12	17	44	45	46
30	12	11	26	Null	Null
31	13	8	18	15	16
32	13	14	35	36	37
33	13	18	48	49	50
34	13	12	29	30	27
35	14	9	22	19	20
36	14	15	39	Null	Null
37	14	19	52	53	54
38	14	13	33	34	31
39	15	10	24	23	Null
40	15	14	37	38	35
41	16	17	43	44	45
42	16	21	Null	Null	Null
43	17	12	30	27	28
44	17	18	47	48	49
45	17	22	Null	Null	58
46	17	16	42	Null	Null
47	18	13	34	31	32
48	18	19	51	52	53
49	18	23	Null	Null	60
50	18	17	45	46	43
51	19	14	38	35	36
52	19	20	55	Null	Null
53	19	24	Null	Null	62
54	19	18	49	50	47
55	20	15	40	39	Null
56	20	19	53	54	51
57	22	17	46	43	44
58	22	21	Null	Null	Null
59	23	18	50	47	48
60	23	22	Null	58	57
61	24	19	54	51	52
62	24	23	Null	60	59
63	25	20	56	55	Null
64	25	24	Null	62	61

Table A.4. Link-Node Connectivity Data File

Link	Head	Tail	Left Turn	Straight Turn	Right Turn
1	1	2	Null	3	7
2	1	6	7	11	Null
3	2	3	Null	4	8
4	2	7	8	12	6
5	3	4	Null	5	9
6	3	8	9	13	7
7	4	5	Null	Null	Null
8	4	9	10	14	8
9	6	7	2	8	12
10	6	11	12	16	0
11	7	2	Null	Null	3
12	7	8	3	9	13
13	7	12	13	17	11
14	7	6	11	Null	Null
15	8	3	Null	Null	4
16	8	9	4	10	14
17	8	13	14	18	12
18	8	7	12	6	2
19	9	4	Null	Null	5
20	9	10	5	Null	Null
21	9	14	15	19	13
22	9	8	13	7	3
23	10	5	Null	Null	Null
24	10	9	14	8	4
25	11	12	7	13	17
26	11	16	17	21	Null
27	12	7	6	2	8
28	12	13	8	14	18
29	12	17	18	22	16
30	12	11	16	Null	Null
31	13	8	7	3	9
32	13	14	9	15	19
33	13	18	19	23	17
34	13	12	17	11	7
35	14	9	8	4	10
36	14	15	10	Null	Null
37	14	19	20	24	18
38	14	13	18	12	8
39	15	10	9	5	Null
40	15	14	19	13	9
41	16	17	12	18	22
42	16	21	Null	Null	Null
43	17	12	11	7	13
44	17	18	13	19	23
45	17	22	Null	Null	21
46	17	16	21	Null	Null
47	18	13	12	8	14
48	18	19	14	20	24
49	18	23	Null	Null	22
50	18	17	22	16	12
51	19	14	13	9	15
52	19	20	15	Null	Null
53	19	24	Null	Null	23
54	19	18	23	17	13
55	20	15	14	10	Null
56	20	19	24	18	14
57	22	17	16	12	18
58	22	21	Null	Null	Null
59	23	18	17	13	19
60	23	22	Null	21	17
61	24	19	18	14	20
62	24	23	Null	22	18
63	25	20	19	15	Null
64	25	24	Null	23	19

APPENDIX B

PrintOut of the Program Code

General Remarks

The computer programs for the traffic simulator developed in this thesis are implemented in JAVA™. All the programs are written and compiled against Sun® Microsystem's J2SE version 1.4 java libraries using Borland® JBuilder® 2005. Java package 'roadsim' for the simulator contains total 21 java source files. The total number lines of the program in all 21 java source files are 7,293. So, it is not possible to show the complete program. Only the objects that are defined to develop the traffic simulator as discussed in chapter 5 are selected to show the sample program. The value shown in the bracket associated with each java source file in the following sections represents total number of lines in the corresponding file. But only selected program lines are shown.

B.1: Program Code for Cell.java File (Total 167 lines)

```
package roadsim;
import java.util.*;

public class Cell extends Vector{
    Link belongs_to;
    RoadCanvas network;
    int ID,link_id;
    double Length;
    double density; // stores the density of this Cell
    double right_density,left_density,straight_density;
    double speed; // stores average speed of this Cell
    double running_Spd_min;
    double ST_speed, LT_speed, RT_speed;// stores average ST,LT,RT speed of this Cell
    double right_L_speed,left_L_speed;
    double travel_time; // stores Cell travel time depending on the speed
    double last_pos; // stores the last vehicle position in the cell
    double T; // available movement time " applicable for head cell only"
    int type; // Define Cell type: "0"=Link exit cell/first Cell,
    // "1"=Transfer Cell; "2"= Link entry Cell
```



```

Cell next_Cell; // next Cell in the Link towards the traffic direction
Cell next_LT,next_RT; // next Cell in the Left or Right Link for the Head Cell
boolean next = false; // decides if the current Cell is the end of the Network path
boolean prev = false; // decides if the current Cell the beginning of the Network path
boolean Left_outflow_chk,Right_outflow_chk;
Cell prev_Cell; // previous Cell in the Link toward the reverse direction
Cell prev_LT,prev_RT;// previous Left and Right Cell for incoming flow
boolean LT_check = false; // check if the LT outflow is already determined
boolean RT_check = false; // check if the RT outflow is already determined
Cell parent_Cell;
Cell Right_T;
Cell Left_T;
Cell Straight_T;
double max_inflow,min_outflow; /* calculate the number of inflow or outflow vehicle
within simulation time step for the current Cell*/
int pos_out_critical,pos_lt_out_critical,pos_rt_out_critical; /* determine the
no of vehicle that can satisfy the cell position criteria that is (cell.length
-2>pos>v_Length)*/

int diff_out_flow; /* diff in min out flow param from capacity and position analysis*/
double max_LT_in, max_RT_in; // max no of vehicle can take the entry from left
// and right direction
double flow; // flow from the current properties in veh/h for cell type 1/2
double ST_flow, LT_flow, RT_flow; // flow from the current properties in veh/h

double lt_max_inflow,lt_min_outflow,rt_max_inflow,rt_min_outflow; /* the number of
inflow or outflow vehicle
within simulation time step for the head current Cell to the left and right link*/

double veh_entry; // no of st veh entry from previous Cell
double lt_veh_entry,rt_veh_entry; // no of lt/rt veh entry from previous Cell

int LT_count,ST_count,RT_count; // stores total no of Left,Straight and Right
// vehicle into the Cell

Vector OUT; // temporarily stores the outgoing Vehicle
Vector LT_OUT, RT_OUT; // temporarily stores the LT & RT outgoing Vehicle

// variable for min_out_flow//
////////////////////
int veh_no_crossing; /* no of vehicles crossing the Cell in sampling period T
using the avg Cell speed. will be effective in the next time step.*/
int veh_no_LT_crossing,veh_no_RT_crossing,veh_no_ST_crossing; /* no of
//vehicles going out to Left,Right n Straight direction from the current Cell*/

```

```

int max_veh; // maximum no of vehicles can be in the Cell
int max_possible_veh; // maximum no of possible vehicle depending on the no of
// vehicles that already in the cell
int max_veh_entrance; // max no of vehicles can cross the current Cell using
// Cell avg speed

double flow_balance; // stores the fraction veh part to add it with the later
// time step "flow_balance" to make vehicle flow according to Cell capacity
// (only straight dir)

double inflow_balance; // stors the fraction veh part to add it with the later
// time step "inflow_balance" to make vehicle inflow in a whole number

// constructor
double lt_flow_bal,rt_flow_bal;// stores the fraction veh part to add it with
// the later time step lt/rt "flow_bal" to make vehicle flow according to Cell
// capacity(only left and right dir)

double lt_inflow_bal,rt_inflow_bal;// stores the fraction veh part to add it with
// the later time step lt/rt "inflow_bal" to make vehicle flow according to Cell
// capacity(only left and right dir)
public Cell(Link to_whom,int id,double length) {
    belongs_to = to_whom;
    ID = id;
    Length = length;
    link_id = belongs_to.ID;
    network = belongs_to.network;
    OUT = new Vector();
    LT_OUT = new Vector();
    RT_OUT = new Vector();
}
public double Min(int cell_veh,double max_in,double max_next_in,int max_next,
    int max_crossing){
    double[] sort_flow = new double[5];
    double temp;
    sort_flow[0] = (double)cell_veh;
    sort_flow[1] = max_in;
    sort_flow[2] = max_next_in;
    sort_flow[3] = (double)max_next;
    sort_flow[4] = (double)max_crossing;
    for(int i=0;i<sort_flow.length;i++){
        //System.out.println(sort_flow[i]+"\\t");
    }
    for(int i=0;i<sort_flow.length;i++){
        for(int j=0;j<sort_flow.length-i-1;j++){
            if(sort_flow[j]>sort_flow[j+1]){

```

```

        temp = sort_flow[j];
        sort_flow[j] = sort_flow[j+1];
        sort_flow[j+1] = temp;
    }
}
}
//System.out.println("sorted value....."+"\\r\\n");
for(int i=0;i<sort_flow.length;i++){
    //System.out.print(sort_flow[i]+"\\t");
}
return sort_flow[0];
}

```

C.2: Program Code for Link.java File (Total 217 lines)

```

package roadsim;
import java.awt.*;
import java.util.*;
public class Link extends Vector implements Constants{
    /*******
    ****General variable****
    *****/
    Intersection head = null;
    Intersection tail = null;
    //Car vehicle;
    RoadCanvas network;
    boolean headLight;
    boolean ActiveLight; // to determine the signal is started or not
    // start and finish time for green and red signal
    long green_Start_Time,green_End_Time,red_End_Time,red_Start_Time;
    double next_timestp_greentime,next_timestp_redtime,next_timestp_move_time;
    // remaining green, red and init lag time for the link
    long Rem_Green_period,Rem_Red_period,Rem_init_lag;
    int Green_time_stp_counter, Red_time_stp_counter;
    // signal color
    Color Signalcolor;
    Vector vehiclequeue; // for waiting vehicle to get into the network
    Vector generatedqueue; // for generating vehicle to get into the vehiclequeue
    /*******
    ****Network connectivity variable****
    *****/
    int ID; // unique id for each link
    Link ST;
    int Straight; // straight link id
    Link LT;

```

```

int Left; // left link id
Link RT;
int Right; // right link id
double Length; // length of the link in meter
int cell_no;
Cell current_cell; // always stores task performing current cell
Intersection ST_N,LT_N,RT_N;
/*****
****Link properties variable****
*****/
String Name; // Name of the link
Vector Link_Vehicles; // stores all the vehicles on the link
//int link_car_total; // total number of vehicles in the link
int leading_veh_cell_id; // stores the Cell ID of the leading Car in the link
int available_from; // stores the cell ID after which random vehicle can be
// inserted for the particular link
boolean availability; // checks if random vehicle insertion is possible
/*****
****Link characteristics variable****
*****/
int no_of_lane; // no of lane in the Link
double density; // traffic density veh/km
double j_density; // jam density for this Link veh/km
double Capacity; // Link capacity veh/h
double FFS; // free flow speed of the Link
double TTime; // Link travel time cumulative of all the Cell travel time
double last_veh_pos; // stores the position of the last vehicle in the link
double LT_Delay,ST_Delay,RT_Delay; // turning delay of the link
/*****
****Link Output variable****
*****/
int no_of_veh_passed;
//boolean green_Active;
// constructors
public Link(int id, int left_id, int straight_id, int right_id) {
    ID = id;
    Straight = straight_id;
    Left = left_id;
    Right = right_id;
}
public Link(int id){
    ID = id;
}
public Link(int id, Intersection T, Intersection H, double len){
    ID = id;
    head = H;

```

```

    tail = T;
    Length = len;
    no_of_lane = 2;
    Link_Vehicles = new Vector();
}
public void Cell_arrange(Link currnt_link,double length){
    int no;
    double CELL_L =Math.round ((currnt_link.FFS* 1000*currnt_link.network.T)/3600);
    double remainder = length%CELL_L;
    if(remainder>0){
        no = (int)((length-remainder)/CELL_L);
    }else{
        no = (int)(length/CELL_L);
    }
    this.cell_no = no; // stores the number of cells for the current Link
    for(int i=0;i<no;i++){
        if(i == 0){
            // for the first cell the length might be higher than the regular Cell
            // length
            this.addCell(new Cell(this,i+1,CELL_L+remainder));
            currennt_cell = (Cell) this.elementAt(i);
            //System.out.println("Link: "+currnt_link.ID+" cell: "+currennt_cell.ID+
            //" len: "+currennt_cell.Length);
        }else{
            this.addCell(new Cell(this,i+1,CELL_L));
            currennt_cell = (Cell) this.elementAt(i);
            //System.out.println("Link: "+currnt_link.ID+" cell: "+currennt_cell.ID+
            //" len: "+currennt_cell.Length);
        }
    }
}
}

```

C.3: Program Code for Intersection.java File (Total 80 lines)

```

package roadsim;
import java.awt.*;
import java.util.*;

public class Intersection implements Constants,Comparable{
    int ID; // unique id for each intersection
    // location coordinate of the intersection
    Point loc = new Point();
    int outnum,innum; // number of incoming and outgoing link from the node
    int mir_outnum, mir_innum; // miirror of the previous variables to be used in
    // the algorithm
    int k_path_innum,k_path_outnum,k_path_innum_dummy,k_path_outnum_dummy;
    int[] outgoing; // stores outgoing node
    int[] incoming; // stores incoming node
}

```

```

int[] outgoing_K_path; // stores outgoing node info for K_path iteration
int[] incoming_K_path; // stores incoming node info for K_path iteration
int[] outgoing_k_path_dummy;
int[] incoming_k_path_dummy;
// stores surrounding intersections in a clockwise fashion starting from North
Intersection North,East,South,West;
// stores outgoing Links in a clockwise fashion starting from North
Link North_O,East_O,South_O,West_O;
//Link North_L,East_L,South_L,West_L;
//stores incoming Links in a clockwise fashion starting from North
Link North_I,East_I,South_I,West_I;
//Link North_B,East_B,South_B,West_B;
int type; // 1 = entry node, 2 = entry and exit node, 3 = exit node, 4 = transfer node
RoadCanvas network;
TrafficLight light;
int Signalcount; // no of times signal changes
int CycleCount; // no of times signal runs
Color NorthSouth,EastWest;
long init_lag; // initial lag time for signal starting time
long Green_phase = 35;
long Red_phase = 25;
long LightPeriod;
long period; // traffic generator interval period
Vector VehicleQueue; // for waiting vehicle to get into the network
Map Reached_destination;
public Intersection(int id) {
    ID = id;
    outgoing = new int[NIGHB_SIZE];
    outgoing_K_path = new int[NIGHB_SIZE];
    outgoing_k_path_dummy = new int[NIGHB_SIZE];
    incoming = new int[NIGHB_SIZE];
    incoming_K_path = new int[NIGHB_SIZE];
    incoming_k_path_dummy = new int[NIGHB_SIZE];
    for(int i = 0;i<NIGHB_SIZE;i++){
        outgoing[i] = 0;
        outnum = 0;
        incoming[i] = 0;
        innum = 0;
    }
    light = new TrafficLight(this,1);
    Reached_destination = new HashMap();
}
/* Compare two cities by name. */
public int compareTo(Object n){
    return compareTo((Intersection) n);
}

```

```

    public int compareTo(Intersection c){
        return this.ID - c.ID;
    }
}

```

C 4: Program Code for Car.java File (Total 228 lines)

```

package roadsim;
import java.util.*;
import java.lang.*;

/*****
 ***General variable*****/
 *****/
RoadCanvas network;
MacroModel model;
RouteEngine PathDoctor;
Random ran = new Random();
double ran_dir;
/*****
 ***Network variable*****/
 *****/
int ID; // unique car id
Link belongs_to; // denotes passing Link
Cell Running_cell; // current running Cell
Cell next_Cell; // next cell on the current Link
int Running_Cell_ID; // current running Cell id
int Cell_order; // stores the position in the Cell
Link last; // denotes passed link
Link next; // denotes the next link that vehicle will take the entry
Intersection leaving_from, going_towards, Origin, Destination;
boolean link_end = false; // check if the car reaches at the end of the link
/*****
 ***Update variable*****/
 *****/
Intersection next_intersection;
boolean reach_intersection; /* event trigger variable for each vehicle when
that reach the intersection*/
double new_pos,dummy_new_pos; /* relative position of the car in the link measured
from
the tail coordinate of that particular link for the next time step.*/
double old_pos; /* relative position of the car in the link measured from
the tail coordinate of that particular link for the current time step.*/
double last_stp_pos; /* the position of the car at the end the of last time
step*/
double Rem_Celldist_cross_time; // time taken by the vehicle to cross the
// remaining distance

```

```

double Rem_cross_time_ana; // will be used turning calculation
double Next_Cell_balance_time; // remaining time from the Sampling period that
// will determine the vehicle position in the next Cell
boolean dir_decision = false; // checks if the direction is already decided
String Turning;
fWriter info_file;
/*****
****Vehicle variable****
*****/
double headway;
Car leading_Car; // leading Car of this one
Car following_Car; // following Car of this one
int type; // Define Car type: "1"=Leading, "2"=Following
int category; // "0"=random; "1"=mid-level; "2"=high-level
double trip_start_time; // stores the trip start time for the vehicle
double trip_finish_time; // stores the trip finish time for the vehicle
double dist_traveled; // stores the total distance the vehicle traveled
double trip_Time;
double link_TTime;
double Total_link_time;
/*****
****Path variable****
*****/
Path Init_Path; // initial expected route between origin and destination
Path temp_Route; // keeps track all the temp route checked frm
Path Traveled_path; // traveled path from origin to destination
Path Path_to_follow; // stores the new path calculated from each intersection
// each intersection
Vector Path; // keeps track the travel path
Vector Link_TT; // stroes travel path link crossing time
// constructor
public Car(Link curr_L,Cell cur_cell,int id) {
    belongs_to = curr_L;
    ID = id;
    Traveled_path = new Path();
    Traveled_path.addElement(curr_L.tail);
    //Path = new Vector();
    //Path.addElement(curr_L);
    Running_cell = cur_cell;
    Running_Cell_ID = cur_cell.ID;
    this.network = belongs_to.network;
    this.PathDoctor = new RouteEngine(this.network,this);
    Link_TT = new Vector();
}
public Car(Intersection origin,Intersection destination,int id){
    Origin = origin;

```



```

Destination = destination;
ID = id;
Traveled_path = new Path();
Traveled_path.addElement(origin);
Path_to_follow = new Path();
info_file = new FileWriter(Integer.toString(
this.ID)+".txt",Vehicle_Path_Heading);
Link_TT = new Vector();
}

```

C.5: Program Code for TrafficLight.java File (Total 315 lines)

```

package roadsim;
// import required lib
import java.awt.*;
import java.util.*;
import java.util.Timer;
import java.util.Random.*;
import java.awt.Graphics;
public class TrafficLight implements Constants{
    int Type; // determine the signal type (moderate/heavy)
    int Cycle; // cycle timings
    int GREEN,RED,AMBER;
    Intersection belongs_to; // intersection to which this is belongs to
    Vector Intersections,Links;
    Link curr_L;
    int North_B,South_B,East_B,West_B; // incoming link direction
    // Ex. --North_B>>> towards North direction
    boolean active; // activate the light
    boolean greenState,redState; // state of the trafficligh for the Link
    Timer Light;
    boolean lightColor;
    Light_scheduler Light_set;
    Random generateColor = new Random();
    int ran_num;
    public TrafficLight(Intersection belongs_to,int type) {
        belongs_to = belongs_to;
        Type = type;
        if(Type == 1)
            Cycle = MODERATE;
        else
            Cycle = HEAVY;
        Light = new Timer();
        Light_set = new Light_scheduler();
    }
    public void initializeLight(){
        ran_num = generateColor.nextInt(10)%2;
    }
}

```

```
if(ran_num==0){
    lightColor = true;
}else{
    lightColor = false;
}
//System.out.println(ran_num);
if(lightColor){
    belongs_to.NorthSouth = Color.green;
    if(belongs_to.North_I != null){
        belongs_to.North_I.Signalcolor = belongs_to.NorthSouth;
        belongs_to.North_I.headLight = true;
        belongs_to.North_I.ActiveLight = false;
    }
    if(belongs_to.South_I != null){
        belongs_to.South_I.Signalcolor = belongs_to.NorthSouth;
        belongs_to.South_I.headLight = true;
        belongs_to.South_I.ActiveLight = false;
    }
    belongs_to.EastWest = Color.red;
    if(belongs_to.East_I != null){
        belongs_to.East_I.Signalcolor = belongs_to.EastWest;
        belongs_to.East_I.headLight = false;
        belongs_to.East_I.ActiveLight = false;
    }
    if(belongs_to.West_I != null){
        belongs_to.West_I.Signalcolor = belongs_to.EastWest;
        belongs_to.West_I.headLight = false;
        belongs_to.West_I.ActiveLight = false;
    }
}else{
    belongs_to.NorthSouth = Color.red;
    if(belongs_to.North_I != null){
        belongs_to.North_I.Signalcolor = belongs_to.NorthSouth;
        belongs_to.North_I.headLight = false;
        belongs_to.North_I.ActiveLight = false;
    }
    if(belongs_to.South_I != null){
        belongs_to.South_I.Signalcolor = belongs_to.NorthSouth;
        belongs_to.South_I.headLight = false;
        belongs_to.South_I.ActiveLight = false;
    }
    belongs_to.EastWest = Color.green;
    if(belongs_to.East_I != null){
        belongs_to.East_I.Signalcolor = belongs_to.EastWest;
        belongs_to.East_I.headLight = true;
        belongs_to.East_I.ActiveLight = false;
    }
}
```

```

    }
    if(belongs_to.West_I != null){
        belongs_to.West_I.Signalcolor = belongs_to.EastWest;
        belongs_to.West_I.headLight = true;
        belongs_to.West_I.ActiveLight = false;
    }
}
}
}

```

C.6: Program Code for TrafficGenerator.java File (Total 212 lines)

```

package roadsim;
import java.util.*;
public class Traffic_Generator extends Timer implements Constants{
    RoadCanvas network;
    boolean car_unguided; // chk whether the number of unguided vehicles is
// inserted already
    boolean car_guided; // chk whether the number of guided vehicles is inserted
// already .
    Random ran = new Random();
    double ran_decide;
    long delay;
    int no_guided_veh,no_unguided_veh,tno_guided_veh,tno_unguided_veh,counter;
    int next_car_type;
    int Prod_trip;
    Intersection origin,destination;
    Traffic_Generator_task generating_task;
    Traffic_Generator(Intersection Start, Intersection End, int trips,RoadCanvas child) {
        network = child;
        Prod_trip = trips;
        if(Prod_trip != 0){
            delay = 3600/trips;
        }else{
            delay = 0;
        }
        origin = Start;
        destination = End;
        generating_task = new Traffic_Generator_task(origin,destination,this);
        //System.out.println("Origin: "+origin.ID+" Destination: "+destination.ID+
        //" delay: "+ delay);
    }
}

```

C.7: Program Code for MacroModel.java File (Total 601 lines)

```

package roadsim;
import java.util.*;
import java.lang.*;

```

```

public class MacroModel implements Constants{
    // define class variable
    RoadCanvas belongs_to;
    Random rgen = new Random();
    int init_car_num,link_id;
    int link_cell_total; // stores total number of cells in the link
    int link_car_total; // stores current no of cars into the link
    Link Running_link; // the link which particular car belongs to
    Cell Current_cell;
    int car[][] = new int[200][2];
    Car Vehicle;

    public MacroModel(RoadCanvas network) {
        belongs_to = network;
    }
    public void generate_init_veh(){
        init_car_num = belongs_to.LINKNUM*3;
        // changing the car[][] array size according to the init_car_num variable

        car = (int[][]).resizeArray(car,init_car_num);
        //System.out.println("links: "+belongs_to.LINKNUM+" init: "+init_car_num+" car
"+car.length);
        for(int i=0;i<init_car_num;i++){
            // col 1 stores car ID, col 2 stores the link ID on which the
            // vehicle running in
            car[i][0] = i+1;
            //System.out.println(i);
            link_id = rgen.nextInt(belongs_to.LINKNUM)+1;
            car[i][1] = link_id;
            //System.out.println("link id: "+car[i][1]);
            Running_link = (Link) belongs_to.links.elementAt(link_id-1);
            link_cell_total = Running_link.size();
            //System.out.println(Running_link.ID+"st: "+Running_link.Straight);
            if(Running_link.Link_Vehicles.size()==0){
                Running_link.availability = next_cell(0,link_cell_total,Running_link);
            }else{
                Running_link.availability = next_cell(Running_link.available_from,
                    link_cell_total,Running_link);
            }
        }
        //System.out.println("link: "+Running_link.ID+" next cell: "+
            //Running_link.available_from);
        if(Running_link.availability){
            Current_cell = (Cell) Running_link.elementAt
                (Running_link.available_from);
            this.belongs_to.Vehicles.addElement(new Car(Running_link,Current_cell,
                this.belongs_to.Vehicles.size()+1));
        }
    }
}

```



```

double travel_time;
double manuver_complexity;
double trip_quality;
ArrayList Bookmark_Info;
int ID;
// constructor
public Path(Intersection origin, Intersection destination) {
    Origin = origin;
    Destination = destination;
    Bookmark_Info = new ArrayList();
}
public Path(){
    try {
        jbInit();
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}
}
/* Compare two paths by attributes. */
public int compareTo(Object n){
    return compareTo((Path) n);
}

private void jbInit() throws Exception {
}
/*public int compareTo(Path c){
    return this.ID - c.ID;
}*/
}

```

C.9: Program Code for RoadCanvas.java File (Total 2878 lines)

```

package roadsim;

import java.awt.*;
import java.io.*;
import java.util.*;
import java.util.Timer;
import java.util.List;
import java.awt.Graphics;
import java.lang.*;
public class RoadCanvas extends Canvas implements Constants{
    Roadframe parent;
    /**double buffering*****/
    Image offscreen;

```

```

Dimension offscreenSize;
Graphics offgraphics;

/*****read-write*****/
fReader readNode; // read intersection coordinate
fReader readLink; // read link coordinate
fReader readFS; // read FS link description
fReader readOut;
fReader readIn;
fReader read_OD;
fWriter writeFSL;

/*****write output file*****/
fWriter write_Vehicles;
fWriter write_Signals;
fWriter write_pos; // test
fWriter write_veh_path;
fWriter write_OutPut_TTime;
/*****storing network elements*****/
Vector intersections;// = new Vector(); // intersections vector
Vector links;// = new Vector(); // links vector
Vector Vehicles; // holds all the vehicle in the network
Vector Reached_Vehicles;
Vector traffic_generators;
Map File_writers;
Map Origin_Destination;
Map Vehicle_Depot;

/*****network drawing variable*****/
boolean new_Net = false;
Intersection curr_n = null; // node for which identifying the surrounding
Intersection neigh_n = null; // surroundings of the curr_n
int curr_n_id,neigh_n_id, direction;
boolean drawlink = false;
boolean linkout = false;
boolean linkin = false;
boolean roadpaint = false;
boolean drawLights = false;
boolean init_Lights;
boolean update_lights = false;
boolean HEADLIGHT, ACTIVELIGHT;
final int T = Sim_Time_Stp; // Sampling Period
int NODENUM,LINKNUM;
int linkcounter = 0;
int TOTAL_VEHICLE;
Link curr_l = null;

```

```

MacroModel model;
Car vehicle; // scanning properties for individual Car obj
Cell segment; // scanning properties for individual Cell obj
RouteEngine Pathfinder;
RouteEngine_Back Path_back;
Random ran = new Random();
Timer scheduler; // Main timer for the simulation
Sim_scheduler scheduler_task; // stores simulation TimerTask class
Timer GenerateTraffic; // Main timer for vehicle generation into the network
//Trafficgenerator traffic_generator_task; // stores the scheduling task class for
// traffic generation

int[][] OD_Matrix;
int K; // stores simulation time step

long[] Steptimebank; // stores starting time of each time step in millsec
long init_vehicle_start_time; // starting time for all the init_veh
long currenttimebound; // stores the time in millsec of the last time step
long GREEN_Start,RED_start,GREEN_end,RED_end;
private final Comparator Node_ID_Comparator = new Comparator(){
    public int compare(Object left, Object right){
        Intersection n1 = (Intersection)left;
        Intersection n2 = (Intersection)right;
        int id_Left = n1.ID;
        int id_Right = n2.ID;
        if (id_Left > id_Right){
            return +1;
        } else if (id_Left < id_Right){
            return -1;
        }
        else { // equal
            //System.out.println("comparing.....");
            return n1.compareTo(n2);
        }
    }
};
// the Node_Set of peripheral nodes to input vehicle
private final SortedSet North_West_Set;
private final SortedSet South_West_Set;
List O_D_Input_List;
List North_West;
List South_East;
Vector GeneratorLink;

double gap=0; // gap between the vehicles in the cell
Car last_vehicle; // the last vehicle in the cell
double last_pos; // position of the last vehicle in the cell

```



```

double temp_pos;
public RoadCanvas(Roadframe child) {
    parent = child;
    this.setBackground(Color.darkGray);
    intersections = new Vector();
    links = new Vector();
    Vehicles = new Vector();
    Reached_Vehicles = new Vector();
    traffic_generators = new Vector();
    model = new MacroModel(this);
    Pathfinder = new RouteEngine(this);
    Path_back = new RouteEngine_Back(this);
    scheduler = new Timer();
    GenerateTraffic = new Timer();
    ran = new Random();
    Steptimebank = new long[T*Total_Time_Stp];
    schedular_task = new Sim_scheduler();
    North_West = new ArrayList();
    North_West_Set = new TreeSet(Node_ID_Comparator);
    South_East = new ArrayList();
    South_West_Set = new TreeSet(Node_ID_Comparator);
    O_D_Input_List = new ArrayList();
    read_OD = new fReader(F_OD_Matrix,this);
    GeneratorLink = new Vector();
    //File_writers = new HashMap();
    Vehicle_Depot = new HashMap();
    Origin_Destination = new HashMap();
    writeFSL = new fWriter(FLinkFile,fsfHeading);
    write_Vehicles = new fWriter(F_vehicles_output,VehiclesHeading);
    write_Signals = new fWriter(F_signal_output,SignalHeading);
    write_pos = new fWriter(F_veh_pos,Vehicles_pos_Heading);
    write_OutPut_TTime = new fWriter(FOutPut_TTime,OutPut_TTime_Heading);
}

```

C.10: Program Code for RouteEngine.java File (Total 681 lines)

```

package roadsim;
import java.util.*;
import java.util.List;
import java.io.*;
/**
public class RouteEngine implements Constants{
    RoadCanvas parent;
    Car belongs_to;
    // the set of settled nodes(CLOSE LIST), the nodes whose shortest distances
    // from the source have been found
    int K_path; // the shortest path no

```

```

Vector Path_bank_time = new Vector(); // all the possible path stored
Vector Path_bank_complexity = new Vector();
Path Travel_path;
private final Set CLOSENodes_F = new HashSet(); // forward dir
private final Set CLOSENodes_B = new HashSet(); // backward dir
private final Set CLOSENodes_com = new HashSet();
private final Vector BFS_Clsoe_F = new Vector();
private final Vector BFS_Clsoe_B = new Vector();
private final Vector BFS_Open_F = new Vector();
private final Vector BFS_Open_B = new Vector();
// best estimate of the shortest time from the source in forward dir
private final Map shortestTimes_F = new HashMap();
// best estimate of the least manuver complexity from the source in backward dir
private final Map shortestTimes_B = new HashMap();
private final Map LeastComplexity = new HashMap();
// @stores the best heuristic shortest time to reach to destination node
// from each node
private final Map heuristic_shortestTimes_F = new HashMap();
// @stores the best heuristic shortest time to reach to destination node
// from each node
private final Map heuristic_shortestTimes_B = new HashMap();
// previous node on the shortest path from the source in forward dir
private final Map predecessors_F = new HashMap();
// previous node on the shortest path from the source in backward dir
private final Map predecessors_B = new HashMap();
// stores the prodecessor of each node on the shortest path from the source
private final Map prodecessors = new HashMap();
/*This comparator orders nodes according to their shortest ttime,
in ascending fashion.*/
private Path RunBidirectional(Intersection origin, Intersection destination){
    Path tempPath = new Path();
    Path lowest = new Path();
    Intersection Node_eva;
    //Algorithm = algorithm; // define the Algorithm type
    Path Bidirectional_Path = new Path();
    boolean forpass = false;
    boolean backpass = false;
    Origin_F = origin;
    Destination_F = destination;
    Origin_B = destination;
    Destination_B = origin;
    initialize_F(Origin_F, Destination_F);
    initialize_B(Origin_B, Destination_B);
    Intersection forward = origin;
    Intersection backward = destination;
    lowest.clear();

```

```

out:
while((forward = extractMin_F())!=null){
    //System.out.println(((Intersection)forward).ID);
    markSettled_F(forward);
    relaxneighbours_Forward(forward);
    if(CLOSENodes_F.contains(forward)&& CLOSENodes_B.contains(forward)){
        forpass = true;
        break out;
    }
    //System.out.println("adding...");
in:
while((backward = extractMin_B())!= null){
    //System.out.println(((Intersection)backward).ID);
    markSettled_B(backward);
    relaxneighbours_Backward(backward);
    //System.out.println("breaking.....");
    if(CLOSENodes_B.contains(backward) && CLOSENodes_F.contains(backward)){
        backpass = true;
        break out;
    }
    break in;
}
}
if(forpass){
    tempPath.clear();
    tempPath.travel_time = 0;
    tempPath.addAll(getShortestpath_For(forward));
    tempPath.travel_time += getShortestTime_F(forward);
    tempPath.addAll(getShortestpath_back(forward));
    tempPath.remove(forward);
    tempPath.travel_time += getShortestTime_B(forward);
    // adding initial path to the path bank
    lowest = tempPath;
    //System.out.println(lowest.travel_time);
    //Bidirectional_Path_Bank.add(tempPath);
    // check if there is anyother shortest path
    for(Iterator i = CLOSENodes_F.iterator();i.hasNext();){
        tempPath = new Path();
        Node_eva =(Intersection) i.next();
        tempPath.addAll(getShortestpath_For(Node_eva));
        tempPath.travel_time += round(getShortestTime_F(Node_eva),2);
        tempPath.addAll(getShortestpath_back(Node_eva));
        tempPath.remove(Node_eva);
        tempPath.travel_time += round(getShortestTime_B(Node_eva),2);
        //System.out.println("old: "+lowest.travel_time+" new: "+tempPath.travel_time);
        //Bidirectional_Path_Bank.add(tempPath);
    }
}

```

```

    if(round(lowest.travel_time,2)>round(tempPath.travel_time,2)){
        lowest = tempPath;
    }
}
/*System.out.println("new travel time_F: "+lowest.travel_time);
for(int i=0;i<lowest.size();i++){
    Intersection temp = (Intersection)lowest.elementAt(i);
    System.out.println("node: "+temp.ID);
}*/
}else{
tempPath.clear();
tempPath.travel_time = 0;
tempPath.addAll(getShortestpath_For(backward));
tempPath.travel_time += getShortestTime_F(forward);
tempPath.addAll(getShortestpath_back(forward));
tempPath.remove(forward);
tempPath.travel_time += getShortestTime_B(backward);
// adding initial path to the path bank
lowest = tempPath;
//System.out.println(lowest.travel_time);
//Bidirectional_Path_Bank.add(tempPath);
// check if there is anyother shortest path
for(Iterator i = CLOSENodes_B.iterator();i.hasNext();){
    tempPath = new Path();
    Node_eva =(Intersection) i.next();
    tempPath.addAll(getShortestpath_For(Node_eva));
    tempPath.travel_time += round(getShortestTime_F(Node_eva),2);
    tempPath.addAll(getShortestpath_back(Node_eva));
    tempPath.remove(Node_eva);
    tempPath.travel_time += round(getShortestTime_B(Node_eva),2);
    //Bidirectional_Path_Bank.add(tempPath);
    if(lowest.travel_time>tempPath.travel_time){
        lowest = tempPath;
    }
}
/*System.out.println("new travel time_B: "+lowest.travel_time);
for(int i=0;i<lowest.size();i++){
    Intersection temp = (Intersection)lowest.elementAt(i);
    System.out.println("node: "+temp.ID);
}*/
}
K_path+=1;
if(K_path == 1){
// add all the nodes of the best path for k_shortest path evaluation
//System.out.println("adding.....");
Collections.reverse(lowest);

```

```

    Path_evaluate.addAll(lowest);
    Collections.reverse(lowest);
}
//System.out.println("pinku"+tempPath.travel_time);
//for(int i=0;i<Bidirectional_Path.size();i++){
//System.out.println(((Intersection)Bidirectional_Path.elementAt(i)).ID);
//}
/*for(int i = 0;i<parent.intersections.size();i++){
    Intersection temp = (Intersection)parent.intersections.elementAt(i);
    System.out.println(temp.ID+ " "+ temp.mir_innum);
}*/
//Collections.reverse(lowest);
//System.out.println("k_path: "+K_path+" "+ lowest.travel_time);
return lowest;
}
private Path RunDijkstra(Intersection origin,Intersection destination){
    Origin_F = origin;
    Destination_F = destination;
    /*if(Path_Exists_F(origin,destination)){
        System.out.println("Possible.....");
    }else{
        System.out.println("ImPossible.....");
    }*/
    if(destination.k_path_innum_dummy != 0){
        initialize_F(origin,destination);
        // the current node
        Intersection i;
        while ((i = extractMin_F()) != null){
            // destination reached, stop
            //assert !isSettled(i);
            if (i == destination) {
                //System.out.println("destination reached!!!!!!!!");
                break;
            }
            markSettled_F(i);
            relaxneighbours_Forward(i);
        }
        K_path+=1;
        return getShortestpath_For(destination);
    }else{
        //System.out.println("returning null...");
        return null;
    }
}
}

```

APPENDIX C

Full Size Montreal Map Showing Selected Road Network

General Remarks

The Montréal city road network as discussed in Chapter 7 is given in this appendix. The selected road network and the surrounding area of the network are taken from the full size Montréal road map. All the arterials selected for the network as given in Table 7.1 in Chapter 7 are shown on the map. All the numbers shown on the attached map correspond to the intersection ID in Table 7.1 from the same chapter. Each of the link length between the selected arterials shown in Figure 7.1 in Chapter 7 is measured from the attached map using map scale.

NOTE TO USERS

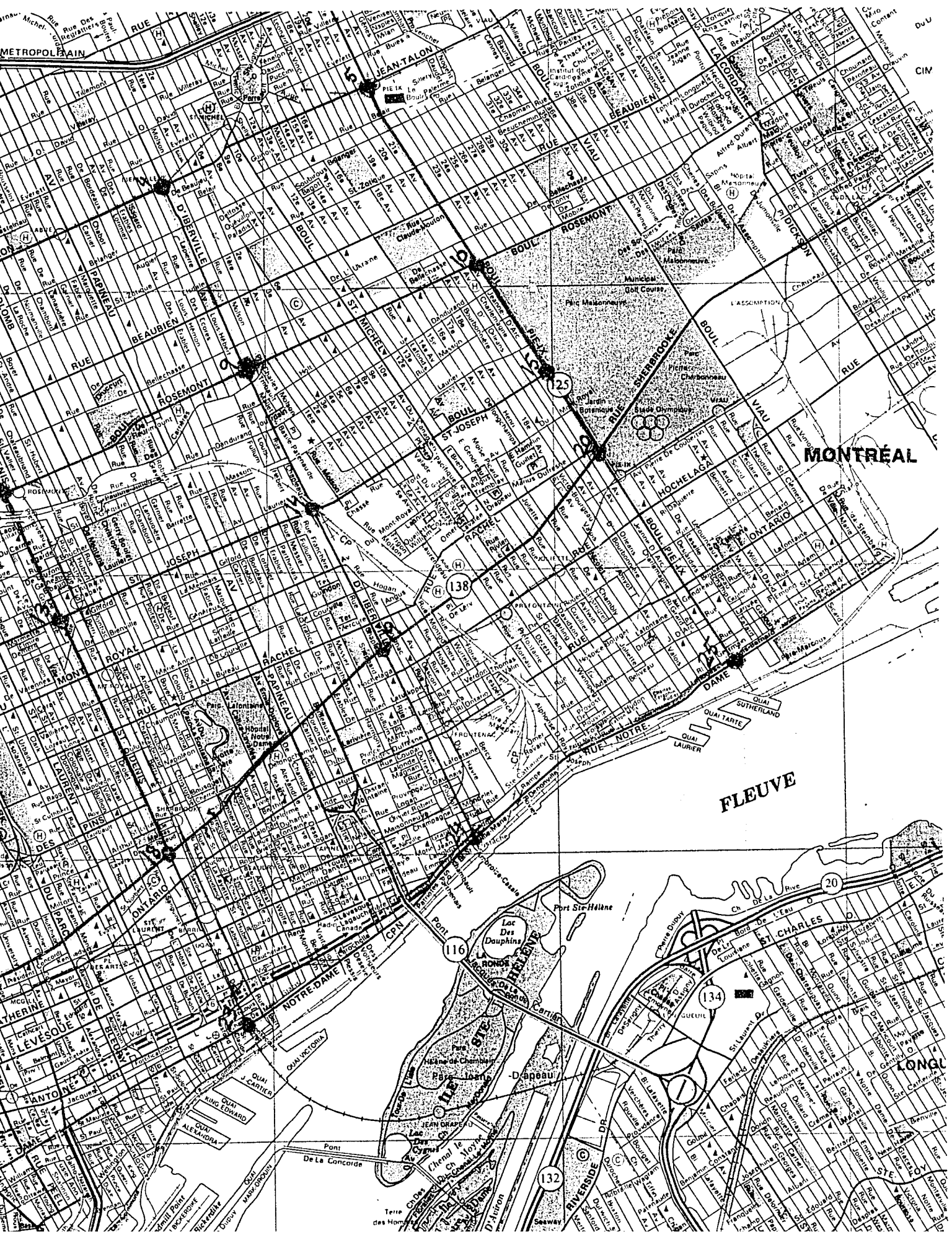
Oversize maps and charts are microfilmed in sections in the following manner:

LEFT TO RIGHT, TOP TO BOTTOM, WITH SMALL OVERLAPS

This reproduction is the best copy available.

UMI





MONTREAL

FLEUVE

116

132

134

20

138

125

Map containing numerous street names and neighborhood labels such as: METROPOLITAIN, JEAN-TALON, BEAUVIEN, ROSEMONT, ST-JOSEPH, ST-HUBERT, ST-JACQUES, LONGUE-POINTE, FLEUVE, and various boulevards and streets.



