

DRAFT

Revised 08/2001

**User's Guide
for
MITSIMLab
and
Road Network Editor (RNE)**

March 2001

Massachusetts Institute of Technology
Intelligent Transportation Systems Program

1. Introduction

This document instructs the user on running:

- MITSIMLab
- Road Network Editor

1.1 Running Instructions for MITSIMLab

MITSIMLab requires the following input files:

- ✓ `Master.mitsim`

The `master.mitsim` file is used to define parameters that control the overall simulation. Some of the parameters in this file are input and output directories, input files, output files to be generated and simulation time.

- ✓ `Master.tms`

The `master.tms` file is used to define the input files that contain all traffic management system data including signal timings and signal control logic and parameters. Some of the input files specified in the `master.tms` file include the `ctrlpara.dat` and `ctrllogic.dat` files described below.

- ✓ `Master.smc`

The `master.smc` file identifies the `master.mitsim` and `master.tms` files and links them together for simulation.

- ✓ `Paralib.dat`

The `paralib.dat` file (refer to Appendix A) is used to define the parameters used in MITSIMLab's simulation models.

- ✓ `Ctrlpara.dat`

The `ctrlpara.dat` file contains execution and simulation parameters related to the traffic management simulator component of MITSIMLab.

- ✓ `Ctrllogic.dat`

The `ctrllogic.dat` file contains parameter definitions related to the incident response logic.

- ✓ Network Database

The Network Database contains the physical representation of the network. This includes the network system consisting of nodes, links, segments and lanes, the sensors and the allowable movements.

✓ Demand file

The demand file contains time-varying origin-destination flows.

✓ Path File

The path file lists the probable paths between each origin-destination pair.

The main output files from MITSIMLab include:

➤ **moe.out**

The **moe.out** file contains maximum head speed, maximum tail speed and maximum gap within lanes.

➤ **segstats.out**

The **segstats.out** file contains segment-specific data from the simulation, for each output time interval. The data includes traffic counts at segment entry and exit, average segment density, speed and travel time. The file also contains speeds and densities on each lane within the segments.

➤ **segtime.out**

The **segtime.out** file contains segment-specific flows and travel times.

➤ **lft_i.out**

The **lft_i.out** file contains link-specific flows and travel times.

➤ **linktime.out**

The **linktime.out** file contains link-specific travel times.

➤ **vehicle.out**

The **vehicle.out** file contains data pertaining to origin, destination, departure and arrival times, mileage and speed of vehicles that have reached their destinations.

➤ **trajectory.out**

The `trajectory.out` file contains the positions of all vehicles on the network. This data helps reconstruct vehicle paths.

➤ `pathrec.out`

The `pathrec.out` file contains link entry and exit data.

➤ `dep.out`

The `dep.out` file contains the departure record of each vehicle in the simulation. This comprises of departure time, vehicle ID, origin, destination, vehicle type and path.

➤ `queue.out`

The `queue.out` file contains queue lengths observed during the simulation.

➤ `sensor.out`

The `sensor.out` file[s] contains the traffic counts registered by the sensors.

➤ `vrc.out`

The `vrc.out` file contains vehicle information as recorded by the point-to-point sensors. Each record corresponds to a single vehicle that has activated such a sensor.

Running MITSIMLab

The following steps lead to the execution of MITSIMLab:

Step 1: Prepare the data files.

This involves the creation of the various input files, such as `master.mitsim`, `network.dat` and `demand.dat` files.

Step 2: Change to the Input file directory.

Step 3: Execute MITSIMLab.

MITSIMLab can be used in two modes: with and without traffic control.

To execute MITSIM alone (i.e. without traffic control), enter the following command:

```
xmitsim -m master.mitsim
```

Complete the following steps to execute MITSIMLab (i.e. with traffic control):

- Run PVM:

- Remove the existing PVM files by typing the following:

```
rm -fr /tmp/pvm*
```

- Execute PVM:

```
pvm ~/.pvm_hosts
```

- Verify that the host server has been added to the PVM repository by typing the following:

```
conf
```

- Push PVM to the background:

```
Quit
```

- Execute MITSIMLab by typing the following:

```
smc -m master.smc
```

Step 4: MITSIMLab's graphical interface pops up on the screen. If executing MITSIM in stand-alone mode, click on the red rectangle at the bottom right hand corner of the screen to start the simulation.

Step 5: The MITSIMLab window disappears automatically at the end of the simulation. Change to the Output directory to view MITSIMLab's output files.

Step 6: If you use the smc version of MITSIMLab, stop running PVM:

- Go into PVM

```
pvm
```

- Stop running PVM

```
halt pvm
```

2. MITSIMLab Files

2.1 Input Files

All of MITSIMLab's input files are in ASCII text format, and can be edited in a standard text editor. The input files required to run MITSIMLab are:

- **Master.mitsim**
- **Master.tms**
- **Master.smc**
- **Paralib.dat**
- **Ctrlpara.dat**
- **Ctrllogic.dat**
- Network Database
- Demand file
- Incident file
- Path Definition File (**path.dat**)

2.1.1 Master.mitsim

The main input file for MITSIMLab is the **master.mitsim** file. This file determines the

- Input and output directory specifications
- MITSIMLab input file names
- MITSIMLab output file names and file formats
- Simulation start time, stop time and step size
- Output parameters
- Simulation parameters

Sections from a sample **master.mitsim** file are shown below.

The file header consists of the title of the network. This is followed by the complete path specifications of the input and output directories.

```
/*
 * MITSIMLab master file
 */

[Title] = "Modified Central Artery Network"

[Default Parameter Directory] = "/export/home/mit/DTA-Mitsim/data/"
[Input Directory]             = "/export/home/mit/DTA-Mitsim/data/"
[Output Directory]            = "/export/home/mit/DTA-Mitsim/data/Output/"
[Working Directory]           = "/tmp/"
```

The next section details the input and output files for MITSIMLab. The main input files are **paralib.dat**, **network.dat** and **demand.dat**. The **paralib.dat** file contains the parameter values that MITSIMLab uses in its models. The **network.dat** file contains the

physical representation of the network. The `demand.dat` file specifies the origin-destination demand as a function of time.

```
[Parameter File]                = "paralib.dat"
[Network Database File]         = "networkrun.dat"
[Trip Table File]               = "/export/home/mit/DTA-Mitsim/data/demand.dat"
[Vehicle Table File]           = ""
[State Dump File]              = ""
[GDS Files]                    = {
%   Filename      MinScale    MaxScale
}
```

The next section in the file specifies the format of the link travel time output file. See section 2.2.5 for details.

```
[Link Travel Times Input File]  = {
    "linktime.dat"    # Historical travel time
    "update.dat"     # Updated travel time
}
```

The next part of the file specifies options relating to the computation of shortest paths. The “`Time variant path calculation`” option directs MITSIMLab to compute time-dependent path travel times instead of an average value. The “`Calculate shortest path periodically`” option causes MITSIMLab to re-evaluate shortest paths en-route for drivers who do not have a pre-assigned path. The “`Update path table travel time periodically`” option forces the path travel times to be updated for all vehicles (including the vehicles with pre-assigned paths). The “`Use existing (cached) shortest path table`” option directs MITSIMLab to use the stored shortest path trees in special cases (for example, when a vehicle misses an off ramp). This option might be used only when most drivers have pre-assigned paths. The “`Updated travel time used for pre-trip plan`” option relates to guided vehicles. If this flag is set, guided vehicles make their pre-trip decisions based on the updated travel times currently available. If this flag is not set, then guided vehicles use the updated travel time information only after they begin their trips.

The numbers associated with all the desired options are summed up in hexadecimal format (base 16), and the result is entered beside the `# SP flags` entry. For example, `0x109` corresponds to {`Time variant path calculation, Use existing (cached) shortest path table, Updated travel time used for pre-trip plan`}

```
0x109 # SP flags
% 0x001 Time variant path calculation
% 0x002 Calculate shortest path periodically
% 0x004 Update path table travel time periodically
% 0x008 Use existing (cached) shortest path table
% 0x100 Updated travel time used for pre-trip plan
}
```

The next section specifies files of interest.

```
[Incident File]                = "incident.dat"
```

[Path Table File] = "path.dat"

The `incident.dat` file contains timing and severity data for incidents. This file is discussed in detail in section 2.1.7.

The `path.dat` file lists the probable paths between each origin and destination in the network. This file is an input to MITSIMLab, and is generated offline.

MITSIMLab can generate a wide range of output files. The main output files from MITSIMLab include:

```
[MOE Specification File]      = ""
[MOE Output File]            = "moe.out"
[Network State Tag]          = "i3dm"
[Segment Statistics File]     = "segstats.out"
[Segment Travel Times File]   = "segtime.out"
[LinkFlowTravelTimes Output File] = "lft_i.out"
[Link Travel Times Output File] = "linktime_mitsim.out"
[Vehicle File]               = "vehicle.out"
[Vehicle Trajectory File]     = "trajectory.out"
[Vehicle Path Record File]    = "pathrec.out"
[Departure Record File]      = "dep.out"
[Queue File]                 = "queue.out"
[Point Sensor File]          = "sensor.out"
[VRC Sensor File]            = "vrc.out"
```

➤ `moe.out`

The `moe.out` file contains maximum head speed, maximum tail speed and maximum gap within lanes.

➤ `i3dm`

The `i3dm` files (also called State 3D files) contain segment-specific flows, speeds and densities, as a snapshot of the network state.

➤ `segstats.out`

The `segstats.out` file contains segment-specific data from the simulation, for each output time interval. The data includes traffic counts at segment entry and exit, average segment density, speed and travel time. The file also contains speeds and densities on each lane within the segments.

➤ `segtime.out`

The `segtime.out` file contains segment-specific flows and travel times.

➤ `lft_i.out`

The `lft_i.out` file contains link-specific flows and travel times.

➤ **linktime.out**

The **linktime.out** file contains link-specific travel times.

➤ **vehicle.out**

The **vehicle.out** file contains data pertaining to origin, destination, departure and arrival times, mileage and speed of vehicles that have reached their destinations.

➤ **trajectory.out**

The **trajectory.out** file contains the positions of all vehicles on the network. This data helps reconstruct vehicle paths.

➤ **pathrec.out**

The **pathrec.out** file provides link entry time and link travel time for all link on the path traced by each vehicle.

➤ **dep.out**

The **dep.out** file contains the departure record of each vehicle in the simulation. This comprises of departure time, vehicle ID, origin, destination, vehicle type and path.

➤ **queue.out**

The **queue.out** file contains queue lengths observed during the simulation.

➤ **sensor.out**

The **sensor.out** file[s] contains the traffic counts registered by the sensors. Sensors with non-default step sizes output to files named **sensor.out##**, where **##** is the length of the step size in seconds. For example, 30-second sensors would output to **sensor.out30**.

➤ **vrc.out**

The **vrc.out** file contains vehicle information as recorded by vrc (vehicle-to-road communication) sensors. Each record corresponds to a single vehicle that has activated an area sensor.

See section 2.2 for more details on MITSIMLab output files.

The next section defines the start time, stop time and time step size of the simulation. The start and stop times define the overall interval for which traffic is being simulated. The step size dictates the smallest unit of time by which the simulation proceeds.

```
[Start Time]           = 07:00:00
[Stop Time]            = 08:12:00
[Step Size]            = 0.2
```

The following section defines various time step sizes. The parameters of importance are the segment data sampling step size, segment data report step size and point sensor step size. All three parameters are defined in seconds. The segment data sampling step size defines the frequency with which data is sampled from the simulation run. It also defines the frequency at which data is reported in the segment statistics file (`segstats.out`). For example, a value of 30 indicates that MITSIMLab retrieves data from its simulation results every 30 seconds. The segment data report step size indicates the time over which MITSIMLab aggregates the data in order to generate its output files. For example, a value of 300 seconds indicates that MITSIMLab averages the sampled data every 300 seconds. The point sensor step size indicates the frequency at which sensors, by default, report their counts. For example, a value of 120 indicates that sensors for which no other step size is specified report their counts every 120 seconds. Additional point sensor step sizes for non-default groups of sensors are in the next line, and the sensor output flags determine whether an output file will be generated for each of these additional groups. In the example shown below, one group of additional sensors will use a 1-second step size and won't have an output file, and another will have a 30-second step size and will have an output file.

```
[Segment Data Sampling Step Size] = 30
[Segment Data Report Step Size]   = 300
[Point Sensor Step Size]          = 120
[Point Sensor Step Sizes]         = { 1 30 }
[Sensor Output Flags]             = { 0 1 }
[Area Sensor Step Size]           = 60
[Animation Step Size]             = 0.1
[Segment Color Step Size]         = 15
[Console Message Step Size]       = 60
```

The next two parameters control the reporting of Measures of Effectiveness by MITSIMLab. `[MOE Step Size]` defines the length of the time interval over which data is collected and averaged. The OD pairs specified after `[MOE OD Pairs]` instruct MITSIMLab on the OD pairs for which the maximum speed and gap data is reported.

```
[MOE Step Size]           = 600
[MOE OD Pairs]            = {
}
}
```

Though MITSIMLab can generate a wide range of output files, the user can control the files that are actually written during the simulation run. Each output file is associated with a unique hexadecimal number. The numbers corresponding to the desired output

files are summed up and entered after the [Output] entry. Note that the summation is carried out in the 16th base i.e. in the hexadecimal representation. For example, a sum of 13357 corresponds to {vehicle log, sensor readings, VRC readings, link travel times, segment statistics, travel time tables, vehicle path records, output rectangular text, no comments and State 3D}

```
[Output] = 0x13357
% 0x00001 = Vehicle log
% 0x00002 = Sensor readings
% 0x00004 = VRC readings
% 0x00010 = Link travel times
% 0x00020 = Segment travel times
% 0x00040 = Segment statistics
% 0x00080 = Queue statistics
% 0x00100 = Travel time tables
% 0x00200 = Vehicle path records
% 0x00400 = Vehicle departure record
% 0x00800 = Vehicle trajectories
% 0x01000 = Output rectangular text
% 0x02000 = No comments
% 0x10000 = State 3D
```

The next section of the master.mitsim file defines parameters that MITSIMLab uses in displaying the network elements and vehicles.

```
[Segments] = 3
% 0 = Link type
% 1 = Density
% 2 = Speed
% 3 = Flow

[Signals] = 0x0
% 0x01 = Traffic signals
% 0x02 = Portal signals
% 0x04 = Variable speed limit signs
% 0x08 = Variable message signs
% 0x10 = Lane use signs
% 0x20 = Ramp meters

[Sensor Types] = 0x0
% 0x1 = Loop detectors
% 0x2 = VRC sensors
% 0x4 = Area sensors

[Sensor Color Code] = 3
% 0 = Count
% 1 = Flow
% 2 = Speed
% 3 = Occupancy

[Vehicles] = 4
% 0 = None
% 1 = Vehicle type
% 2 = Information availability
% 3 = Turning movement
```

```

% 4 = Driver behavior group
% 5 = Lane use

[Vehicle Shade Params] = {
    0      # Shade
    86400 # Outstanding time in a segment
    86400 # Outstanding time in the network
}

```

The next section defines some general parameters.

```

# [Verbose] = 1
# [Nice]    = 1

```

The `[Verbose]` parameter controls the display of output text messages in the MITSIMLab console window. A value of 1 instructs MITSIMLab to display extra messages about MITSIMLab's output, while a value of 0 suppresses them.

The `[Nice]` parameter controls the assignment of processor resources to the various processes running on the system. A value of 0 assigns the highest priority to the MITSIMLab process, and instructs the system to allocate as much of the processor resource to MITSIMLab. A value of 1 allows the user to run other processes in parallel with the MITSIMLab execution.

2.1.2 The `master.smc` file

The `master.smc` file controls the overall running of all the components. The sample file below shows the definitions of input and output directories as well as host machine and display settings for each of the interacting components.

```

# Simlab master file

[Title] = "Central Artery Network"

[Input Directory] = "/export/home/mit/Mitsim/data/"
[Output Directory] = "/export/home/mit/Mitsim/data/output/"

[MITSIMLab] = {
    "master.mitsim"      # master file
    "dtaev"              # host
    "dtaev.sdi.utk.edu:0.0" # display
}

[TMS] = {
    "master.tms"        # master file
    "dtaev"             # host
    ""                  # display
}

[MesoTS] = {
    "master.meso"      # master file
    "dtaev"            # host
}

```

```
        ""                # display
    }

# [Misc] = {
#     "master.mdi"        # master file
#     "dtaev"             # host
#     ""                  # display
# }

# [Verbose] = 1
# [Nice] = 1
```

2.1.3 The `master.tms` file

The `master.tms` file contains the information required to control the movement of traffic on the MITSIMLab network (see Appendix E for an example of the `master.tms` file). The initial section of this file contains the parameter, input, output and working directories.

```
/*
 * TMS master file
 */

[Title] = "Modified Central Artery Network"

[Default Parameter Directory] = "/export/home/mit/Mitsim/data/"
[Input Directory]             = "/export/home/mit/Mitsim/data/"
[Output Directory]            = "/export/home/mit/Mitsim/data/Output/"
[Working Directory]           = "/tmp/"
```

The next section details the names of the network file (`network.dat`), control parameters file (`ctrlpara.dat`), control logic file (`ctrllogic.dat`) and some parameters related to traffic control on the network. Refer to Appendix C and Appendix D for examples of the `ctrlpara.dat` and `ctrllogic.dat` files.

```
[Network Database File]       = "networkrun.dat"
[GDS Files]                   = {
%   Filename   MinScale   MaxScale
}
[Parameter File]              = "ctrlpara.dat"
[Control Logic File]          = "ctrllogic.dat"
```

The Signal Plan File contains the control logic for different types of signals in the network. However, the signal logic data is currently stored in the `ctrllogic.dat` file (see Appendix D for details).

```
[Signal Plan File]           = ""
[Control Logic]               = 0
%   0 = None
%   1 = A1 incident response
%   2 = Gating logic
```

The next section defines the mode of operation with respect to information. Option 0 causes MITSIMLab to simulate traffic based on historical data. Option 1 causes drivers in MITSIMLab to use instantaneous travel times. Under option 2, drivers in MITSIMLab use predicted travel times. Under option 0, no information is used.

```
[Information]                 = 2
%   0 = Historical data
%   1 = Real time measurement
%   2 = Prediction
```

The following section defines the simulation time parameters. The start time and stop times define the simulation period. The step size defines the time step at which the simulation runs.

```
[Start Time]           = 07:00:00
[Stop Time]            = 08:12:00
[Step Size]            = 0.1
```

The next section defines the parameters related to the design of the ATIS, the generation of guidance, and its dissemination to MITSIMLab drivers.

```
[RollingStepSize]      = 600
[RollingLength]        = 1800
[NumOfDTAIterations]   = 2
[DTAComputationalDelay] = 110
```

[Rolling Step Size] defines (in seconds) the frequency at which the ATIS generates guidance. [Rolling Length] defines the prediction step length (in seconds). The Number of DTA Iterations (**NumOfDTAIterations**) specifies the number of iterations performed by the ATIS within each guidance generation phase.

[DTAComputationalDelay] specifies the time delay (in seconds) before which the guidance becomes available to drivers.

The following section defines parameters relating to the depiction of the network on the graphical interface.

```
[Console Message Step Size]      = 60

[Segments] = 3
% 0 = Direction
% 1 = Link type
% 2 = Density
% 3 = Speed
% 4 = Flow

[Signals] = 0x0
% 0x01 = Traffic signals
% 0x02 = Portal signals
% 0x04 = Variable speed limit signs
% 0x08 = Variable message signs
% 0x10 = Lane use signs
% 0x20 = Ramp meters

[Sensor Types] = 0x0
% 0x1 = Loop detectors
% 0x2 = AVI sensors
% 0x4 = VRC sensors
% 0x8 = Area sensors

[Sensor Color Code] = 3
% 0 = Count
% 1 = Flow
% 2 = Speed
```

```
% 3 = Occupancy

# [Randomize] = 0
# [Verbose]   = 1
# [Nice]     = 1
```

2.1.4 The `paralib.dat` file

The `paralib.dat` file contains the values of all the parameters used within MITSIMLab. Sections from a sample `paralib.dat` (see Appendix A) file are provided in this section with an explanation of the various inputs. A comprehensive list of all parameters included in the `paralib.dat` file is provided in Appendix B.

The first part of the `paralib.dat` file contains constants used throughout the simulation and legends used by the GUI.

```
/*
 * This file contains the parameters used by MITSIMLab
 */

# Following parameters transfer units in the British system into
# metric system.

[Native Length to Meter]           = 0.3048
[Native Speed to Meters per Second] = 0.4470
[Native Density to Vehicles per Kilometer] = 0.6214
[Native Flow to Vehicles per Hour]   = 1.0000
[Native Travel Time to Minute]       = 1.0000
[Native Demand to Vehicles per Hour] = 1.0000

# LEGEND UNITS

[Density Label]    = "Density (vpm)"
[Speed Label]     = "Speed (mph)"
[Flow Label]      = "Flow (vph)"
[Occupancy Label] = "Occupancy (%)"

# THRESHOLDS FOR VIEWING GRAPHICAL OBJECTS (optional)
# Show the objects when pixels per meter is greater than the
# value provided here.

[Resolution] = {
  1 # lanes marks
  2 # lane type, vehicles, sensors and signals
  6 # vehicle lane change indicators
 10 # vehicle labels
}

[FontSizes] = {
  100 120 140 160 180
}
}
```

The next section contains parameters that determine the route choice models used to simulate drivers' route choice behavior in MITSIMLab.

The models are based on a C-Logit implementation that uses travel time (normalized by the travel time on the shortest path), commonality factor (in order to correct for the violation of the IIA property when paths have common sections), freeway bias and diversion penalty in computing the systematic utility components.

$$V_{in} = \beta_1 tt_i + \beta_2 (CF)_i$$

The diversion penalty is added to the travel time of the corresponding route(s).

The parameter `Fraction` defines the fraction of unguided and guided vehicles. Beta is the coefficient of the travel time variable for unguided and guided drivers (respectively). The remaining lines determine the values of the beta coefficients for the other explanatory variables (common to both unguided and guided drivers). A more detailed description of the route choice implementation in MITSIMLab is provided in Appendix H.

ROUTE CHOICE PARAMETERS FOR UNGUIDED AND GUIDED VEHICLES

% MITSIM uses the following parameters in its logit route choice models to determine the paths that drivers will choose. Vehicles may be guided or unguided and may have predefined or dynamically computed paths.

% The probability that a driver will choose a certain route is determined using a logit choice model, whereby drivers judge the utilities of alternative paths. The utility of a path is a function of its travel time, calculated based on either historical information or real-time link travel times, depending on whether the vehicle is guided or not. The first column is percentage of each type of vehicle, and the 2nd column is the parameter coefficient of travel time on a path choice in the logit model (travel time ratio with respect to the travel time on shortest path).

```
[Route Choice] = {
%   Fraction   Beta
%   1.0        -5.0
%   0.0        -5.0
}
```

% When two paths have common sections, drivers may not consider one of the feasible paths. The Commonality Factor is a penalty applied to a path to correct for this error such that the driver will consider each path option as distinct and separate choices.

```
[Commonality Factor] = -1.0
```

% Drivers tend to prefer to travel on limited access roadways such as freeways. To capture this bias, a Freeway Bias factor is applied in the when link travel times are updated and used in shortest path calculations. The travel times on freeway links are divided by this factor to reduce the perceived cost of travel on the link. Therefore, the factor should be greater than or equal to 1.

```

[Freeway Bias]      = 1.0      # Travel time factor

% In the route generation model, the routes for vehicles without
% pre-specified paths are generated at each intersection. In the
% route switching model, vehicles are assigned pre-specified paths,
% but may switch to alternative routes at intermediate nodes. In
% the route switching model, a diversion penalty is applied to
% alternative routes to simulate the phenomenon that drivers are less
% inclined to divert from their current, or habitual, routes. In the
% route generation model, the diversion penalty is applied to paths
% require exit from a freeway link to a non-freeway link.

[Diversion Penalty] = {
    5.0 # in route generation model (minutes)
    1.0 # in route switching model (minutes)
}

% Not all feasible routes are considered by drivers. In MITSIM's
% vehicle routing models, path choices with travel times greater than
% the shortest path travel time multiplied by the Valid Path Factor
% are not valid and are thus not considered by drivers.

[Valid Path Factor] = 3.0      # compare to shortest path

```

Among the route choice parameters included in the `paralib.dat` file is the rational link factor (RLF), which is applied to those vehicles whose paths are determined on a link-by-link basis. The value of the RLF determines which of the downstream links connecting to the vehicle's destination will be considered. The RLF can have a value between 0 and 1, where 0 is the least restrictive (i.e. all links connecting to the vehicle's destination will be considered, including links on redundant, circulating paths) and 1 is the most restrictive (i.e. only links which bring the vehicle closer to the destination in terms of travel cost will be considered).

```

% Drivers without a pre-specified path travel according to the link
% based route choice model (i.e. route generation model). They
% decide at each node which link will be taken at the downstream
% node of the current link. The Rational Link Factor serves two
% purposes. The first is to avoid discarding downstream links
% connecting to the destination that will take the driver to a node
% that has a travel time to the destination greater than that at
% at the upstream node when the difference is small. The second is
% to avoid the case where drivers choose unrealistic, circulating
% paths. There are two extremes:
%
% RLF = 1.0 (default); any link ending at a node with a cost label
%                   (i.e. travel time to destination) greater
%                   than the upstream node is ineligible, even
%                   reasonable, comparable links
% RLF = 0.0; all links are eligible, even circulating, irrational
%                   link choices
%
% The RLF condition looks like so:
% if RLF*Label(j) <= Label(i) then consider the link
% otherwise disregard the link

```

```

% where Label(j) is the cost associated with the downstream node
% of the link in question and Label(i) the cost of the shortest
% path from the upstream node of the link in question
%
% note: values of RLF less than about 0.5 have essentially the
% same effect as RLF = 0

```

```
[Rational Link Factor] = 0.5
```

Updated MITSIMLab travel time information is computed as a function of new and previous travel time data. The section that follows contains specifications regarding the travel time information generation process.

```
# PARAMETER FOR INFORMATION UPDATE
```

```

% This parameter is used to combine the new travel time with the
% earlier information when travel time table are updated periodically
% based on prevailing traffic condition. i.e.
%

```

```

%   updated = (1-r) * old + r * new
%   where r=[Path Alpha]

```

```
[Path Alpha] = 0.75
```

[Path Alpha] is substituted for 'r' in generating updated travel times as a linear combination of old and new travel times.

The next section specifies parameters governing the frequency with which drivers update their driving behavior in response to the changing traffic environment. These parameters are related to the drivers' reaction time and depend upon the state of the vehicle (e.g. accelerating, decelerating, traveling at uniform speed, or stopped).

```
# UPDATE STEP SIZE (REACTION TIME RELATED)
```

```

% Driver behavior (e.g. responses to stimuli) is updated periodically
% during simulation depending on the state of the vehicle (i.e.
% depending on whether the vehicle is decelerating, accelerating,
% travelling at uniform speed, or is stopped). The update time
% step sizes are as follows:

```

```

[Update Step Sizes] = { # normal distributions (seconds)
%   mean  stdev  lower  upper
%   0.5   0.0    0.5    0.5    # decelerating
%   1.0   0.0    1.0    1.0    # accelerating
%   1.0   0.0    1.0    1.0    # uniform speed
%   0.5   0.0    0.5    0.5    # stopped vehicle
}

```

The following section contains parameters describing the manner in which vehicles are loaded into the simulation at the origins of the network. The loading parameters describe the minimum time headway with which subsequent vehicles may be loaded onto the same entry link from the pretrip queue and the entering speeds of the vehicles.

```
# VEHICLE LOADING PARAMETERS
```

```

% The vehicle loading parameters govern the way vehicles are
% entered into the network from pretrip queues at each entry
% link.  If the time since the last vehicle entered the
% simulation and left the queue is greater than the loading
% headway parameter below, then the next vehicle in the
% queue is eligible for entry.  The coefficient for the queue
% (Beta) is a parameter describing the entry speed of the
% vehicle.  The entry speed will be some value between the
% mean speed on the entry link and the desired speed.
% Initially, the entry speed was some value between the
% 'speed at capacity' parameter value and the desired speed.
% Currently, MITSIM does not use the 'speed at capacity'
% value, but rather the average speed on the link.
%
% where:   queue = (queue length)/(number of lanes)
%          r = Beta * queue^2
%
% and if r > -5, then
%   entry speed = (mean speed)
%                 + exp(r)*(desired speed - mean speed)
%
% else entry speed = mean speed

[Loading Model] = {
    0.60 # headway (seconds) (var:loadingHeadway)
   -0.25 # coef for queue (var:loadingQueueBeta)
    45.0 # speed at capacity (var:loadingSpeed)
}

```

The section that follows contains information on the types of vehicles, vehicle characteristics, and the vehicle mix. The summation of all vehicle classes should equal unity.

VEHICLE MIX AND ATTRIBUTES

```

% The mix of vehicles in the simulation will is assigned the
% following attributes.
%
% (1) Label (Type)
% (2) Typical length
% (3) Typical width
% (4) Percentage
% (5) delay factor at toll booth
% (6) ETC probability
% (7) Over height probability
% (8) HOV probability.

```

```

[Vehicle Classes] =
{
%   1           2 3         4 5       6 7       8
%   "New Cars"   18 6  0.410 1.0  1.0  0.0  0.00
%   "Old Cars"   18 6  0.400 1.0  1.0  0.0  0.00
%   "Taxis"      18 6  0.100 1.0  0.0  0.0  1.00
%   "Buses"      40 8  0.060 1.5  0.0  0.0  1.00
%   "Trucks"     50 8  0.030 1.5  1.0  0.0  0.00

```

```

    "Trailer Trucks" 70  8  0.000 2.0  1.0  0.0  0.00
}

% The Min Heavy Vehicle Length is the minimum vehicle length for
% which vehicles will be classified as heavy.

[Min Heavy Vehicle Length] = 30      # feet

```

The section that follows specifies the various parameters used by the driver behavior models, which include:

- car following
- lane changing and merging
- gap acceptance

Drivers' acceleration behavior in MITSIM is modeled according to a general acceleration model that distinguishes between car following and free flow regimes. In a car following regime, the driver's acceleration and deceleration is directly influenced by the speed of the lead vehicle. In a free flow regime, a proximate lead vehicle does not influence the driver's acceleration behavior and the driver will vary the speed of the vehicle in order to achieve a desired speed. A MITSIMLab simulation may use either of two car following models: General Motors (GM) model and Kazi's model.

The GM model has the following functional form:

$$a_n = \alpha * (V_n^\beta / \Delta X^\gamma) * \Delta V + N(0, \text{stddev}^2)$$

Kazi's car following model takes on the following functional form:

$$a_n = \alpha * (V_n^\beta / \Delta X^\gamma) * k_n^\rho * \Delta V^\lambda + N(0, \text{stddev}^2)$$

where

- a_n = acceleration/deceleration of the subject vehicle n
- V_n = speed of subject vehicle n
- ΔX = distance between the subject vehicle n and its leader
- k_n = density of traffic ahead of subject vehicle n
- ΔV = speed difference between the lead and subject vehicles
- $\alpha, \beta, \gamma, \rho, \lambda$ = car following parameters used by MITSIMLab

CAR-FOLLOWING PARAMETERS

```

% The following parameters describe the boundaries of the car following
% regime. The following vehicle is in a car following regime (i.e.
% accelerates according to the car following models described below)
% when the time headway between the lead and following vehicles is
% between the CF Upper and Lower Bounds. The Min Response Distance is
% the minimum space headway between the lead and following vehicles for
% which the following vehicle must apply some acceleration (or
% deceleration).

```

```

[Min Response Distance] = 15 # feet

```

```

[CF Lower Bound] = 0.40      # seconds
[CF Upper Bound] = 1.36      # seconds, not used for KAZI's ACC model

% The General Motors (Herman) car-following model describes the
% acceleration of the following vehicle in a car following regime:
%
% Acceleration = alpha * (v ^ beta / dx ^ gamma) * dv + N(0, stddev^2)
%
% where v = speed of the following vehicle, dx = distance gap between
% the lead and following vehicles, and dv = speed difference between
% the lead and following vehicles. Below are the car following
% parameters used by MITSIM when using the General Motors CF model.
% In this model, stddev is 0.

[CF Parameters] = {
%   alpha beta gamma stddev
%   2.15  -1.67  -0.89 #0      # acc
%   1.55  1.08   1.65 #0      # dec
}

% Similarly, Kazi's car following model takes on the following
% functional form:
%
% Acceleration = alpha * (v ^ beta / dx ^ gamma)
%               * (k ^ rho) * (dv ^ lambda) + N(0, stddev^2)
%
% where v = speed of the following vehicle, dx = distance gap between
% the lead and following vehicles, k = the traffic density downstream
% of the following vehicle, and dv = speed difference between the lead
% and following vehicles. Below are the car following parameters used
% by MITSIM when using Kazi's CF model. In Kazi's new car following
% model, the stddev are used to create the bins in columns 4-5 in
% 'Driver Groups' and are not used by MITSIM directly.

[CF Kazi Parameters] = {
%   alpha beta gamma rho lambda stddev
%   0.0500 0.722 0.242 0.682 0.600 #0.825 # acc 0.0225 (original alpha)
%   -0.0550 0.000 0.151 0.804 0.682 #0.802 # dec
}

% Kazi formulates the free flow acceleration rate as:
%
% acc = b[2] * stimulus + N(0, stddev)
%
% where:
%
% Spd      = speed of the subject vehicle
% Stimulus = b[3] + spdlmt + b[4] * front->Spd + b[5] * IsHeavy
%           + b[6] * IsFreeFlow - spd
%
% This model applies only if Spd is greater than b[0]. IsFreeFlow is
% defined as segment density and is less than or equal to b[1] vehicles
% per km per lane (level of service A, B, or C). All units are already
% in the metric system and no conversion is required.
%
% The stddev is indirectly used in creating col 6 in 'Driver Group'.

```

```
[FF Acc Params] = { # Spd is in meter/sec, acc is in meter/sec2
%   b[0]  b[1] b[2]   b[3]      b[4]    b[5]    b[6]    # std dev
      10.0  19  0.3091  -12.365  0.6182  -0.670  7.5974  #  1.134
}
```

The merging model parameters below are used by MITSIMLab to determine whether a vehicle is in the act of merging. The merging model specifies the physical extents and capacity of the merging area. A vehicle is tagged as merging if it is located within the merging area and there is available capacity.

MERGING

```
% The merging model is used to describe driver merging behavior. A
% vehicle is merging if it is within some distance D2 downstream of the
% merging point (e.g. point where 2 lanes become 1). If a vehicle is
% some distance D1 upstream of the merging point, it will be merging if
% there is available capacity in the merging area (i.e. there are fewer
% than NCAP vehicles in the merging area defined by D1 and D2). If
% there exists an upstream vehicle in the freeway lanes when the
% vehicle in question is entering from an on-ramp, the entering
% vehicle may or may not accelerate to merge based in part on the
% probability P that the driver will perform an aggressive merge.
```

```
[Merging Model] = {
    100 # D1, upstream area (feet) (var: upMergingArea)
    200 # D2, downstream area (feet) (var: dnMergingArea)
    8   # NCAP, number of vehicles allowed
        # (var: nVehiclesAllowedInMergingArea)
    0.2 # P, probability of aggressive merge from ramp
        # (var: aggressiveRampMergeProb)
}
```

The following section contains lane changing parameters that are used to determine whether a driver will respond to a mandatory lane changing situation, whether a driver will accept a gap in an adjacent lane once it has been selected, whether merging vehicles will nose in and whether other vehicles will yield to the merging vehicle.

In MITSIMLab, drivers enter a hierarchical decision making process before making a lane change. The first decision that must be made is whether a vehicle is in a mandatory lane changing situation. A vehicle might be in a mandatory lane changing situation if its current lane is ending or if its current lane does not connect to the next link in its path. If a vehicle is in a mandatory lane changing situation, the probability that the driver will respond to and begin the mandatory lane change is determined by the parameters in the following section.

LANE-CHANGING PARAMETERS

```
% Drivers are assumed to make a series of decisions to determine
% whether a lane change will be undertaken. The hierarchical decision
% process is modeled using the random utility approach.
```

```
[LC Entry Lane Check Rightness Prob] = 1.0
```

```

% The probability of starting a mandatory lane change or staying in the
% current lane is determined according to the following parameters.
% The model requires that the vehicle be greater than a distance 'lower
% bound' from the downstream node (or lane drop) requiring the
% mandatory lane change in order for the vehicle to be eligible for a
% mandatory lane change. The probability that a vehicle will change to
% the appropriate lane is given by:
%
% Prob = exp (-dis*dis/(delta^2))
%
% where: dis = (Xn - Xo) - 'lower bound'
%         where Xn = distance from vehicle to downstream node (or lane
%         drop) and Xo = distance of a critical location (e.g. final
%         exit warning)
%         delta = delta1 * [1.0 + alpha1*number of lanes +
%         alpha2*(segment density/jam density)]
%
% A vehicle must remain in a lane for a minimum of 1.0 seconds before
% considering another lane change.

[LC Mandatory Probability Model] = {
    330.0      # lower bound, feet
    1320.0    # delta1, feet
    0.5       # coef for number of lanes, alpha1
    1.0       # coef for congestion level, alpha2
    1.0       # minimum time in lane
}

```

If a driver has decided to make a lane change, whether mandatory or discretionary, the driver must then determine whether gaps in the target lane(s) are acceptable for merging. In the section below, parameters are specified for calculating critical lead and lag gaps. Figure 1.1 illustrates the gap acceptance models for vehicles considering merging into an adjacent lane.

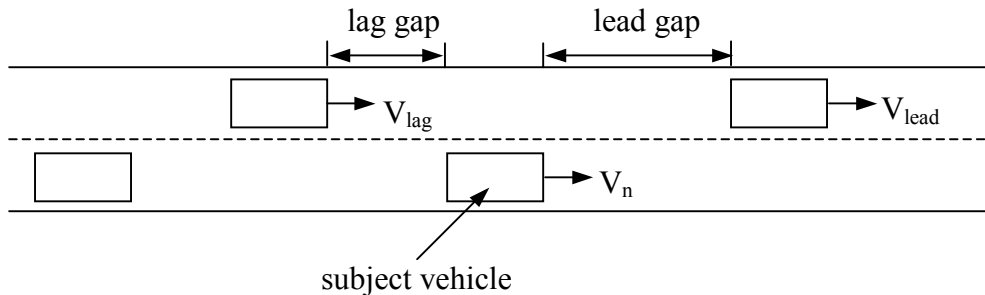


Figure 1.1: Lead and lag gaps for gap acceptance models.

```

# GAP ACCEPTANCE MODEL FOR LANE CHANGING

```

```

% Once a driver has decided to change lanes, he/she will examine
% adjacent gaps and deem them acceptable or unacceptable for merging.
% Below are parameters used by MITSIM to describe driver gap

```

```

% acceptance.

% 1. Parameters in Kazi's LC gap model (see Kazi's PhD thesis)
%
% G(cr,lead,dlc)= critical lead gap for discretionary LC case
%               = exp[beta0-beta4*min(0,dVlead)+N(0,sigma^2)]
% G(cr,lag,dlc) = exp[beta0+beta3*min(0,dVlag)
%                   +beta4*max(0,dVlag)+N(0,sigma^2)]
% G(cr,lead,mlc)= critical lead gap for mandatory LC case
%               = exp[beta0+N(0,sigma^2)]
% G(cr,lag,mlc) = exp[beta0+beta3*min(0,dVlag)
%                   +beta4*max(0,dVlag)+N(0,sigma^2)]
%
% where:
% dVlead = speed difference between subject vehicle and lead
%         vehicle in gap in target lane
% dVlag  = speed difference between subject vehicle and lag
%         vehicle in gap in target lane
% scale  = scaling the critical gap
% alpha  = minimum gap (meter)
% lambda = remaining distance to impact
%         = 10 [1 - 1/{1+exp(lambda * dis)}]
% beta0  = constant
% beta1  = coef of remaining distance to impact
% beta2  = coef of speed difference (m/s, d/s-u/s)
% beta3  = coef of negative speed difference (m/s, d/s-u/s)
% beta4  = coef of positive speed difference (m/s, d/s-u/s)
% sigma  = std dev of generic random term

[Kazi LC Gap Models] = {
% scale alpha lambda beta0 beta1 beta2 beta3 beta4 sigma
% Discretionary (Lead and Lag)
  1.00 0.0 0.000 0.508 0.000 0.000 0.000 0.420 0.488
  1.00 0.0 0.000 2.020 0.000 0.000 0.153 0.188 0.526
% Mandatory (Lead and Lag)
  0.00 0.0 0.000 0.384 0.000 0.000 0.000 0.000 0.859
  1.00 0.0 0.000 0.587 0.000 0.000 0.048 0.356 1.073
}

% 2. Parameters in Qi's gap acceptance model
%
% G(cr,lead,dlc) = max{alpha,alpha+beta1*Vn+beta2*(Vn-Vlead)}
% G(cr,lag,dlc)  = max{alpha,alpha+beta1*Vlag+beta2*(Vlag-Vn)}
% G(cr,lead,mlc) = max{alpha,alpha+[beta1*Vn+beta2*(Vn-Vlead)]
%                   * [1-exp(-gamma*Xn^2)]}
% G(cr,lag,mlc)  = max{alpha,alpha+[beta1*Vlag+beta2*(Vlag-Vn)]
%                   * [1-exp(-gamma*Xn^2)]}
%
% where:
% Vn      = speed of subject vehicle
% Vlead   = speed of lead vehicle in gap in target lane
% Vlag    = speed of lag vehicle in gap in target lane
% 0 scale = type specific scaler
% 1 gamma = distance parameter = 1 - exp(-gamma * dis * dis)
% 2 alpha = minimum gap (feet)
% 3 beta1 = coef of following vehicle's speed (spd, feet/sec)
% 4 beta2 = coef of speed difference (dv, feet/sec)

```

```

[Qi LC Gap Models] = {
% scale   gamma   alpha   beta1   beta2
% Discretionary (Lead and Lag)
    0.50    0.00    3.00    0.05    0.10
    0.50    0.00    5.00    0.10    0.30
% Mandatory (Lead and Lag)
    1.00   2.5E-5    3.00    0.05    0.10
    1.00   2.5E-5    5.00    0.10    0.30
}

```

The following sections specify parameters used in MITSIMLab to determine the probability that a vehicle will nose into a target lane and the probability that a vehicle in the target lane will yield to the merging vehicle.

The probability that a vehicle will nose into the target lane can be determined by either of two nosing models in MITSIMLab. In the first, the probability that a vehicle will nose in is assumed to be a function of the number of lane changes necessary to connect with the next link on the vehicle's path and of the time the subject vehicle has been waiting to make a lane change (i.e. the time elapsed since the vehicle has been tagged as making a lane change).

In the second model, Kazi's nosing model, the probability to nose in is estimated as a function of the lead relative speed of the subject vehicle (i.e. the ratio of the subject vehicle's speed to the speed of the lead vehicle in the target gap), the remaining distance to the point before which the lane change must be completed, the sum of the lead and lag gaps, and the number of lane changes necessary to connect with the next link on the vehicle's path.

```

# PARAMETERS FOR YIELDING PROBABILITIES

```

```

[LC Yielding Model] = { # currently yields to nosing vehicle only
    0.80    # previously not yielding
    1.00    # previously yielding
}

```

```

# PARAMETERS FOR NOSING AND YIELDING

```

```

% The following parameters are used to return the probability
% that a vehicle will nose in.
%
% Prob = Pmax*[{1+cos(pi*y^z)}/2]
%
% where y    = x/L where x=distance of vehicle from critical position
%           and L=length of link or view range
%       z    = beta0 + beta1*N + beta3*T
%       Pmax = maximum probability of nosing
% also, the probability will be updated every Thor seconds, the vehicle
% will wait no longer than 'max stuck time' to nose in, the vehicle
% is only eligible for nosing when less than 'max distance for nosing'
% and greater than 'min distance for nosing' from the critical position
% before which lane change must be completed.

```

```

[LC Nosing Model] = {
    1.0 # Thor, time horizon (seconds) for constant acc
    0.5 # beta0, constant
    0.6 # beta1, coef for number of lanes, N
    0.1 # beta3, coef for time (minutes) since tagged, T
    0.2 # max prob: case of ramp lane drop, incident, red LUS, etc
    1.0 # max prob: case of connection to next link on path
    300.0 # max yielding time (seconds)
    180.0 # max stuck time (seconds)
    600.0 # max distance for nosing
    40.0 # min distance for nosing
}

% Kazi's Nosing model directly calculates the probability to nose when
% distance from a point is less than a threshold.
%
%  $P = 1 / (1 + \exp(-u))$ 
%
% where:
%
%  $u = b[0] + b[1] * \min(0, \text{lead\_rel\_spd}) + b[3] * \text{rem\_dis\_impact} +$ 
%  $b[4] * \text{gap} + b[5] * \text{nlanes}$ 
%
%  $\text{rem\_dis\_impact} = 10 - 10 / (1 + \exp(b[2] * \text{dis}))$ 
%
% where:
% lead_rel_spd = speed of subject vehicle subject to speed of lead
%               vehicle in gap, i.e.  $V_n/V_{\text{lead}}$ 
% rem_dis_impact = remaining distance impact = f(dis, the critical
%               position before which lane change must be completed)
% gap = sum of the lead and lag gaps
% nlanes = number of necessary lane changes in order to
%         connect to next link on vehicle's path

[LC Kazi Nosing Model] = {
%   b[0]   b[1]   b[2]   b[3]   b[4]   b[5]
    -3.159  0.313  -0.027  2.050  0.028  0.6
}

# PROBABILITY OF YIELDING TO OTHER VEHICLES

% The following are the probabilities used by MITSIM that a driver will
% yield to no other vehicles, up to 1 other vehicle, up to 2, or up to
% 3 other vehicles.

[MLC Yielding Probabilities] = {
    0.13 # None
    0.71 # Up to 1
    0.13 # 2
    0.03 # 3
}

```

In the following section, the parameters governing discretionary lane changing in MITSIMLab are stored. There are two models.

In Kazi's discretionary lane change model, drivers that are not in a mandatory lane changing situation or that are not responding to an existing mandatory lane changing situation will consider a discretionary lane change at every discrete point in time. The decision to make a discretionary lane change is modeled according to the random utility approach and is performed in two steps. The driver must first decide that the driving conditions in the current lane are unsatisfactory. Second, the driver must prefer the conditions in an adjacent lane to those in the current lane.

PARAMETER FOR DISCRETIONARY LANE CHANGE

% The following are parameters used by MITSIM governing whether a driver will undertake a discretionary lane change.

```
[LC Discretionary Lane Change Model] = { # Do not change the order
    5.00      # 0 Step size for update lane speed
    0.50      # 1 Prob of DLC given not DLC in last step
    0.90      # 2 Prob of DLC given DLC in last step
    0.80      # 3 Impatient lower bound
    1.00      # 4 Impatient upper bound
    0.85      # 5 Slower lead threshold
    0.10      # 6 Speed threshold
    0.85      # 7 Acceleration threshold
    300.00    # 8 Speed ahead looking distance
    3.00      # 9 Minimum time in lane (same direction)
    10.00     # 10 Minimum time in lane (different direction)
    10.00     # 11 Maximum speed difference between neighbor lanes
    (mph)
}
```

% In Kazi's lane changing model, drivers not in a mandatory lane change situation or not responding to a mandatory lane change are assumed to enter the discretionary lane change decision process at every discrete point in time. The driver first decides whether conditions in the current lane are satisfactory, second whether other lanes are preferable to the current lane, third which lane to select, and finally whether the lead and lag gaps in the selected lane are acceptable. Each decision is made based on the random utility approach, where the probability of selecting a discrete choice is determined based on their relative utilities.

% Each driver's desired speed is assumed to be a function of the average speed of the vehicles ahead of the subject vehicle, or the desired speed = $V^* = (\text{avg speed ahead}) * (B11)$, where B1 is the model parameter coefficient of the average speed ahead.

% Prob(driving conditions not satisfactory) = $1 / [1 + \exp\{- (B2 + B3 * (V_{\text{sub}} - V^*) + B4 * d_{\text{heavy}} + B5 * d_{\text{tail}})\}]$

% where V_{sub} = speed of subject vehicle
 d_{heavy} = 1, if subject is a heavy vehicle, 0, otherwise
 d_{tail} = 1, if subject being tailgated, 0, otherwise

% Prob(choose left (or right) lane | driving conditions not satisfactory) = $1 / [1 + \exp\{- (B6 + B7 * (V_{\text{lead}} - V^*) + B8 * (V_{\text{front}} - V^*) + B9 * (V_{\text{lag}} - V_{\text{sub}}))\}]$

```

%
% where Vlead = speed of lead vehicle in lane being considered
%       Vfront = speed of vehicle directly in front of subject
%       Vlag   = speed of lag vehicle in lane being considered
%
% The 'gap behind threshold for tailgate dummy' is the gap length
% between the subject and its following vehicle below which the
% following vehicle will be considered to be tailgating. In densities
% above the 'density threshold for tailgate dummy', no vehicles will be
% considered to be tailgating.

```

```

[LC Kazi DLC Model] = {
    0.7682      # B1, Segment average speed for desired spd model
    0.2250      # B2, Constant for driving conditions unsatisfactory
   -0.0658     # B3, Speed less desired speed
   -3.1472     # B4, Heavy vehicle dummy
    0.4226     # B5, Tailgate dummy
    10         # gap behind threshold for tailgate dummy (meters)
    19         # density threshold for tailgate dummy
    2.0800     # B6, constant for utility of an adjacent lane
    0.0337     # B7, (lead_speed_in_target_lane - desired_speed)
   -0.1520     # B8, (front_speed_in_same_lane - desired_speed)
    0          # B9, subject_speed - lag_speed
    0.0000     # previous DLC state dummy
}

```

The sections that follow contain parameters governing the start-up delays applied to each vehicle according to its position in a queue and the driver familiarity with the network.

START-UP DELAYS

```

% Drivers in a queue will resume movement after some delay depending
% on their position in a queue, with some limiting maximum delay.
% Below are the mean delay values for vehicles in particular positions
% in a queue. Start up delay for vehicles further back in the queue
% than those listed are assumed to have zero start up delay.

```

```

[Start Up Delay Parameters] = { # seconds
    2      # maximum delay
% PositionInQueue Mean
    0      1.5
    1      1.2
    2      1.0
    3      0.9
    4      0.8
    5      0.6
    6      0.5
    7      0.4
}

```

```

% The Familiarity parameter describes driver familiarity with the
% network which can affect driver behavior, where Familiarity = 1.0
% means all drivers are completely familiar with the driving
% environment.

```

```
[Familiarity] = 0.9
```

The `paralib.dat` file also contains parameters pertaining to the level of compliance of traffic to controls such as traffic signals and route guidance messages, driver reaction to a yellow signal indication and tollbooth delays.

```
# RESPONSE TO CONTROLS
```

```
% The Visibility Scaler is a measure of driver visibility of traffic  
% controls, where 1.0 means all drivers may see all controls clearly.
```

```
[Visibility Scaler] = 1.0
```

```
% Compliance Rates determine the extent to which drivers heed traffic  
% controls, where 1.0 means that all drivers obey the controls.
```

```
[Compliance Rates] = {  
    1.0 # Traffic signals only  
    1.0 # Portal signals and traffic signals only  
    0.9 # Ramp meters, portal signals and traffic signals  
    0.9 # Red LUS only  
    0.5 # Yellow and red LUS  
    1.0 # Lane use rules only  
    1.0 # Lane change and lane use rules  
    1.0 # Vehicle type related lane use message  
    1.0 # Path related lane use message  
    0.7 # Route guidance message  
}
```

```
% When the signal is yellow, then, if the distance to the intersection  
% divided by the MAX(current speed, minimum speed) (i.e. the time  
% to reach the intersection) is greater than the 'Time to Treat  
% Yellow as Red', then the vehicle will break to stop.
```

```
[Time To Treat Yellow as Red] = 1.0 # seconds
```

```
% The Maximum Toll Booth Delay is the maximum delay to a vehicle at  
% a toll booth.
```

```
[Maximum Toll Booth Delay] = 10.0 # seconds
```

The following sections contain parameters regarding the variance in headway gap acceptance in lane changing, merging and car-following, the jam density and the minimum speed.

```
# PARAMETERS FOR HEADWAY VARIANCE
```

```
% A headway buffer is found for each vehicle and is a behavioral  
% parameter that describes the aggressiveness of a driver for accepting  
% a headway gap in lane changing, merging, and car-following. Some  
% value between the lower and upper bound will be added to the minimum  
% headway gaps for the population, which are constants also provided in  
% this file.
```

```
[Headway Buffer Lower Bound] = 0.1 # seconds
[Headway Buffer Upper Bound] = 0.2 # seconds
```

```
# LIMITING PARAMETERS
```

```
% Jam Density is the maximum traffic density in vehicles per mile when
% speed is a minimum. Minimum Speed is the minimum vehicle speed on
% the network.
```

```
[Jam Density] = 210.0 # vehicles/mile
[Minimum Speed] = 5.0 # mph
```

```
% The next parameter is used in calculation of harmonic mean of speed.
% If a vehicle is stopped at the sensor, its speed is zero. The
% Detected Minimum Speed value below is used to avoid the problem of
% dividing by zero.
```

```
[Detected Minimum Speed] = 5.0 # mph (must be > 0)
```

The section below contains parameters describing the distribution of driver groups in the simulation and a number of behavioral characteristics pertaining to each driver group.

```
# DISTRIBUTION OF DRIVER GROUPS
```

```
% Speed limits come from JPL data. MaxAccScale is made up. Each row
% corresponds to the percentile specified in the first column. Each
% column is a variable:
%
% 0 = Max acceleration scale
% 1 = Max deceleration scale
% 2 = Normal deceleration scale
% 3 = Car-following acceleration add on (see also CF Kazi Parameters)
% 4 = Car-following deceleration add on (see also CF Kazi Parameters)
% 5 = Free-flow acceleration add on (see also FF Acc Params)
% 6 = Speed limit add on for desired speed
% 7 = Upper headway threshold (Kazi, was 1.36)
% 8 = Percentile (not read)
```

```
%[Driver Groups] =
%{
%      0      1      2      3      4      5      6      7      8
%      0.6  1.0000 1.0000 -0.4721 -0.3520 -0.3158 -4.67  1.1741 # 0.05
%      0.7  1.0000 1.0000 -0.2975 -0.2218 -0.1990 -1.73  1.4650 # 0.15
%      0.8  1.0000 1.0000 -0.1936 -0.1443 -0.1295 -0.20  1.7155 # 0.25
%      0.9  1.0000 1.0000 -0.1106 -0.0825 -0.0740  0.97  1.9470 # 0.35
%      1.0  1.0000 1.0000 -0.0361 -0.0269 -0.0241  1.98  2.1723 # 0.45
%      1.1  1.0000 1.0000  0.0361  0.0269  0.0241  3.05  2.4019 # 0.55
%      1.2  1.0000 1.0000  0.1106  0.0825  0.0740  4.06  2.6480 # 0.65
%      1.3  1.0000 1.0000  0.1936  0.1443  0.1295  5.33  2.9296 # 0.75
%      1.4  1.0000 1.0000  0.2975  0.2218  0.1990  6.71  3.2902 # 0.85
%      1.5  1.0000 1.0000  0.4721  0.3520  0.3158  8.94  3.9091 # 0.95
%}
```

```
[Driver Groups] =
```

```

{
%   0      1      2      3      4      5      6      7      8
0.6  1.00  1.00 -1.3564 -1.3187 -1.8654 -0.1911  1.7498 # 0.05
0.7  1.00  1.00 -0.8547 -0.8309 -1.1754 -0.0708  2.2737 # 0.15
0.8  1.00  1.00 -0.5562 -0.5407 -0.7649 -0.0082  2.5871 # 0.25
0.9  1.00  1.00 -0.3178 -0.3089 -0.4370  0.0397  2.8379 # 0.35
1.0  1.00  1.00 -0.1036 -0.1007 -0.1425  0.0810  3.0633 # 0.45
1.1  1.00  1.00  0.1036  0.1007  0.1425  0.1248  3.2814 # 0.55
1.2  1.00  1.00  0.3178  0.3089  0.4370  0.1661  3.5068 # 0.65
1.3  1.00  1.00  0.5562  0.5407  0.7649  0.2180  3.7578 # 0.75
1.4  1.00  1.00  0.8547  0.8309  1.1754  0.2745  4.0718 # 0.85
1.5  1.00  1.00  1.3564  1.3187  1.8654  0.3657  4.5979 # 0.95
}

```

The remaining sections specify performance characteristics, such as maximum acceleration and deceleration rates and limiting maximum speeds, of the various vehicle types included in the simulation.

ACCELERATION TABLES

```

% Following are the acceleration and deceleration characteristics
% of the vehicle classes named earlier in the file.

```

```

% Converting factors used in next three tables:
% The following factors are used to convert the acceleration table
% units to the metric system.

```

```

[Acc Table Speed to Meters per Second] = 0.3048
[Acc Table Acc to Meters per Sq Second] = 0.3048

```

```

% In determining the maximum acceleration of a vehicle based on
% grade and speed, max acc = max acc (based on speed and vehicle
% type, from the tables below) - acc grade factor * grade

```

```

[Acceleration Grade Factor] = 0.305 # feet/sec/sec/grade

```

```

[Grade Scaler] = { 5 -2 2 }
[Speed Scaler] = { 5 20 20 }

```

```

% Global scalers: The acceleration scaler is multiplied by the
% acceleration rate for each vehicle type when the acceleration
% rate is read from this file. The deceleration scaler is
% similarly multiplied by the deceleration rate for each vehicle
% type.

```

```

[Acceleration Scaler] = 1.25 % global scaler
[Deceleration Scaler] = 1.00 % global scaler

```

```

% The following table lists the maximum accelerations of vehicle
% types travelling in a given speed range.

```

```

[Maximum Acceleration] =
{
% Each row corresponds to one vehicle class and each column to a
% speed range (<20 20-40 40-60 60-80 >80 feet/sec)

```

% These values are based on level grade data of INTRAS document

%	11.0	11.0	10.0	5.0	3.0
%	6.0	6.0	6.0	3.0	2.0
%	3.0	2.0	1.0	1.0	1.0
%	3.0	2.0	1.0	1.0	1.0
%	1.0	1.0	1.0	1.0	1.0

% These were values used in previous runs

%	11.0	11.0	10.0	5.0	3.0
%	10.0	10.0	8.0	5.0	3.0
%	8.0	8.0	6.0	4.0	2.0
%	8.0	8.0	6.0	4.0	2.0
%	6.0	6.0	4.0	2.0	1.0

% These are the values from traffic engineering handbook (1992)

%	7.50	7.00	5.91	4.99	4.50	
%	7.50	7.00	5.91	4.99	4.50	# N/A
%	7.50	7.00	5.91	4.99	4.50	# N/A
%	1.60	1.45	0.89	0.47	0.40	
%	1.60	1.45	0.89	0.47	0.40	# N/A

% These are the values used in fresim

%	10.00	7.90	5.60	4.00	4.00	
%	8.71	5.17	4.43	2.89	2.00	
%	8.71	5.17	4.43	2.89	2.00	# N/A
%	2.80	2.50	1.50	1.00	0.50	
%	2.80	2.50	1.50	1.00	0.50	# N/A

% These are the values we use now

10.00	7.90	5.60	4.00	4.00
8.71	5.17	4.43	2.89	2.00
10.00	7.90	5.60	4.00	4.00
7.00	5.00	4.00	1.50	1.00
2.80	2.50	1.50	1.00	0.50
1.60	1.45	0.89	0.47	0.40

}

% The following table lists typical deceleration rates of
% vehicles classes in a given speed range under normal
% deceleration conditions.

[Normal Deceleration] =

{

Rows: vehicle types
Cols: Speed (<20 20-40 40-60 60-80 >80 feet/sec)

7.8	6.7	4.8	4.8	4.8
7.8	6.7	4.8	4.8	4.8
7.8	6.7	4.8	4.8	4.8
7.8	6.7	4.8	4.8	4.8
7.8	6.7	4.8	4.8	4.8
7.8	6.7	4.8	4.8	4.8

}

% The following table lists the maximum deceleration rates of
 % each vehicle class travelling in a given speed range.

```
[Maximum Deceleration] =
{
  # Rows: vehicle types
  # Cols: Speed (<20 20-40 40-60 60-80 >80 feet/sec)

  16.0  14.5  13.0  11.0  10.0
  16.0  14.5  13.0  11.0  10.0
  16.0  14.5  13.0  11.0  10.0
  16.0  14.5  13.0  11.0  10.0
  16.0  14.5  13.0  11.0  10.0
  16.0  14.5  13.0  11.0  10.0
}
```

LIMITING SPEED BY VEHICLE TYPE AND GRADE

% The speed of each vehicle class is limited by the following
 % maximum values for given surface grades.

```
[Limiting Speed] = {
  # Rows: vehicle types
  # Cols: Grade (<-2 -2~0 0~2 2~4 >=4%)

%   200  200  200  200  200
%   200  200  200  200  200
%   100   98   84   70   57
%   100   98   84   70   57
%    85   84   50   32   23

  200  200  200  200  200 # feet/sec
  200  200  200  200  200
  200  200  200  200  200
  150  125  100   80   60
  130  105   80   65   45
  100   90   80   60   40
}
```

LANE SPEED RATIO

% The speed at which vehicles travel is observedly influenced by
 % the lane in which the vehicle is travelling and the number of
 % lanes. Where there are multiple lanes, vehicles typically
 % travel faster in the left lanes (Lane Speed Ratio > 1) and slower
 % in the right lanes (LSR < 1). The Lane Speed Ratios below
 % describe this phenomenon.

```
[Lane Speed Ratio] =
{
  # Rows: Total number of lanes
  # Cols: Lane number (left to right)

  1.00
  1.06 0.94
  1.06 1.01 0.93
}
```

```

    1.05 1.05 0.97 0.93
    1.04 1.06 1.01 0.95 0.94
}

```

2.1.5 The `network.dat` File

The `network.dat` file stores the physical representation of the network. This includes the nodes, links, segments and lanes. This file also contains the link connectivity that defines the network and the lane connectivity that defines the allowable vehicle movements.

The `network.dat` file is created either manually (through a text editor) or using RNE (Road Network Editor), a GUI-based application developed at MIT. The RNE generates this data file automatically (see section 3).

The following sections specify the structure of the network file. This would aid in understanding the function of each component defined within the file, and also in manually modifying or adding components to the network.

2.1.5.1 Network Data

The network file contains information about the following:

- Nodes
- Links
- Segments
- Lanes
- Lane connectivities
- Sensors

Nodes

The `network.dat` file first specifies the nodes. The number of nodes follows `[Nodes]`. A `Node ID`, a `node Type` and a `Name` characterize each node. The node type is designated by a number (centroid = 0, entry/exit node = 1, intersection = 2). An entry or exit node lies on the boundary of the network representation. It might not correspond to a node in the actual network. The node name is an identifier that helps the user easily locate each node.

```

[Nodes] : 182
{
  # {NodeID Type "Name"}
  {0 1 "nd-airp-egr-a"}
  {1 0 "nd-i93nb-rc-d/s"}
  {2 1 "nd-rrv-u/s"}
  {3 1 "nd-rrt-u/s-to-i93nb"}
  {4 0 "nd-i93sb(w)-rrs-d/s"}
  {5 1 "nd-rrs-u/s-to-i93sb(w)"}
  {6 1 "nd-rrr-d/s-from-i93sb"}
  {7 0 "nd-i93sb(w)-rrr-u/s"}
  {8 0 "nd-i93sb-i93sb(w)-u/s"}
  {9 1 "nd-rcsp-d/s-from-i93sb"}
}

```

Links, Segments and Lanes

After the nodes the file specifies the links, segments and lanes. Links form connections between the nodes, and correspond to actual links in the physical network. Links are further sub-divided into segments, and segments consist of lanes. Segments are user-defined. Links may be divided into segments with homogeneous characteristics.

In the `network.dat` file, the symbol `[Links]` is followed by the number of links, the number of segments, and the number of lanes.

The link, segment and lane components are specified from `[Links]` downwards. For each link, the segments and lanes it contains are specified.

A link is characterized by a `Link ID`, a link type (`LinkType`), an upstream node (`UpNodeID`), a downstream node (`DnNodeID`), and a parameter called `LinkLabelID`. The link type can be one of the following:

- 1 = freeway
- 2 = ramp
- 3 = urban road

In addition, each link type listed above can be defined as a tunnel section by adding eight (8) to the basic link type. For example, a freeway-tunnel section would have a link type code of $1+8=9$.

The `LinkLabelID` is set to zero for all links. This attribute is not used by MITSIMLab and DynaMIT.

Segments are characterized by a segment ID (`SegmentID`), speed limit (`SpeedLimit`) (mph), free flow speed (`FreeSpeed`) (mph), gradient (`Grad`) (%), speed-density index (`SpeedDensityIndex`), (X,Y) coordinates of the starting point (`StartingPntX`, `StartingPntY`), (X,Y) coordinates of the ending point (`EndPntX`, `EndPntY`), and bulge (`Bulge`).

The speed density index is an input parameter and is used by the supply simulator. The bulge specifies the curvature of the segment in radians. (See figure 2.1.)

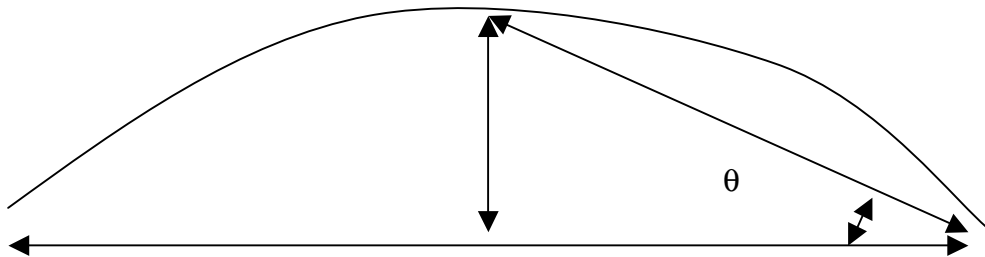


Figure 2.1: Illustrating Bulge for a Segment

Segments are further subdivided into lanes. The lanes are defined below each segment definition in the `network.dat` file. A lane is characterized by a **LaneID** and a list of lane rules (**Rules**). The lane rules define allowed lane changes, and lane usage. The list below enumerates the various options:

- 0 Can go straight ahead
- 1 Can change to a lane on the right
- 2 Can change to a lane on the left
- 64 ETC (Electronic Toll Collection) Lane
- 128 HOV (High Occupancy Vehicle) Lane

The Lane Rules Code for a lane is the sum of all possible options from the list above. For example, a lane rule of 3 allows changes to both left and right lanes, while a lane rule of 129 corresponds to a HOV lane allowing changes to lanes on the right.

Sample link, segment and lane definitions are shown below.

```
[Links] : 211 : 639 : 1259
{
# Link: // listed in arbitrary order
# {LinkID LinkType UpNodeID DnNodeID LinkLabelID
#
# Segment: // listed upstrm to dnstrm within link
#   {SegmentID SpeedLimit FreeSpeed Grad SpeedDensityIndex
#     {StartingPntX StartingPntY Bulge EndPntX EndPntY}
#
# Lane: // listed from left to right
#   {LaneID Rules}
# }

{0 3 0 90 0
  {0 25 30 0
    {785002 2.95962e+06 -0.025405 784775 2.95969e+06}
    {3 1} {2 3} {1 3} {0 2}
  }
  {1 25 30 0
    {784775 2.95969e+06 0.00479548 784670 2.95973e+06}
    {7 1} {6 3} {5 3} {4 2}
  }
}
}
.
.
.
```

The Lane Connectors

The next section in the file details the connectivity of the lanes between sections. The number following `[Lane Connectors]` specifies the total number of lane connectors defined. A lane connector is defined as {upstream laneID, downstream laneID}. If two

lanes are not connected, a vehicle will have to change lanes in the upstream segment before moving into the lane in the downstream segment. Sample lane connectivity definitions are shown below. For example, {3 7} defines a lane connection between lane 3 and lane 7.

```
[Lane Connectors] : 1242
{
  # {UpLane DnLane}
  {3 7}
  {2 6}
  {1 5}
  {0 4}
  {7 1256}
  {6 1255} {6 1254}
  {5 1253}
  {4 1252}
  {9 11} {9 10}
  {12 13}
  .
  :
  .
}
```

Turn Prohibitors

Turn prohibitors help control link movements. Each entry in this section of the `network.dat` file defines an “illegal” movement from one link to another, even though the two links might physically be connected. MITSIMLab does not allow vehicles to make these turns. A `FromLink` and a `ToLink` characterize each turn prohibitor. The number following `[Turn Prohibitors]` specifies the number of turn prohibitors defined.

```
[Turn Prohibitors] : 6
{
  # {FromLink ToLink}
  {1 185}
  {59 62}
  {145 147}
  {146 149}
  {153 157}
  {161 164}
}
}
```

Sensors

Sensors are defined according to the following format:

```
[Sensors] : 35
{
  # {TypeCode TaskCode ZoneLength SegmentID PosInSegment
  #   {SensorID WorkProbability LaneID} // lane specific
  #   or
  #   {SensorID WorkProbability} // link specific
  # }
{257 0x01c1 6 636 0.50
  {0 1 }
}
}
```

```

{257 0x01c1 6 477 0.50
  {1 1 : 1}
}
{257 0x01c1 6 497 0.50
  {2 1 }
}
.
.
.

```

The number following `[Sensors]` specifies the number of sensors on the network. A sensor is associated with a sensor ID (`SensorID`), type code (`TypeCode`), task code (`TaskCode`), length (`ZoneLength`), segment ID (`SegmentID`, the ID of the segment on which the sensor is located), and a position in the segment (`PosInSegment`, expressed as a fraction from the downstream node). The length of the sensor corresponds to the length over which the sensor can be triggered. A sensor position of 0.50, for example, corresponds to the mid point of the segment. In addition, each sensor has an associated working probability. The working probability represents the reliability of the sensor's measurements and performance. Sensors can be specific to either a link or a lane. Sensors which use non-default step sizes are denoted by a ":" followed by a number, which corresponds to the order of the non-default point sensor step sizes listed in **master.mitsim**. In the example above, sensor 1 will use the first non-default step size listed in **master.mitsim**. The type code can be one of the following:

```

Traffic           :1
Vehicle           :2
VRC               :3
Area              :4

```

The type code is the number corresponding to the sensor type. In addition, the sensor can either be specific to a lane, or can cover the entire width of the link. If the sensor is operational across the entire width of the link, a value of 256 is added to the type code. For example, a type code of 257 represents a traffic sensor operational across the entire width of the link.

The sensor task code (`TaskCode`) represents the mode of operation of the sensor and the data it collects. The corresponding hexadecimal value is the sum (in base 16) of the different options described below:

Aggregate traffic data can be sent to TMS or written into a **traffic sensor** data file:

```

0x0001 Flow
0x0002 Speed
0x0004 Occupancy

```

Data generated by **individual** probe vehicles is sent to TMS or written into **vehicle to roadside communication (VRC) sensor** data file immediately after the a vehicle pass the detector:

```

0x0010 ID
0x0020 Type
0x0040 Departure time
0x0080 Origin
0x0100 Destination
0x0200 Path
0x0400 Height
0x0800 Current speed

```

Finally, a sensor device (such as a camera) can take **snapshots** of vehicles which are in the detection zone and report the following data items:

```

0x0010 ID
0x0020 Type
0x0400 Height
0x0800 Current speed
0x1000 Lane ID
0x2000 Position in lane

```

For example, a task code of 0x01c1 corresponds to {flow, departure time, origin, and destination}.

2.1.6 The demand.dat File

The `demand.dat` file specifies the time-dependent historical OD flows. The file specifies a set of OD flows and the times when these OD flows are loaded onto the network. The sample section from a demand file is shown below.

```

25200 0 1
{
  { 90 44 2600 }
  { 90 130 400 }
  { 106 44 150 }
  { 106 130 150 }
  { 129 44 180 }
  { 129 130 180 }
  { 153 44 160 }
  { 153 130 160 }
  { 169 44 20 }
  { 169 130 20 }
}

26100 0 1
{
  { 90 44 2600 { 1 2 } }
  { 90 130 400 { 3 4 } }
  .
  .
  .

```

The numbers in parenthesis (e.g. {1 2}) next to the OD flow entries show the allowed paths between the corresponding OD pair. If no paths are listed for a particular OD pair, MITSIMLab moves the vehicles using a link-based route choice model.

Refer below for an illustration of the data contained within the demand file.

The diagram shows a sample demand file with three callout boxes:

- Top Callout:** Points to the time field '0' in the header '25200 0 1'. Text: "Indicates the time (in seconds) when the OD matrix is loaded. A time of zero signifies midnight. This OD matrix will be in effect until OD flows for a next time are defined."
- Middle Callout:** Points to the scaling factor '0.80' in the entry '169 44 20 0.80 { 1 2 }'. Text: "Modify this **Scaling Factor** to alter the demand level. For example, a scaling factor of 0.80 represents 80% of the demand that follows."
- Bottom Callout:** Points to the path notation '{ 1 2 }' in the entry '169 44 20 0.80 { 1 2 }'. Text: "(Origin, Destination, Demand (veh/hr))"

```

25200 0 1
{
  { 90 44 2600 }
  { 90 130 400 }
  { 106 44 150 }
  { 106 130 150 }
  { 129 44 180 }
  { 129 130 180 }
  { 153 44 160 }
  { 153 130 160 }
  { 169 44 20 0.80 { 1 2 } }
  { 169 130 20 }
}
  
```

A set of OD demands defined for a particular time is valid until the next time for which a new set of OD demands is specified.

The historical OD flows can be modified in three primary ways:

- Change the scaling factor for one or more time intervals.
- Manually edit the flows for specific OD pairs.
- Append / modify time intervals with new OD pairs and OD flows.

Additional parameters may be specified for each OD pair: the standard deviation of the departure rate (demand), and a distribution factor. In the example below, the standard deviation is 0.0 and the distribution factor is 1.0 for both OD pairs.

```

26100 0 1
{
  { 90 44 2600 0.0 1.0 { 1 2 } }
  { 90 130 400 0.0 1.0 { 3 4 } }
  .
  .
  .
}
  
```

The standard deviation of the average departure rate represents the randomness of total travel demand for the corresponding OD pair. The departure rate used in a particular simulation run is sampled from a normal distribution based on the given average departure rate and its standard deviation. The default value for the standard deviation is 0, i.e. no error in the departure rate.

The distribution factor, which is a value between 0 and 1, determines the percentage of vehicles departing randomly. For example, a distribution factor of 0.4 indicates that 40% of vehicles will depart according to a Poisson distribution (random) and the remaining 60% of vehicles will depart according to constant headways. The default value for the distribution factor is 0, i.e. all vehicles depart with constant headways.

2.1.7 The `incident.dat` file

The `incident.dat` file defines the locations, timings, and severity of incidents. Each incident is associated with a visibility distance (**VisDist**) (in feet), **SegmentID** and the location of the incident as a fraction of segment length from downstream end of the segment (**EndPosition**). An incident is assigned a severity code (**SeverityCode**) (see file header below for detailed explanation), capacity factor (**CapFactor**), incident start time (**StartTime**) (measured in seconds from midnight), incident duration (**Duration**) (in seconds), speed limit during the incident (**SpeedLimit**) (miles per hour), ID of the lane affected by the incident (**LaneID**), and an **IncidentID**. The capacity factor defines the capacity reduction caused by the incident. For example, a capacity factor of 0.40 in the example below reduces the capacity of the lane to 40% of its initial value. DynaMIT utilizes this capacity factor when the user runs MITSIMLab and DynaMIT in a closed loop. MITSIMLab captures the impact of incidents through the speed limit.

```

/*
 * Incident File
 *
 * Severity of an incident is defined by a severity code
 * (0=ignore 1=minor 2=major) and a factor of the deduction
 * of the lane capacity.
 */

# { VisDist SegmentID EndPosition(%)
#   { SeverityCode CapFactor StartTime Duration SpeedLimit LaneID
IncidentID }
# }

{
    # 26100 start at 7:15
    150 586 0.50
    {2 0.40 25200 1800 2 1174 912}
    {2 0.40 25200 1800 2 1175 913}
}

```

2.1.8 The Path Definition File (`path.dat`)

The path definition file specifies the list of available paths between each OD pair. Each path is characterized by a path ID, origin node, destination node and a list of consecutive links that defines the path from origin to destination (See sample file shown below).

```

/*
 * Path Table File

```

*/

[Path Table] : 16

```
{
  { 1 90 44 { 209 194 196 198 145 149 148 188 124 126 127 131 132 }
327.1 }
  { 2 90 44 { 209 178 180 11 156 184 5 6 61 64 65 67 70 72 73 100 103
104 132 } 813.5 }
  { 3 90 130 { 209 178 180 11 156 184 5 6 75 52 50 36 28 } 751.4 }
  { 4 90 130 { 209 194 196 198 145 149 148 188 143 141 108 86 57 40 45
46 47 49 36 28 } 543.9 }
  { 5 106 44 { 146 147 148 188 124 126 127 131 132 } 593.6 }
  { 6 106 130 { 146 147 148 188 143 141 108 86 57 40 45 46 47 49 36 28
} 779.7 }
  { 7 129 44 { 152 154 160 156 184 5 6 61 64 65 67 70 72 73 100 103 104
132 } 692.2 }
  { 8 129 44 { 152 157 7 151 183 188 124 126 127 131 132 } 321.6 }
  { 9 129 130 { 152 154 160 156 184 5 6 75 52 50 36 28 } 662.9 }
  { 10 129 130 { 152 157 7 151 183 188 143 141 108 86 57 40 45 46 47 49
36 28 } 535.8 }
  { 11 153 44 { 153 154 155 159 7 151 183 188 124 126 127 131 132 }
327.3 }
  { 12 153 44 { 153 154 160 156 184 5 6 61 64 65 67 70 72 73 100 103
104 132 } 791.5 }
  { 13 153 130 { 153 154 160 156 184 5 6 75 52 50 36 28 } 650.7 }
  { 14 153 130 { 153 154 155 159 7 151 183 188 143 141 108 86 57 40 45
46 47 49 36 28 } 537.7 }
  { 15 169 44 { 179 180 11 156 184 5 6 61 64 65 67 70 72 73 100 103 104
132 } 850.1 }
  { 16 169 130 { 179 180 11 156 184 5 6 75 52 50 36 28 } 800 }
}
```

% Converted from:

% ptabsetpath35sens_1.out

% Using the program PathSorter by Qi Yang

2.2 MITSIMLab Output Files

MITSIMLab has the capability to report the results of its simulations through a wide range of output files. The particular files of interest are listed below:

- Segment Statistics : `segstats.out`
- Segment Travel Time Information : `segtime.out`
- Link Flow and Travel Time Data : `lft_i.out`
- Link Flow Travel Time Data : `linktime_mitsim.out`
- Speed and Occupancy Data : `Sensor.out`

The details of these files are discussed below.

Note: The names of the files mentioned above are user-defined.

2.2.1 The `segstats.out` File

The `segstats.out` file is generated by MITSIMLab, and contains segment-specific data from the MITSIMLab simulation. This therefore represents what actually happened on the network. The file is organized by time interval. Section 2.2.5 details the method of controlling the frequency and length of the output time intervals. For each time interval, the file contains data classified by segment ID. Each segment record contains:

- Segment ID
- Counts at segment entry (vehicles)
- Counts at segment exit (vehicles)
- Density (vehicles per mile per lane)
- Speed (miles per hour)
- Travel time (seconds)
- Density and speed on each lane within the segment

```
% SEGMENT STATISTICS
% Generated by MITSIMLab on Fri Jul 14 16:03:31 2000
% Reporting time {
%   Segment EnterCount LeaveCount Density Speed TravelTime {
%     { LaneDensity LaneSpeed }
%   }
% }

25200 {
  0 0 0 0 30 5.4 {
    {0 31.5}
    {0 31.5}
    {0 29.1}
    {0 27.9}
  }
.
.
.
563 0 0 0 30 6 {
  {0 31.8}
  {0 28.2}
}
}
```

```

564 4 4 11.3 24.1 8.2 {
  {17 20.4}
  {0 30.3}
  {17 27.8}
}
565 4 4 13.8 23.1 6.9 {
  {0 31.8}
  {20.7 26}
  {20.7 20.1}
}
566 4 3 6.8 24.8 7 {
  {0 31.8}
  {0 30.3}
  {20.4 24.8}
}
}
.
.
.

```

2.2.2 The segtime.out File

The `segtime.out` file is generated by MITSIMLab, and contains segment travel time data. The file header specifies the start time of the simulation, number of time periods and the time (in seconds) per period, for which the average travel time is reported.

```

% SEGMENT TRAVEL TIMES
% Generated on Wed Aug 30 16:05:22 2000
25200 % Start time
18    % Number of periods
240   % Seconds per period

```

The next section lists the average travel times aggregated for each segment, during each period. Each record consists of a segment ID, segment type and segment length (in feet), followed by flow (counts) and travel time (in seconds) entries in pairs.

```

0 3 237.65 0 5.4 0 5.4 0 5.4 0 5.4 0 5.4 0 5.4 0 5.4 0 5.4 0 5.4
0 5.4 0 5.4 0 5.4 0 5.4 0 5.4 0 5.4 0 5.4 0 5.4
.
.
.
11 9 213.961 51 4.8 83 4.7 78 4.8 100 4.9 102 5.1 110 5.1 99 4.9 92 5.2
75 4.7 86 4.8 79 4.7 73 4.6 62 4.6 70 4.9 62 4.7 62 4.6 48 4.8 51 4.7
.
.
.
12 9 316.423 49 6.1 84 6 76 6.2 98 5.9 105 6.3 109 6.3 99 6.2 93 6.5 76
5.9 86 5.9 79 5.8 73 5.6 61 5.6 72 5.9 62 5.7 62 5.7 48 6.1 49 5.6
.
.
.
13 9 721.071 46 13.3 85 13.2 78 13.3 96 13 105 13.1 109 13.2 98 13.2 95
13.3 77 13 83 12.7 78 12.8 76 12.6 61 12.3 72 12.7 63 12.6 60 12.5 50
.
.
.

```


.
.
.
.

2.2.4 The linktime_mitsim.out File

The linktime_mitsim.out file contains link-based travel times for each period within the simulation window. Seen below is a portion of a sample file. The file is organized similar to the link flow and travel time data files. Each record contains a link ID, link type, link length (in feet), and travel time entries (in seconds) for each time period. In the file shown below, there are 18 travel time entries in each link record. See section 2.2.5 for the template specification.

```
% LINK TRAVEL TIMES
% Generated on Mon Aug 28 17:28:17 2000
0      % Start time period (07:00:00)
18     % Number of periods
240    % Seconds per period

% ID Type Length T0 ... T17
{
  0 3 350 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
  1 1 426.7 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5
  4.5 4.5 4.5
  2 2 196 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5
  4.5 4.5 4.5
  .
  .
  .
  6 9 6041.3 111.5 112 114.4 112.6 112.4 112.7 111.9 115.9 109.9 111.3
  108.3 107.6 108 109.1 107.9 108 110.4 109.3
  7 1 1523.7 35.7 35.1 35.7 33.6 36.7 35.5 34.1 35.7 34.5 36.7 36.9
  37.2 36.9 33.9 36.5 34.1 36.4 36.2
  8 2 792 . . .
  .
  .
  .
```

2.2.5 Specifying the number of periods in the link flow and travel times output files

The number of periods and period duration to report within the link flow and travel time file (lft_i.out) and linktime_mitsim.out file are specified through a template file placed in the input file directory. This file contains the number of periods and the period length at which to report data in the output files. The template file is titled linktime.dat, and an example of such a file is illustrated below:

```
% GENERATED GUIDANCE: LINK TRAVEL TIMES
```

```
% Generated on Fri May 1 00:09:08 1999
0      % Start time period (07:00:00)
18     % Number of periods
240    % Seconds per period
```

```
% ID Type Length T-0 ... T17
```

```
{
}
```

The template filename is specified in the master.mitsim file (see below).

```
[Link Travel Times Input File] = {
    "linktime.dat"    # Travel Time template File
    "update.dat"     # Updated travel time
```

3. The Road Network Editor (RNE)

- Table of Content
 - [How to get on-line help?](#)
 - [How to start the road network editor?](#)
 - [How to choose a different editing mode?](#)
 - [How to create network objects?](#)
 - [How to create surveillance sensors?](#)
 - [How to create traffic signals?](#)
 - [How to edit parameters for each object?](#)
 - [How to select figures?](#)
 - [How to move figures?](#)
 - [How to measure the distance between two points?](#)

How to get on-line help?

A brief help message can be obtained by running *rne* with a help switch:

```
rne -h or  
rne -help
```

In the main window of *rne*, type either `?` or `ctrl-h` will popup a message window show the key binding.

Key Bindings	
<code>^h or ?</code>	display this message
<code>^o</code>	display the options menu
<code>^p</code>	print canvas (not implemented)
<code>^s</code>	save all databases
<code>1</code>	append nearest existing point to figure
<code>2</code>	append new point to figure
<code>3</code>	alter last figure point
<code>4</code>	select previous figure
<code>5</code>	select current figure
<code>6</code>	select next figure
<code><Enter></code>	complete current figure or place selected point
<code>N</code>	mark current figure (link) incomplete
<code></code>	delete current figure
<code>.</code>	delete last point from figure or delete selected point
<code><LMB></code>	select point nearest cursor

+ or <MMB>	display the figure parameters dialog
0 or <RMB>	display the editor selection menu
q	quit program
i	zoom in
o	zoom out
l	move view left
r	move view right
u	move view up
d	move view down
c	center view on cursor
>	rotate view CW
<	rotate view CCW
v	toggle tape-measure
^c	abort program

where <LMB> is left mouse button, <MMB> middle mouse button and <RMB> right mouse button.

A . can be used to delete the last segment of the selected link. <Enter> complete a selected link if the link is incomplete or place a select point at cursor position otherwise.

How to start the Road Network Editor?

The road network editor should be started with a filename that specifies the network database:

```
rne filename
```

If the file *filename* exists, that file will be loaded into *rne*; otherwise, *rne* will create an empty network database with the given name.

The following command will load a network database with the named GDS files as the background map features:

```
rne filename -gds gdsfile1 gdsfile2 ...
```

This is useful because GDS objects (drawings of the roadway alignments, building, signs, etc) can be used as reference in creating the network to be used in the simulation..

Each new node created in *rne* is automatically assigned a code. For existing nodes, their codes are verbosely copied when the database is saved. However, an optional switch "-n" can be used to renumber all the nodes in the network:

```
rne filename -n
```

With this "-n" option, the *rne* will renumber the nodes in the network with unique codes started from 0.

The filename for a network database should have an extension *.rne*. This file is used only internally in *rne* itself. The network file created for simulation will have the same root name but with an extension *.dat*.

How to choose a different editing mode?

Click right mouse button or press **0** in the view area will popup a menu which allows you choose various editing mode (read only, editing nodes, links, sensors, and so on). The new editing mode will affect all subsequential operations until you choose a different one.

How to create network objects?

Network objects include *nodes*, *links*, *segments*, *lane connections*, and *toll-plaza*.

Create Nodes

A node is either an intersection of several roadways or a origin and/or destination where traffic flows enter or leave the simulated network.

1. Click the **right mouse button** in the drawing window or press key **0** to popup the editor menu. Select the editor "Nodes".
2. Move the mouse pointer to the position where you want to add a node. Press key **2** to create the node.

Repeat step 2 to create more nodes. You do not have to make a node where road geometry or number of lanes changes; instead, you divide the [link](#) (which connects two end nodes) into [segments](#) to represents the changes in road geometry and traffic attributes.

Each node is represented by its *type* (e.g. intersection, origin/destination), a unique *identification number*, and an optional *name*. You can [select a node](#) and then click **middle mouse button** to popup a dialog which allow you specify these parameters.

Create Links

Links are directional roadways that connect nodes. Each link consists of one or more [segments](#), whose geometries are represented by arcs.

1. Click the **right mouse button** or key **0** to select the editor "Links".
2. To create a new link, move the mouse pointer near the starting [node](#) and press key **1** or **2**. With key 1 the starting point will be snapped to the nearest [shape point](#) and with key 2 a new shape point will be created as the starting point of the link.
3. [Create one or more segments](#) for the link.
4. To end a link, press the **Enter** key. This will complete the link by connecting the end of the link to the closest node. If there is an ambiguity for **rne** to choose automatically an unique end node for the link, it will ask for your help.

Each link is characterized by its *type* (freeway, ramp, urban street, tunnels, etc.), an *identification number*, *starting and end nodes*, and the [segments](#) it contains. You can [select a link](#) and then click **middle mouse button** to popup a dialog which allow you specify the type of the link. You may also [select a shape point](#) and [change the geometry](#) of any segment contained in the link.

Create Segments

Segments are road sections with uniform characteristics such as number of lanes, grade, curvature, design speed, etc. The geometry of a segment is referenced to the left curb line represented an arc. An arc is described by the *coordinates* of its two end points and a *bulge*. If the bulge is zero, the arc is simply a straight line.

Segments are actually created in [Link Editing Mode](#) with the following two methods:

1. To make a straight segment, create a straight line two shape points by two key strokes:

2 2

with cursor (+) pointed at the beginning and ending points of the segment.

2. To make a curved segment, create an arc with the following sequence of key strokes:

2 3 2

with cursor (+) pointed at the starting, a middle, and the ending points of the segment respectively. Key "3" in this sequence indicates that the segments to be created has three control points and the next key stroke will make the middle point.

Any of the "2" in above sequences can be replaced by "1" if there already exist a shape point at the desired position. In other words, type a "2" will create a new shape point and "1" will use the

nearest existing shape point. The recommended sequence of keys (without the "-") for creating the segments of a link is:

2-32-2 1-32-2 1-32-2 1-32-2 <Enter>

where each of four key strokes (2322 or 1322) creates one curved segment which shares the ending shape point with the previous segment. The last key stroke (a <Enter>) complete the link by connecting the end point of its last segment to the nearest [node](#). You may omit the middle point for any segment (i.e. use 22 or 12 instead of 2322 or 1322) if the segment is straight.

Also notice that at the end of each two (or four for a curved segment) key strokes, a dialog is automatically displayed for specifying the parameters of the segment just created. Each segment is characterized by its *speed limit* , *design speed* , *grade* , *geometry* (i.e. the arc or straight line created above), a unique *identification code* , and the *lanes* it consists of. In the segment parameter dialog, you can set the number of lanes contained in the segment, the type of the lanes, and lane change regulations. Note that the number of lanes is specified by the "Lanes to right". By default this number is zero for the first (upstream) segment of the link, which means only one lane in the segment. For the second and the remaining segments in the link, the *number of lane to the right* will inherit its value from the previous (upstream) segment in the link. If number of the segment is changed, the dialog is reposted so that you will be able to modify the attributes for both the existing and newly added lanes.

Create Lane Connections

Click the **right mouse button** or press **0** in the view area to select the editor in "Lane Connections" mode. Move the mouse pointer in the upstream lane and type **2**, then move the mouse pointer in the downstream lane and type **2**. A short arrow will be drawn to indicate a lane connection. In case there is ambiguity for *rne* deciding a unique lane connection, it will popup a dialog and ask you to make a choice.

Note that if the number of lanes in a segment is the same as that in the upstream segment, the lane connections between the two consecutive segments will be automatically created.

To remove a lane connection, [select the lane connection](#) and press key .

Create Toll Plazas

In the down left side of parameter dialog for a segment, choose "Has Toll Plaza". The toll plaza will be placed at the end of the segment.

How to create sensors?

- Click the **right mouse button** or **0** in the view area to popup the editor chooser dialog and select "Sensors" mode.
- Type key **3** to popup the type dialog and select a sensor type.
- Move the mouse pointer to the location you want to place the sensor and type **2**.

Repeat step 3 above to create more sensor of the same type or follow step 2 to choose a different sensor type.

How to create traffic signals?

- Click the **right mouse button** or **0** in the view area to popup the editor chooser dialog and select "Controls" mode.
- Type key **3** to popup the type dialog and select a signal type.
- Move the mouse pointer to the location you want to place the signal and type **2**.

Repeat step 3 above to create more signals of the same type or follow step 2 to choose a different signal type.

How to edit parameters for an object?

- Select the proper [editing mode](#) first, then use key **4** or **6** to traverse the figures until the object you want to edit is highlighted (colored white).
 - Click middle mouse button to popup the dialog and edit the parameters of the selected object.
-

How to select figures?

Select the proper [editing mode](#) first, then use key **4** or **6** to traverse the figures. The selected object will be highlighted (colored white).

How to move figures?

Select the proper [editing mode](#) first, then move the cursor to the object that you want to move and click the left mouse button to choose the shape point. Then move the cursor to the new position and click the left mouse button again (or press return when cursor pointed at the new position).

How to measure the distance between two points?

Move the cursor to the starting point and then type key **v**, move the cursor to the ending point, a circle will be drawn as the cursor moves and the distance will be shown in the upper right corner of the window. To cancel this function, type key **v** again. The unit is feet in the display.

Appendix A: Example of paralib.dat File

```
/*
 * This file contains the parameters used by MITSIM
 */

# Following parameters transfer units in the British system into
# metric system.

[Native Length to Meter]           = 0.3048
[Native Speed to Meters per Second] = 0.4470
[Native Density to Vehicles per Kilometer] = 0.6214
[Native Flow to Vehicles per Hour]   = 1.0000
[Native Travel Time to Minute]       = 1.0000
[Native Demand to Vehicles per Hour] = 1.0000

# LEGEND UNITS

[Density Label]      = "Density (vpm)"
[Speed Label]        = "Speed (mph)"
[Flow Label]         = "Flow (vph)"
[Occupancy Label]    = "Occupancy (%)"

# THRESHOLDS FOR VIEWING GRAPHICAL OBJECTS (optional)

% Show the objects when pixels per meter is greater than the
% value provided here.

[Resolution] = {
  1 # lanes marks
  2 # lane type, vehicles, sensors and signals
  6 # vehicle lane change indicators
 10 # vehicle labels
}

[FontSizes] = {
  100 120 140 160 180
}

# ROUTE CHOICE PARAMETERS FOR UNGUIDED AND GUIDED VEHICLES

% MITSIM uses the following parameters in its logit route choice
% models to determine the paths that drivers will choose. Vehicles
% may be guided or unguided and may have predefined or dynamically
% computed paths.

% The probability that a driver will choose a certain route is
% determined using a logit choice model, whereby drivers judge the
% utilities of alternative paths. The utility of a path is
% a function of its travel time, calculated based on either
% historical information or real-time link travel times, depending
% on whether the vehicle is guided or not. The first column is
% percentage of each type of vehicles, and the 2nd column is the
% parameter coefficient of travel time on a path choice in the
% logit model (travel time ratio with respect to the travel time
```

```

% on shortest path).

[Route Choice] = {
%   Fraction   Beta
%   1.0        -5.0
%   0.0        -5.0
}

% When two paths have common sections, drivers may not consider one of
% the feasible paths. The Commonality Factor is a penalty applied to a
% path to correct for this error such that the driver will consider
% each path option as distinct and separate choices.

[Commonality Factor] = -1.0

% Because drivers on a freeway are less likely to choose a path
% that requires exit from the freeway, a Freeway Bias factor is
% applied in the logit model such that the probability that a
% driver will choose routes requiring exit from the freeway is
% less.

[Freeway Bias]      = 1.0      # Travel time factor

% In the route generation model, the routes for vehicles without
% pre-specified paths are generated at each intersection. In the
% route switching model, vehicles are assigned pre-specified paths,
% but may switch to alternative routes at intermediate nodes. In
% both cases, there is a diversion penalty is applied to alternative
% routes to simulate the phenomenon that drivers are less inclined
% to divert from their current, or habitual, routes.

[Diversion Penalty] = {
%   5.0 # in route generation model (minutes)
%   1.0 # in route switching model (minutes)
}

% Not all feasible routes are considered by drivers. In MITSIM's
% vehicle routing models, path choices with travel times greater than
% the shortest path travel time multiplied by the Valid Path Factor
% are not valid and are thus not considered by drivers.

[Valid Path Factor] = 3.0      # compare to shortest path

% Drivers without a pre-specified path travel according to the link
% based route choice model (i.e. route generation model). They
% decide at each node which link will be taken at the downstream
% node of the current link. The Rational Link Factor serves two
% purposes. The first is to avoid discarding downstream links
% connecting to the destination that will take the driver to a node
% that has a travel time to the destination greater than that at
% at the upstream node when the difference is small. The second is
% to avoid the case where drivers choose unrealistic, circulating
% paths. There are two extremes:
%
% RLF = 1.0 (default); any link ending at a node with a cost label
% (i.e. travel time to destination) greater
% than the upstream node is ineligible, even

```

```

%             reasonable, comparable links
% RLF = 0.0; all links are elibible, even circulating, irrational
%             link choices
%
% The RLF condition looks like so:
% if RLF*Label(j) <= Label(i) then consider the link
% otherwise disregard the link
% where Label(j) is the cost associated with the downstream node
% of the link in question and Label(i) the cost of the shortest
% path from the upstream node of the link in question
%
% note: values of RLF less than about 0.5 have essentially the
% same effect as RLF = 0

[Rational Link Factor] = 0.5

# PARAMETER FOR INFORMATION UPDATE

% This parameter is used to combine the new travel time with the
% earlier information when travel time table are updated periodically
% based on prevailing traffic condition. i.e.
%
% updated = (1-r) * old + r * new
% where r=[Path Alpha]

[Path Alpha] = 0.75

# UPDATE STEP SIZE (REACTION TIME RELATED)

% Driver behavior (e.g. responses to stimuli) is updated periodically
% during simulation depending on the state of the vehicle (i.e.
% depending on whether the vehicle is decelerating, accelerating,
% travelling at uniform speed, or is stopped). The update time
% step sizes are as follows:

[Update Step Sizes] = { # normal distributions (seconds)
%   mean  stdev lower upper
%   0.5   0.0   0.5   0.5   # decelerating
%   1.0   0.0   1.0   1.0   # accelerating
%   1.0   0.0   1.0   1.0   # uniform speed
%   0.5   0.0   0.5   0.5   # stopped vehicle
}

# VEHICLE LOADING PARAMETERS

% The vehicle loading parameters govern the way vehicles are
% entered into the network from pretrip queues at each entry
% link. If the time since the last vehicle entered the
% simulation and left the queue is greater than the loading
% headway parameter below, then the next vehicle in the
% queue is eligible for entry. The coefficient for the queue
% (Beta) is a parameter describing the entry speed of the
% vehicle. The entry speed will be some value between the
% mean speed on the entry link and the desired speed.
% Initially, the entry speed was some value between the
% 'speed at capacity' parameter value and the desired speed.

```

```

% Currently, MITSIMLab does not use the 'speed at capacity'
% value, but rather the average speed on the link.
%
% where:   queue = (queue length)/(number of lanes)
%          r = Beta * queue^2
%
% and if r > -5, then
%   entry speed = (mean speed)
%                + exp(r)*(desired speed - mean speed)
%
% else entry speed = mean speed

[Loading Model] = {
    0.60 # headway (seconds) (var:loadingHeadway)
    -0.25 # coef for queue (var:loadingQueueBeta)
    45.0 # speed at capacity (var:loadingSpeed)
}

# VEHICLE MIX AND ATTRIBUTES

% The mix of vehicles in the simulation will is assigned the
% following attributes.
%
% (1) Label (Type)
% (2) Typical length
% (3) Typical width
% (4) Percentage
% (5) delay factor at toll booth
% (6) ETC probability
% (7) Over height probability
% (8) HOV probability.

[Vehicle Classes] =
{
% 1      2 3      4 5      6 7      8
  "New Cars"      18 6 0.410 1.0 1.0 0.0 0.00
  "Old Cars"      18 6 0.400 1.0 1.0 0.0 0.00
  "Taxis"         18 6 0.100 1.0 0.0 0.0 1.00
  "Buses"         40 8 0.060 1.5 0.0 0.0 1.00
  "Trucks"        50 8 0.030 1.5 1.0 0.0 0.00
  "Trailer Trucks" 70 8 0.000 2.0 1.0 0.0 0.00
}

% The Min Heavy Vehicle Length is the minimum vehicle length for
% which vehicles will be classified as heavy.

[Min Heavy Vehicle Length] = 30 # feet

# CAR-FOLLOWING PARAMETERS

% The following parameters describe the boundaries of the car following
% regime. The following vehicle is in a car following regime (i.e.
% accelerates according to the car following models described below)
% when the time headway between the lead and following vehicles is
% between the CF Upper and Lower Bounds. The Min Response Distance is
% the minimum space headway between the lead and following vehicles for

```

```

% which the following vehicle must apply some acceleration (or
% deceleration).

[Min Response Distance] = 15 # feet
[CF Lower Bound] = 0.40 # seconds
[CF Upper Bound] = 1.36 # seconds, not used for KAZI's ACC model

% The General Motors (Herman) car-following model describes the
% acceleration of the following vehicle in a car following regime:
%
% Acceleration = alpha * (v ^ beta / dx ^ gamma) * dv + N(0, stddev^2)
%
% where v = speed of the following vehicle, dx = distance gap between
% the lead and following vehicles, and dv = speed difference between
% the lead and following vehicles. Below are the car following
% parameters used by MITSIM when using the General Motors CF model.
% In this model, stddev is 0.

[CF Parameters] = {
%   alpha beta gamma stddev
%   2.15 -1.67 -0.89 #0 # acc
%   1.55 1.08 1.65 #0 # dec
}

% Similarly, Kazi's car following model takes on the following
% functional form:
%
% Acceleration = alpha * (v ^ beta / dx ^ gamma)
%               * (k ^ rho) * (dv ^ lambda) + N(0, stddev^2)
%
% where v = speed of the following vehicle, dx = distance gap between
% the lead and following vehicles, k = the traffic density downstream
% of the following vehicle, and dv = speed difference between the lead
% and following vehicles. Below are the car following parameters used
% by MITSIM when using Kazi's CF model. In Kazi's new car following
% model, the stddev are used to create the bins in columns 4-5 in
% 'Driver Groups' and are not used by MITSIM directly.

[CF Kazi Parameters] = {
%   alpha beta gamma rho lambda stddev
%   0.0500 0.722 0.242 0.682 0.600 #0.825 # acc 0.0225 (original alpha)
%   -0.0550 0.000 0.151 0.804 0.682 #0.802 # dec
}

% Kazi formulates the free flow acceleration rate as:
%
% acc = b[2] * stimulus + N(0, stddev)
%
% where:
%
% Spd = speed of the subject vehicle
% Stimulus = b[3] + spdlmt + b[4] * front->Spd + b[5] * IsHeavy
%           + b[6] * IsFreeFlow - spd
%
% This model applies only if Spd is greater than b[0]. IsFreeFlow is
% defined as segment density and is less than or equal to b[1] vehicles
% per km per lane (level of service A, B, or C). All units are already

```

```

% in the metric system and no conversion is required.
%
% The stddev is indirectly used in creating col 6 in 'Driver Group'.

[FF Acc Params] = { # Spd is in meter/sec, acc is in meter/sec2

%   b[0]  b[1] b[2]   b[3]      b[4]      b[5]      b[6]      # std dev
      10.0  19  0.3091 -12.365  0.6182  -0.670  7.5974  #  1.134
}

# MERGING

% The merging model is used to describe driver merging behavior. A
% vehicle is merging if it is within some distance D2 downstream of the
% merging point (e.g. point where 2 lanes become 1). If a vehicle is
% some distance D1 upstream of the merging point, it will be merging if
% there is available capacity in the merging area (i.e. there are fewer
% than NCAP vehicles in the merging area defined by D1 and D2). If
% there exists an upstream vehicle in the freeway lanes when the
% vehicle in question is entering from an on-ramp, the entering
% vehicle may or may not accelerate to merge based in part on the
% probability P that the driver will perform an aggressive merge.

[Merging Model] = {
    100 # D1, upstream area (feet) (var: upMergingArea)
    200 # D2, downstream area (feet) (var: dnMergingArea)
    8   # NCAP, number of vehicles allowed
        # (var: nVehiclesAllowedInMergingArea)
    0.2 # P, probability of aggressive merge from ramp
        # (var: aggressiveRampMergeProb)
}

# LANE-CHANGING PARAMETERS

% Drivers are assumed to make a series of decisions to determine
% whether a lane change will be undertaken. The hierarchical decision
% process is modeled using the random utility approach.

[LC Entry Lane Check Rightness Prob] = 1.0

% The probability of starting a mandatory lane change or staying in the
% current lane is determined according to the following parameters.
% The model requires that the vehicle be greater than a distance 'lower
% bound' from the downstream node (or lane drop) requiring the
% mandatory lane change in order for the vehicle to be eligible for a
% mandatory lane change. The probability that a vehicle will change to
% the appropriate lane is given by:
%
% Prob = exp (-dis*dis/(delta^2))
%
% where: dis = (Xn - Xo) - 'lower bound'
%         where Xn = distance from vehicle to downstream node (or lane
%         drop) and Xo = distance of a critical location (e.g. final
%         exit warning)
%         delta = delta1 * [1.0 + alpha1*number of lanes +
%         alpha2*(segment density/jam density)]
%
%

```

```
% A vehicle must remain in a lane for a minimum of 1.0 seconds before
% considering another lane chane.
```

```
[LC Mandatory Probability Model] = {
    330.0      # lower bound, feet
    1320.0    # delta1, feet
    0.5       # coef for number of lanes, alpha1
    1.0       # coef for congestion level, alpha2
    1.0       # minimum time in lane
}
```

```
# GAP ACCEPTANCE MODEL FOR LANE CHANGING
```

```
% Once a driver has decided to change lanes, he/she will examine
% adjacent gaps and deem them acceptable or unacceptable for merging.
% Below are parameters used by MITSIM to describe driver gap
% acceptance.
```

```
% 1. Parameters in Kazi's LC gap model (see Kazi's PhD thesis)
```

```
%
% G(cr,lead,dlc)= critical lead gap for discretianary LC case
%                = exp[beta0-beta4*min(0,dVlead)+N(0,sigma^2)]
% G(cr,lag,dlc) = exp[beta0+beta3*min(0,dVlag)
%                    +beta4*max(0,dVlag)+N(0,sigma^2)]
% G(cr,lead,mlc)= critical lead gap for mandatory LC case
%                = exp[beta0+N(0,sigma^2)]
% G(cr,lag,mlc) = exp[beta0+beta3*min(0,dVlag)
%                    +beta4*max(0,dVlag)+N(0,sigma^2)]
%
```

```
% where:
```

```
% dVlead = speed difference between subject vehicle and lead
%          vehicle in gap in target lane
% dVlag   = speed difference between subject vehicle and lag
%          vehicle in gap in target lane
% scale   = scaling the critical gap
% alpha   = minimum gap (meter)
% lambda  = remaining distance to impact
%          = 10 [1 - 1/{1+exp(lambda * dis)}]
% beta0   = constant
% beta1   = coef of remaining distance to impact
% beta2   = coef of speed difference (m/s, d/s-u/s)
% beta3   = coef of negative speed difference (m/s, d/s-u/s)
% beta4   = coef of positive speed difference (m/s, d/s-u/s)
% sigma   = std dev of generic random term
```

```
[Kazi LC Gap Models] = {
% scale alpha lambda beta0 beta1 beta2 beta3 beta4 sigma
% Discretionary (Lead and Lag)
    1.00 0.0 0.000 0.508 0.000 0.000 0.000 0.420 0.488
    1.00 0.0 0.000 2.020 0.000 0.000 0.153 0.188 0.526
% Mandatory (Lead and Lag)
    0.00 0.0 0.000 0.384 0.000 0.000 0.000 0.000 0.859
    1.00 0.0 0.000 0.587 0.000 0.000 0.048 0.356 1.073
}
```

```
% 2. Parameters in Qi's gap acceptance model
```

```
%
```

```

% G(cr,lead,dlc) = max{alpha,alpha+beta1*Vn+beta2*(Vn-Vlead)}
% G(cr,lag,dlc) = max{alpha,alpha+beta1*Vlag+beta2*(Vlag-Vn)}
% G(cr,lead,mlc) = max{alpha,alpha+[beta1*Vn+beta2*(Vn-Vlead)]
%                               * [1-exp(-gamma*Xn^2)]}
% G(cr,lag,mlc) = max{alpha,alpha+[beta1*Vlag+beta2*(Vlag-Vn)]
%                               * [1-exp(-gamma*Xn^2)]}
%
% where:
% Vn      = speed of subject vehicle
% Vlead   = speed of lead vehicle in gap in target lane
% Vlag    = speed of lag vehicle in gap in target lane
% 0 scale = type specific scaler
% 1 gamma = distance parameter = 1 - exp(-gamma * dis * dis)
% 2 alpha = minimum gap (feet)
% 3 beta1 = coef of following vehicle's speed (spd, feet/sec)
% 4 beta2 = coef of speed difference (dv, feet/sec)

[Qi LC Gap Models] = {
% scale  gamma  alpha  beta1  beta2
% Discretionary (Lead and Lag)
    0.50  0.00  3.00  0.05  0.10
    0.50  0.00  5.00  0.10  0.30
% Mandatory (Lead and Lag)
    1.00  2.5E-5  3.00  0.05  0.10
    1.00  2.5E-5  5.00  0.10  0.30
}

# PARAMETERS FOR YIELDING PROBABILITIES

[LC Yielding Model] = { # currently yields to nosing vehicle only
    0.80  # previously not yielding
    1.00  # previously yielding
}

# PARAMETERS FOR NOSING AND YIELDING

% The following parameters are used to return the probability
% that a vehicle will nose in.
%
% Prob = Pmax*[{1+cos(pi*y^z)}/2]
%
% where y      = x/L where x=distance of vehicle from critical position
%              and L=length of link or view range
%           z   = beta0 + beta1*N + beta3*T
%           Pmax = maximum probability of nosing
% also, the probability will be updated every Thor seconds, the vehicle
% will wait no longer than 'max stuck time' to nose in, the vehicle
% is only eligible for nosing when less than 'max distance for nosing'
% and greater than 'min distance for nosing' from the critical position
% before which lane change must be completed.

[LC Nosing Model] = {
    1.0 # Thor, time horizon (seconds) for constant acc
    0.5 # beta0, constant
    0.6 # beta1, coef for number of lanes, N
    0.1 # beta3, coef for time (minutes) since tagged, T
    0.2 # max prob: case of ramp lane drop, incident, red LUS, etc
}

```

```

    1.0 # max prob: case of connection to next link on path
    300.0 # max yielding time (seconds)
    180.0 # max stuck time (seconds)
    600.0 # max distance for nosing
    40.0 # min distance for nosing
}

% Kazi's Nosing model directly calculates the probability to nose when
% distance from a point is less than a threshold.
%
%  $P = 1 / (1 + \exp(-u))$ 
%
% where:
%
%  $u = b[0] + b[1] * \min(0, \text{lead\_rel\_spd}) + b[3] * \text{rem\_dis\_impact} +$ 
%  $b[4] * \text{gap} + b[5] * \text{nlanes}$ 
%
%  $\text{rem\_dis\_impact} = 10 - 10 / (1 + \exp(b[2] * \text{dis}))$ 
%
% where:
% lead_rel_spd = speed of subject vehicle subject to speed of lead
%               vehicle in gap, i.e.  $V_n/V_{\text{lead}}$ 
% rem_dis_impact = remaining distance impact = f(dis, the critical
%               position before which lane change must be completed)
% gap = sum of the lead and lag gaps
% nlanes = number of necessary lane changes in order to
%         connect to next link on vehicle's path

[LC Kazi Nosing Model] = {
% b[0] b[1] b[2] b[3] b[4] b[5]
% -3.159 0.313 -0.027 2.050 0.028 0.6
}

# PROBABILITY OF YIELDING TO OTHER VEHICLES

% The following are the probabilities used by MITSIM that a driver will
% yield to no other vehicles, up to 1 other vehicle, up to 2, or up to
% 3 other vehicles.

[MLC Yielding Probabilities] = {
    0.13 # None
    0.71 # Up to 1
    0.13 # 2
    0.03 # 3
}

# PARAMETER FOR DISCRETIONARY LANE CHANGE

% The following are parameters used by MITSIM governing whether a
% driver will undertake a discretionary lane change.

[LC Discretionary Lane Change Model] = { # Do not change the order
    5.00 # 0 Step size for update lane speed
    0.50 # 1 Prob of DLC given not DLC in last step
    0.90 # 2 Prob of DLC given DLC in last step
    0.80 # 3 Impatient lower bound
    1.00 # 4 Impatient upper bound
}

```

```

0.85 # 5 Slower lead threshold
0.10 # 6 Speed threshold
0.85 # 7 Acceleration threshold
300.00 # 8 Speed ahead looking distancs
3.00 # 9 Minimum time in lane (same direction)
10.00 # 10 Minimum time in lane (different direction)
10.00 # 11 Maximum speed difference between neighbor lanes
(mph)
}

% In Kazi's lane changing model, drivers not in a mandatory lane change
% situation or not responding to a mandatory lane change are assumed
% to enter the discretionary lane change decision process at every
% discrete point in time. The driver first decides whether conditions
% in the current lane are satisfactory, second whether other lanes are
% preferable to the current lane, third which lane to select, and
% finally whether the lead and lag gaps in the selected lane are
% acceptable. Each decision is made based on the random utility
% approach, where the probability of selecting a discrete choice is
% determined based on their relative utilities.

% Each driver's desired speed is assumed to be a function of the
% average speed of the vehicles ahead of the subject vehicle, or the
% desired speed =  $V^* = (\text{avg speed ahead}) * (B11)$ , where B1 is the model
% parameter coefficient of the average speed ahead.
%
% Prob(driving conditions not satisfactory)=
%  $1/[1+\exp\{-(B2+B3*(V_{\text{sub}}-V^*)+B4*d_{\text{heavy}}+B5*d_{\text{tail}})\}]$ 
%
% where  $V_{\text{sub}}$  = speed of subject vehicle
%  $d_{\text{heavy}}$  = 1, if subject is a heavy vehicle, 0, otherwise
%  $d_{\text{tail}}$  = 1, if subject being tailgated, 0, otherwise
%
% Prob(choose left (or right) lane|driving conditions not
% satisfactory)=
%  $1/[1+\exp\{-(B6+B7*(V_{\text{lead}}-V^*)+B8*(V_{\text{front}}-V^*)+B9*(V_{\text{lag}}-V_{\text{sub}}))\}]$ 
%
% where  $V_{\text{lead}}$  = speed of lead vehicle in lane being considered
%  $V_{\text{front}}$  = speed of vehicle directly in front of subject
%  $V_{\text{lag}}$  = speed of lag vehicle in lane being considered
%
% The 'gap behind threshold for tailgate dummy' is the gap length
% between the subject and its following vehicle below which the
% following vehicle will be considered to be tailgating. In densities
% above the 'density threshold for tailgate dummy', no vehicles will be
% considered to be tailgating.

[LC Kazi DLC Model] = {
0.7682 # B1, Segment average speed for desired spd model
0.2250 # B2, Constant for driving conditions unsatisfactory
-0.0658 # B3, Speed less desired speed
-3.1472 # B4, Heavy vehicle dummy
0.4226 # B5, Tailgate dummy
10 # gap behind threshold for tailgate dummy (meters)
19 # density threshold for tailgate dummy
2.0800 # B6, constant for utility of an adjacent lane
0.0337 # B7, (lead_speed_in_target_lane - desired_speed)

```

```

-0.1520      # B8, (front_speed_in_same_lane - desired_speed)
0            # B9, subject_speed - lag_speed
0.0000      # previous DLC state dummy
}

# START-UP DELAYS

% Drivers in a queue will resume movement after some delay depending
% on their position in a queue, with some limiting maximum delay.
% Below are the mean delay values for vehicles in particular positions
% in a queue. Start up delay for vehicles further back in the queue
% than those listed are assumed to have zero start up delay.

[Start Up Delay Parameters]      = { # seconds
    2      # maximum delay
% PositionInQueue Mean
    0      1.5
    1      1.2
    2      1.0
    3      0.9
    4      0.8
    5      0.6
    6      0.5
    7      0.4
}

% The Familiarity parameter describes driver familiarity with the
% network which can affect driver behavior, where Familiarity = 1.0
% means all drivers are completely familiar with the driving
% environment.

[Familiarity]      = 0.9

# RESPONSE TO CONTROLS

% The Visibility Scaler is a measure of driver visibility of traffic
% controls, where 1.0 means all drivers may see all controls clearly.

[Visibility Scaler] = 1.0

% Compliance Rates determine the extent to which drivers heed traffic
% controls, where 1.0 means that all drivers obey the controls.

[Compliance Rates] = {
    1.0 # Traffic signals only
    1.0 # Portal signals and traffic signals only
    0.9 # Ramp meters, portal signals and traffic signals
    0.9 # Red LUS only
    0.5 # Yellow and red LUS
    1.0 # Lane use rules only
    1.0 # Lane change and lane use rules
    1.0 # Vehicle type related lane use message
    1.0 # Path related lane use message
    0.7 # Route guidance message
}

% When the signal is yellow, then, if the distance to the intersection

```

```

% divided by the MAX(current speed, minimum speed) (i.e. the time
% to reach the intersection) is greater than the 'Time to Treat
% Yellow as Red', then the vehicle will break to stop.

[Time To Treat Yellow as Red]      =  1.0      # seconds

% The Maximum Toll Booth Delay is the maximum delay to a vehicle at
% a toll booth.

[Maximum Toll Booth Delay]         = 10.0      # seconds

# PARAMETERS FOR HEADWAY VARIANCE

% A headway buffer is found for each vehicle and is a behavioral
% parameter that describes the aggressiveness of a driver for accepting
% a headway gap in lane changing, merging, and car-following. Some
% value between the lower and upper bound will be added to the minimum
% headway gaps for the population, which are constants also provided in
% this file.

[Headway Buffer Lower Bound]       =  0.1 # seconds
[Headway Buffer Upper Bound]       =  0.2 # seconds

# LIMITING PARAMETERS

% Jam Density is the maximum traffic density in vehicles per mile when
% speed is a minimum. Minimum Speed is the minimum vehicle speed on
% the network.

[Jam Density]                     = 210.0    # vehicles/mile
[Minimum Speed]                   =  5.0     # mph

% The next parameter is used in calculation of harmonic mean of speed.
% If a vehicle is stopped at the sensor, its speed is zero. The
% Detected Minimum Speed value below is used to avoid the problem of
% dividing by zero.

[Detected Minimum Speed]          =  5.0     # mph (must be > 0)

# DISTRIBUTION OF DRIVER GROUPS

% Speed limits come from JPL data. MaxAccScale is made up. Each row
% corresponds to the percentile specified in the first column. Each
% column is a variable:
%
% 0 = Max acceleration scale
% 1 = Max deceleration scale
% 2 = Normal deceleration scale
% 3 = Car-following acceleration add on (see also CF Kazi Parameters)
% 4 = Car-following deceleration add on (see also CF Kazi Parameters)
% 5 = Free-flow acceleration add on (see also FF Acc Params)
% 6 = Speed limit add on for desired speed
% 7 = Upper headway threshold (Kazi, was 1.36)
% 8 = Percentile (not read)

%[Driver Groups] =
%{

```

```

%      0      1      2      3      4      5      6      7      8
%      0.6  1.0000 1.0000 -0.4721 -0.3520 -0.3158 -4.67  1.1741 # 0.05
%      0.7  1.0000 1.0000 -0.2975 -0.2218 -0.1990 -1.73  1.4650 # 0.15
%      0.8  1.0000 1.0000 -0.1936 -0.1443 -0.1295 -0.20  1.7155 # 0.25
%      0.9  1.0000 1.0000 -0.1106 -0.0825 -0.0740  0.97  1.9470 # 0.35
%      1.0  1.0000 1.0000 -0.0361 -0.0269 -0.0241  1.98  2.1723 # 0.45
%      1.1  1.0000 1.0000  0.0361  0.0269  0.0241  3.05  2.4019 # 0.55
%      1.2  1.0000 1.0000  0.1106  0.0825  0.0740  4.06  2.6480 # 0.65
%      1.3  1.0000 1.0000  0.1936  0.1443  0.1295  5.33  2.9296 # 0.75
%      1.4  1.0000 1.0000  0.2975  0.2218  0.1990  6.71  3.2902 # 0.85
%      1.5  1.0000 1.0000  0.4721  0.3520  0.3158  8.94  3.9091 # 0.95
%}

```

```
[Driver Groups] =
```

```

{
%      0      1      2      3      4      5      6      7      8
%      0.6  1.00  1.00 -1.3564 -1.3187 -1.8654 -0.1911  1.7498 # 0.05
%      0.7  1.00  1.00 -0.8547 -0.8309 -1.1754 -0.0708  2.2737 # 0.15
%      0.8  1.00  1.00 -0.5562 -0.5407 -0.7649 -0.0082  2.5871 # 0.25
%      0.9  1.00  1.00 -0.3178 -0.3089 -0.4370  0.0397  2.8379 # 0.35
%      1.0  1.00  1.00 -0.1036 -0.1007 -0.1425  0.0810  3.0633 # 0.45
%      1.1  1.00  1.00  0.1036  0.1007  0.1425  0.1248  3.2814 # 0.55
%      1.2  1.00  1.00  0.3178  0.3089  0.4370  0.1661  3.5068 # 0.65
%      1.3  1.00  1.00  0.5562  0.5407  0.7649  0.2180  3.7578 # 0.75
%      1.4  1.00  1.00  0.8547  0.8309  1.1754  0.2745  4.0718 # 0.85
%      1.5  1.00  1.00  1.3564  1.3187  1.8654  0.3657  4.5979 # 0.95
}

```

```
# ACCELERATION TABLES
```

```
% Following are the acceleration and deceleration characteristics
% of the vehicle classes named earlier in the file.
```

```
% Converting factors used in next three tables:
% The following factors are used to convert the acceleration table
% units to the metric system.
```

```
[Acc Table Speed to Meters per Second] = 0.3048
[Acc Table Acc to Meters per Sq Second] = 0.3048
```

```
% In determining the maximum acceleration of a vehicle based on
% grade and speed, max acc = max acc (based on speed and vehicle
% type, from the tables below) - acc grade factor * grade
```

```
[Acceleration Grade Factor] = 0.305 # feet/sec/sec/grade
```

```
[Grade Scaler] = { 5 -2 2 }
[Speed Scaler] = { 5 20 20 }
```

```
% Global scalers: The acceleration scaler is multiplied by the
% acceleration rate for each vehicle type when the acceleration
% rate is read from this file. The deceleration scaler is
% similarly multiplied by the deceleration rate for each vehicle
% type.
```

```
[Acceleration Scaler] = 1.25 % global scaler
[Deceleration Scaler] = 1.00 % global scaler
```

% The following table lists the maximum accelerations of vehicle
 % types travelling in a given speed range.

[Maximum Acceleration] =

{
 % Each row corresponds to one vehicle class and each column to a
 % speed range (<20 20-40 40-60 60-80 >80 feet/sec)

% These values are based on level grade data of INTRAS document

%	11.0	11.0	10.0	5.0	3.0
%	6.0	6.0	6.0	3.0	2.0
%	3.0	2.0	1.0	1.0	1.0
%	3.0	2.0	1.0	1.0	1.0
%	1.0	1.0	1.0	1.0	1.0

% These were values used in previous runs

%	11.0	11.0	10.0	5.0	3.0
%	10.0	10.0	8.0	5.0	3.0
%	8.0	8.0	6.0	4.0	2.0
%	8.0	8.0	6.0	4.0	2.0
%	6.0	6.0	4.0	2.0	1.0

% These are the values from traffic engineering handbook (1992)

%	7.50	7.00	5.91	4.99	4.50	
%	7.50	7.00	5.91	4.99	4.50	# N/A
%	7.50	7.00	5.91	4.99	4.50	# N/A
%	1.60	1.45	0.89	0.47	0.40	
%	1.60	1.45	0.89	0.47	0.40	# N/A

% These are the values used in fresim

%	10.00	7.90	5.60	4.00	4.00	
%	8.71	5.17	4.43	2.89	2.00	
%	8.71	5.17	4.43	2.89	2.00	# N/A
%	2.80	2.50	1.50	1.00	0.50	
%	2.80	2.50	1.50	1.00	0.50	# N/A

% These are the values we use now

10.00	7.90	5.60	4.00	4.00
8.71	5.17	4.43	2.89	2.00
10.00	7.90	5.60	4.00	4.00
7.00	5.00	4.00	1.50	1.00
2.80	2.50	1.50	1.00	0.50
1.60	1.45	0.89	0.47	0.40

}

% The following table lists typical deceleration rates of
 % vehicles classes in a given speed range under normal
 % deceleration conditions.

[Normal Deceleration] =

{
 # Rows: vehicle types

```

# Cols: Speed (<20 20-40 40-60 60-80 >80 feet/sec)

7.8      6.7      4.8      4.8      4.8
7.8      6.7      4.8      4.8      4.8
7.8      6.7      4.8      4.8      4.8
7.8      6.7      4.8      4.8      4.8
7.8      6.7      4.8      4.8      4.8
7.8      6.7      4.8      4.8      4.8
}

```

% The following table lists the maximum deceleration rates of
 % each vehicle class travelling in a given speed range.

```

[Maximum Deceleration] =
{
# Rows: vehicle types
# Cols: Speed (<20 20-40 40-60 60-80 >80 feet/sec)

16.0    14.5    13.0    11.0    10.0
16.0    14.5    13.0    11.0    10.0
16.0    14.5    13.0    11.0    10.0
16.0    14.5    13.0    11.0    10.0
16.0    14.5    13.0    11.0    10.0
16.0    14.5    13.0    11.0    10.0
}

```

LIMITING SPEED BY VEHICLE TYPE AND GRADE

% The speed of each vehicle class is limited by the following
 % maximum values for given surface grades.

```

[Limiting Speed] = {
# Rows: vehicle types
# Cols: Grade (<-2 -2~0 0~2 2~4 >=4%)

%      200    200    200    200    200
%      200    200    200    200    200
%      100    98     84     70     57
%      100    98     84     70     57
%      85     84     50     32     23

200    200    200    200    200 # feet/sec
200    200    200    200    200
200    200    200    200    200
150    125    100     80     60
130    105     80     65     45
100     90     80     60     40
}

```

LANE SPEED RATIO

% The speed at which vehicles travel is observedly influenced by
 % the lane in which the vehicle is travelling and the number of
 % lanes. Where there are multiple lanes, vehicles typically
 % travel faster in the left lanes (Lane Speed Ratio > 1) and slower
 % in the right lanes (LSR < 1). The Lane Speed Ratios below
 % describe this phenomenon.

```
[Lane Speed Ratio] =  
{  
  # Rows: Total number of lanes  
  # Cols: Lane number (left to right)  
  
  1.00  
  1.06 0.94  
  1.06 1.01 0.93  
  1.05 1.05 0.97 0.93  
  1.04 1.06 1.01 0.95 0.94  
}
```

Appendix B: List of `paralib.dat` Parameters

paralib.dat Label (as found in <code>paralib.dat</code> parameter file)	Vector Variable Name (as used in MITSIMLab code)
[Native Length to Meter]	lengthFactor
[Native Speed to Meters per Second]	speedFactor
[Native Density to Vehicles per km]	densityFactor
[Native Flow to Vehicles per hr]	flowFactor
[Native Travel Time to Minutes]	timeFactor
[Native Demand to Vehicles per hr]	odFactor
[Density Label]	densityLabel
[Speed Label]	speedLabel
[Flow Label]	flowLabel
[Occupancy Label]	occupancyLabel
[Resolution]	resolution
[Font Sizes]	theFontSizes
[Route Choice]	routingParams
[Commonality Factor]	commonalityFactor
[Freeway Bias]	freewayBias
[Diversion Penalty]	diversionPenalty
[Valid Path Factor]	validPathFactor
[Rational Link Factor]	rationalLinkFactor
[Path Alpha]	pathAlpha
[Update Step Sizes]	updateStepSizeParams
[Constant State Time]	constantStateTime
[Loading Model]	loadingParams
[Min Heavy Vehicle Length]	heavyVehicleLength
[Vehicle Classes]	nVehicleClasses
[Min Response Distance]	minResponseDistance
[CF Lower Bound]	cfLower
[CF Upper Bound]	cfUpper
[CF Parameters]	accParams
[CF Kazi Parameters]	accParams
[FF Acc Params]	ffAccParams
[Merging Model]	mergingParams
[LC Entry Lane Check Rightness Prob]	lcEntryLaneCheckRightnessProb
[LC Mandatory Probability Model]	mlcParams
[Kazi LC Gap Models]	lcGapModels
[Qi LC Gap Models]	lcGapModels
[LC Yielding Model]	lcYieldingProb
[LC Nosing Model]	nosingParams

[LC Kazi Nosing Model]	kaziNosingParams
[MLC Yielding Probabilities]	mlcYieldProbs
[LC Discretionary Lane Change Model]	dlcParams
[LC Kazi DLC Model]	dlcKazi
[Start Up Delay Parameters]	sDelayMax, sDelayMu
[Familiarity]	familiarity
[Visibility Scaler]	visibilityScaler
[Compliance Rates]	complianceRates
[Time to Treat Yellow as Red]	yellowStopHeadway
[Maximum Toll Booth Delay]	maxTollBoothDelay
[Headway Buffer Lower Bound]	hBufferLower
[Headway Buffer Upper Bound]	hBufferUpper
[Jam Density]	jamDensity
[Minimum Speed]	minSpeed
[Detected Minimum Speed]	detectedMinSpeed
[Driver Groups]	rndParams
[Acc Table Speed to Meters per Second]	speedConverting
[Acc Table Acc to Meters per Sq Second]	accConverting
[Acceleration Grade Factor]	accGradeFactor
[Grade Scaler]	gradeScaler
[Speed Scaler]	speedScaler
[Acceleration Scaler]	accScaler
[Deceleration Scaler]	decScaler
[Maximum Acceleration]	vehicleLib
[Normal Deceleration]	vehicleLib
[Maximum Deceleration]	vehicleLib
[Limiting Speed]	vehicleLib
[Lane Speed Ratio]	laneSpeedRatio

Appendix C: Example of ctrlpara.dat File

```
/*
 * This file contains the parameters used by TMS
 */

# Following parameters transfer units in the British system into
# metric system.

[NativeLengthToMeter]           = 0.3048
[NativeSpeedToMetersPerSecond]  = 0.4470
[NativeDensityToVehiclesPerKilometer] = 0.6214
[NativeFlowToVehiclesPerHour]    = 1.0000
[NativeTravelTimeToMinute]       = 1.0000
[NativeDemandToVehiclesPerHour]  = 1.0000

# LEGEND UNITS

[DensityLabel]      = "Density (vpm)"
[SpeedLabel]       = "Speed (mph)"
[FlowLabel]        = "Flow (vph)"
[OccupancyLabel]   = "Occupancy (%)"

# THRESHOLDS FOR VIEWING GRAPHICAL OBJECTS (optional)
# Show the objects when pixels per meter is greater than the
# value provided here.

[Resolution] = {
  1 # lanes marks
  2 # lane type, vehicles, sensors and signals
  6 # vehicle lane change indicators
 10 # vehicle labels
}

[FontSizes] = {
  90 110 130 150 170
}

# AVERAGE VEHICLE LENGTH (CONVERTING OCCUPANCY TO DENSITY)

[AverageVehicleLength]      = 20.0      # feet

# PARAMETER FOR COMBINING TRAVEL TIME INFOR IN TRAFFIC PREDICTION
#
# This parameter is used to combine the new travel time with the
# earlier information. i.e.
#
#   updated = (1-r) * old + r * new
#
# If r is set to zero, a MSA like algorithm will be used instead.
# In other words,  $r = 1/(1+k)$ , where k is the iteration counter.
# This is equivalent to take the average of the result of all
# previous runs.

[Path Alpha] = 0.5
```

```
# When incident is cleaned, the capacity is restored in phases. Each  
# phase has a prespecified length of time.
```

```
[IncidentCapacityParameters] = {  
  0.2 # increase per step  
  150 # seconds per step  
}
```

Appendix D: Example of ctrllogic.dat File

```

/*
 * Specification of Control Logic (currently it contains incident
 * response logic only.
 */

# Some common parameters
{
  60.0      # delay of incident detection
  15       # minimum for speed limit sign (mph)
  5280.0   # maximum searching distance for signal setting (feet)
}

# =====
# LUS Logic
# =====

3 # Number of stages

# Delay time for each stage, defined as the time since incident
# has been detected. The delay time for the last stage is defined
# from the clearance of the incident.
# {
# -----
# CaseID   State   FromRefPoint   FromStart   ToRefPoint   ToEnd
# -----
# (A) CaseID is the sum of
#
#     (a) Lane blockage (1 = partial, 2 = complete)
#     (b) EVA (16 = upstream, 32 = downstream)
#
#     For clearance, the CaseID is 0.
#
# (B) State is Red = 1, Yellow = 2, or Green = 3.
#
# (D) FromRefPoint is the reference location for FromStart
#     defined as follows:
#
#     1 = upstream portal signal;
#     0 = incident;
#
# (C) FromStart is the starting point which is measured by the
#     distance upstream (+) or downstream (-) of FromRefPoint.
#
# (D) ToRefPoint is the reference location for ToEnd
#     (definition same as (C)).
#
# (E) ToEnd is the ending point which is measured by the
#     distance upstream (+) or downstream (-) of ToRefPoint.
# }

0 # Delay time from detection
{
  1      2      0 -500  1      1000      # 1

```

```

2      2      0 -500  0      2000      # 2
2      2      1   0  1      2000      # 3
17     2      0 -500  1      1000      # 4
33     2      0 -2000  0      2000      # 5
33     2      1   0  1      2000      # 6
34     2      0 -2000  0      2000      # 7
34     2      1   0  1      2000      # 8
}

```

```
15 # Delay from previous stage
```

```

{
1      1      0 -500  1          0      # 9
2      1      0 -500  0      1000      # 10
2      1      1   0  1      1000      # 11
17     1      0 -500  1          0      # 12
33     1      0 -1000  0      1000      # 13
33     1      1   0  1      1000      # 14
34     1      0 -1000  0      1000      # 15
34     1      1   0  1      1000      # 16
}

```

```
0 # Delay from incident clearance
```

```

{
0      3      0 -2000  1      2000      # 17
}

```

```

# =====
# VSLs Logic
# =====

```

```
2 # Number of stages
```

```

# Delay time for each stage, defined as the time since incident
# has been detected. The delay time for the last stage is defined
# from the clearance of the incident.
# {
# -----
# CaseID SpeedStart SpeedStep FromRefPoint FromStart ToRefPoint ToEnd
# -----

```

```
# (A) CaseID is sum of
```

```

#
# (a) Lane blockage (1 = partial, 2 = complete)
# (b) EVA (16 = upstream, 32 = downstream)
#

```

```
For clearance, the CaseID is 0.
```

```

# (B) Speed limit increase/decrease by SpeedStep from SpeedStart
# upto the default speed limit or minimum speed (15 mph)
#

```

```

# (C) FromRefPoint is the reference location for FromStart,
# defined as follows:
#

```

```

# 1 = upstream portal signal;
# 0 = incident;
#

```

```

# (D) FromStart is the starting point which is measured by the
# distance upstream (+) or downstream (-) of FromRefPoint.
#
# (E) ToRefPoint is the reference location for ToEnd
#
# (F) ToEnd is the ending point which is measured by the
# distance upstream (+) or downstream (-) of ToRefPoint.
# }

```

```

15 # Delay time from incident detection
{
    1   25   0   1       0  1   1000 # 1
    1   25  10   1   1000  1   2000 # 2
    2   15   0   0   -500  0   1000 # 3
    2   15  10   0   1000  1     0 # 4
    2   15   0   1     0    1  1000 # 5
    2   15  10   1   1000  1  2000 # 6
    17  25   0   0   -500  1  1000 # 7
    17  25  10   1   1000  1  2000 # 8
    33  15   0   0  -1000  0   1000 # 9
    33  15   0   1     0    1  1000 # 10
    33  15  10   0   1000  1     0 # 11
    33  15  10   1   1000  1  2000 # 12
    34  15   0   0  -1000  0   1000 # 13
    34  15   0   1     0    1  1000 # 14
    34  15  10   0   1000  1     0 # 15
    34  15  10   1   1000  1  2000 # 16
}

```

```

0 # Delay time from incident clearance
{ # This is the last stage for VSLIS (restore logic)
    0   45   0   0   -1000  1   2000 # 17
}

```

```

# =====
# Portal Closure Logic
# =====

```

```

2 # Number of stages

```

```

# Delay time for each stage, defined as the time since incident
# has been detected. The delay time for the last stage is defined
# from the clearance of the incident.

```

```

# {
# -----
# CaseID State DistanceFromIncident
# -----

```

```

# State of portal signal

```

```

# 0 = Blank
# 1 = Red
# 2 = Yellow
# 3 = Green

```

```

#
# Optional Flags
#

```

```
#      8 = Flashing
# }

15 # Delay time from incident detection
{
    34      1      1000
}

0 # Delay time from incident clearance
{
    0       3      1000
}
```

Appendix E: Example of the master.tms File

```
/*
 * TMS master file
 */

[Title] = "Modified Central Artery Network"

[Default Parameter Directory] = "/home/hippo/DynaMIT/DEMO"
[Input Directory] = "/home/hippo/DynaMIT/DEMO"
[Output Directory] = "/home/hippo/DynaMIT/DEMO/Output"
[Working Directory] = "/home/hippo/DynaMIT/DEMO/temp"

[Network Database File] = "networkrun.dat"
[GDS Files] = {
%   Filename      MinScale      MaxScale
}
[Parameter File] = "ctrlpara.dat"
[Control Logic File] = "ctrllogic.dat"
[Signal Plan File] = ""
[Control Logic] = 0
%   0 = None
%   1 = A1 incident response
%   2 = Gating logic
[Information] = 0
%   0 = Historical data
%   1 = Real time measurement
%   2 = Prediction

[Start Time] = 07:00:00
[Stop Time] = 08:12:00
[Step Size] = 0.1

[RollingStepSize] = 120
[RollingLength] = 1800
[NumOfDTAIterations] = 2
[DTAComputationalDelay] = 110

[Console Message Step Size] = 60

[Segments] = 3
%   0 = Direction
%   1 = Link type
%   2 = Density
%   3 = Speed
%   4 = Flow

[Signals] = 0x0
%   0x01 = Traffic signals
%   0x02 = Portal signals
%   0x04 = Variable speed limit signs
%   0x08 = Variable message signs
%   0x10 = Lane use signs
%   0x20 = Ramp meters

[Sensor Types] = 0x0
```

```
% 0x1 = Loop detectors
% 0x2 = AVI sensors
% 0x4 = VRC sensors
% 0x8 = Area sensors
```

```
[Sensor Color Code] = 3
% 0 = Count
% 1 = Flow
% 2 = Speed
% 3 = Occupancy
```

```
# [Randomize] = 0
# [Verbose] = 1
# [Nice] = 1
```

Appendix F: Simlab Constants

Controller Types

Generic controllers for signals and signs

- 0 Unsignalized (stop and yield signs only)
- 1 Pretimed
- 2 Activated (lookup table based)

These are for ramp metering

- 3 Local feedback ramp metering controller (Alinea)
- 4 Area ramp metering (Owen Chen's top level algorithm)
- 5 Bi-level ramp metering (Area + Local)
- 6 Flow control ramp metering (BP/B's design for CA/T)
- 7 Bottleneck ramp metering (MetaLinea)
- 8 Enhanced bottleneck (METALinea with variable bottleneck)

These are primarily for intersection signal control

- 10 Webster
- 11 Smith
- 12 Delay minimization
- 13 Bi-level delay minimization

Control Devices

- 0x01 Traffic signals (1)
- 0x02 Portal signals (2)
- 0x03 Variable speed limit signs (3)
- 0x04 Variable message signs (4)
- 0x0A Lane use signs (10)
- 0x0C Ramp meters (12)
- 0x10 Toll booth (16)

State for Traffic Signals

Signal colors

- 0x0 Blank
- 0x1 Red
- 0x2 Yellow
- 0x3 Green

For intersection traffic signals and signs, add the following value if the signals or signs indicate that the particular movement is protected or need to yield.

0x4 Arrow (protected)
0x8 Flashing (yield)

Note that a *flashing red* can be used to represent a stop sign.

For lane use signs, add the following value to the color if the sign also suggests for a lane change:

0x10 Right arrow (change lane to the right)
0x20 Left arrow (change lane to the left)

Variable Message Signs

The highest 16 bits are used to specify the signal type and actions suggested by the message; while the lowest 16 bits is used to specify the subject that the given message concerns (e.g. the ID of a link, a vehicle type, and so on). Following types of message signs are defined in MITSIMLab:

0x10000000 Lane use regulations related lane assignment
0x20000000 Path related lane assignment
0x30000000 Prescriptive routing message
0x40000000 Descriptive routing message

The information contained in each type of message are briefly described below:

Following lane changes actions are applicable to all types of messages.

0x00F00000 Number of lanes available for change (if any)
0x00000000 Stay in current lane
0x01000000 Use the given number of lanes on right
0x02000000 Use the given number of lanes on left
0x03000000 Use the lanes other than the current lane

If the following bit is set to *true*, the vehicles that *do not satisfy* the given conditions are subject to the specified action. If this bit is set to *false*, the vehicles meet the given conditions are subject to the action.

0x04000000 Negative operator

By default, whether a vehicle will comply to a message is determined according to an attribute that defines its compliance state, which is assigned randomly based on a predefined probability when it enter the network. However, this can be overwritten if the following bit is set.

0x08000000 Mandatory operator

All vehicles meet the condition will comply to the message regardless of their compliance attribute.

A message related to lane use rules use the last 4 digits to specify which types of vehicles are concerned with the given rule.

```
0x0000000F Vehicle class
0x0000FFFF Vehicle group
```

For a path related *lane assignment message*, the bits represented by the following value is used to specify the maximum number of links to be searched when determining whether a vehicle should be subject to the message or not.

```
0x000F0000 Number of links to be searched downstream for
              lane assignment message.
```

For a prescriptive routing message, an additional digit can be used for specifying the searching depth (compared to the lane assignment message). Specifically, the maximum number of links to be searched is given by:

```
0x00FF0000 Number of links to be searched downstream for
              route guidance message.
```

This allows the route to be specified for a particular reference point within 256 links downstream in the vehicle's path.

The recommended route is given by a *turn movement index* which specifies the downstream link that the subject vehicle should use. The turn movement index is stored in the following bits and should have a value of 1 for the right turn, 2 for straight, 3 for left and so on:

```
0x0F000000 Turn movement at the downstream node.
```

For a descriptive routing message, it is assumed that the information is accessible for a particular class(es) of vehicles (i.e. guided vehicles only or all vehicles) and vehicles comply to the message with a given probability. Furthermore, the message is time tagged and the complied vehicles response to the message (evaluate their current route and make route choice decision) only once when they view the message the first time. Information on vehicle class, compliance rate, and message's time tag are specified by corresponding digits as follows:

```
0x0F000000 Vehicle class
0x00F00000 Comply rate (%)
0x000FFFFF Time tag (seconds since middle night)
```

Surveillance Sensors

Sensor *Type*:

```
0x0100 Linkwide sensor
```

Sensor *Tasks* specifies what types of data a particular sensor will collect.

Aggregate traffic data can be sent to TMS or written into a **traffic sensor** data file:

```
0x0001 Flow
0x0002 Speed
0x0004 Occupancy
```

Data generated by **individual** probe vehicles is sent to TMS or written into **vehicle to roadside communication (VRC) sensor** data file *immediately* after the a vehicle pass the detector:

```
0x0010 ID
0x0020 Type
0x0040 Departure time
0x0080 Origin
0x0100 Destination
0x0200 Path
0x0400 Height
0x0800 Current speed
```

Finally, a sensor device (such as cameras) can take **snapshots** of vehicles which are in the detection zone and reports the following data items:

```
0x0010 ID
0x0020 Type
0x0400 Height
0x0800 Current speed
0x1000 Lane ID
0x2000 Position in lane
```

In order to produce above data, several things have to be done:

- The *tasks* field for a sensor device in the network database should have the value that corresponding to the data items to be collected by that sensor. For example, a task value of *0x0FF4* means that the sensor will collect average occupancy (*0x0004*) for each time interval and "all" information (*0x0FF0*) on individual vehicles that pass through the sensor.
- A vehicle has to be a [probe vehicle](#) in order to generate the VRC sensor reading. The number of probe vehicles can either be specified in the OD matrix directly or randomly assigned based on a percentage specified in the [simulation parameter](#) file.

- The output variable in MITSIMLab master file must indicate that sensor and VRC readings are of interest.
- Finally, a filename must be assigned for *Point Sensor File* and *VRC Sensor File* in the same master file.

The setting for the last two items above can also be done in *Simulation-Output* dialog box in the GUI.

Vehicle Type

A vehicle's type is specified by a two piece of information: class and group.

Vehicle Class

Vehicle class is specified by the right most hexadecimal digit of the type. For example:

```
0x1 High performance passenger cars
0x2 Low performance passenger cars
0x3 Buses
0x4 Trucks
0x5 Trailer trucks
```

Upto 15 types of vehicle classes can be defined. For each class of vehicles, parameters must be specified for [acceleration and deceleration profile](#), [maximum speeds](#), and several [other parameters](#).

Vehicle Group

Vehicle group represents the lane use privilege, availability of real time traffic information, and so on. Following group identification is current used:

```
0x0010 Small
0x0020 Low

0x0040 ETC
0x0080 HOV

0x0800 Guided
0x1000 Fixed path
0x2000 Probe vehicle
0x4000 Equipped with cellular phone
```

Output

Following constants are used to indicate what types of output are of interests for a simulation run.

```
0x0001 Vehicle log
```

0x0002 Sensor readings
0x0004 VRC readings
0x0010 Link travel times
0x0020 Segment travel times
0x0040 Segment statistics
0x0080 Queue statistics
0x0100 Travel time tables
0x0200 Vehicle path records
0x0400 Vehicle departure record
0x0800 Vehicle trajectories
0x1000 Output rectangular text
0x2000 No comments

The value can be specified directly in the master file or set through the output dialog box of the GUI.

Path Flags

Following values can be assigned to the variable SP Flag to specify how travel time information are used for routing vehicles.

0x001 Time variant path calculation
0x002 Calculate shortest path periodically
0x004 Update path table travel time periodically
0x008 Use existing (cached) shortest path table
0x100 Updated travel time used for pretrip plan

Appendix G: Multiple Sensor Step Sizes

In addition to the sensors that use the default step size defined in the **[Point Sensor Step Size]** line in **master.mitsim**, other sensors can use other step sizes. To use this feature, another line in **master.mitsim**, **[Point Sensor Step Sizes]**, must be created with a space-delimited, curly-bracket-enclosed set of additional step sizes, in seconds. In addition, if any of the additional sensor groups are supposed to output their data to a file, another line, **[Sensor Output Flags]**, must be added with a 1 for each sensor type that is supposed to output its data and a 0 for each one that isn't.

```
[Point Sensor Step Size]          = 120
[Point Sensor Step Sizes]       = { 1 30 }
[Sensor Output Flags]          = { 0 1 }
```

In this example, sensors will default to a 120-second step size, those marked with a 1 in **network.dat** will have a 1-second step size and won't have an output file, and those marked with a 2 will have a 30-second step size and will output to the file **sensor.out30**.

The marking of the sensors for non-default step sizes is done by adding a ":" followed by a number to the end of the sensor's line in **network.dat**, where the number corresponds to the list of non-default step sizes (here, 1 for 1-second, 2 for 30-second). These additions to **network.dat** have to be made manually; RNE won't create them, and will erase them if they're in an existing network file that is fed into it. Consequently, these additions should be made only after the user is finished with RNE for this network.

Appendix H: Route Choice Models in MITSIMLab

There are two route choice models in MITSIMLab: the route generation model and the route switching model. This document outlines the theory driving each model and the nature of their incorporation into MITSIMLab. In general, both models determine the probability that a driver will choose a link or path based on a multinomial logit model where the systematic utilities of each link or path in the choice set is a function of the driver's perceived travel time on a given link or path:

$$p(i) = \frac{\exp(V_i(t))}{\sum_{j \in C} \exp(V_j(t))}$$

where:

- $p(i)$ = probability of choosing link or path i ;
- $V_i(t)$ = systematic utility of link or path i ;
- C = set of all available links or paths to the driver's destination;

Once the probabilities of choosing each available link or path are computed, a random number is drawn to determine which link or path the driver will choose. A more detailed discussion of each model and its implementation in MITSIMLab follows.

Route Choice Theory

A driver's path in MITSIMLab is determined by one of two route choice models designed to simulate a driver's route choice decision-making process. The first of the route choice models is the route generation model, which applies only to vehicles that do not have predefined paths. The second model, the route switching model, captures the route choice behavior of vehicles with predefined paths.

Furthermore, drivers with or without predefined paths may be either guided or unguided. Unguided and guided drivers make their routing decisions based on different types of travel time information: historical and real-time link travel times, respectively. These travel times are computed by MITSIMLab to determine the systematic utilities of any link or path in the choice set. Guided, or informed, drivers have access to real-time link travel times, inasmuch as they might be delivered to the driver by a traffic management center via in-vehicle information technologies. Perceived travel times for unguided, or uninformed, drivers are calculated from historical link travel times. A flow diagram in Figure H-1 illustrates the different route choice behavioral types in MITSIMLab.

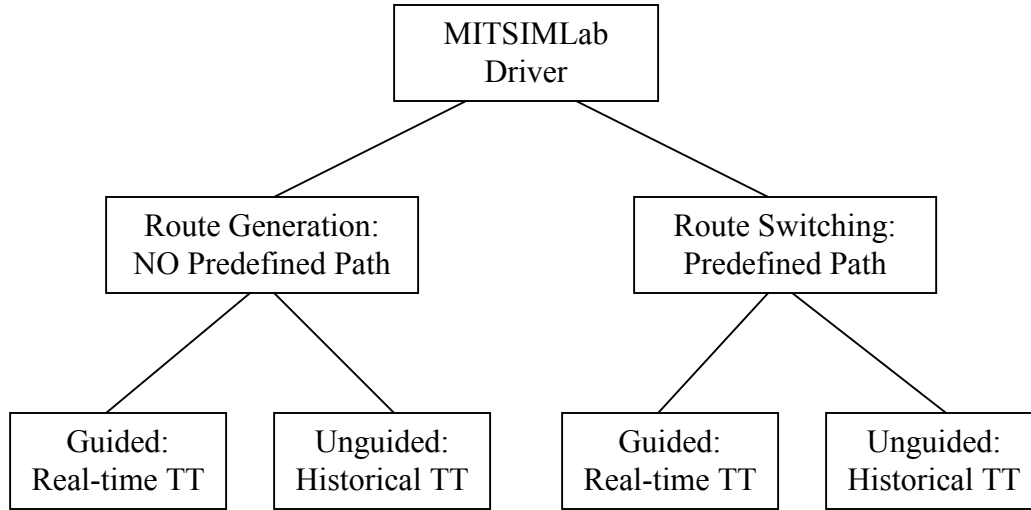


Figure 1. MITSIMLab route choice behavior classifications.

The Route Generation Model

In the route generation model, drivers have no predestined path. Rather, a driver's path is 'generated' on a link-by-link basis. Each time a vehicle enters a node j on any link ij , the driver will decide based on the probabilistic route generation model which link to take at the downstream node of the following link jk (see Figure 2).

The route generation model is mathematically expressed as follows:

$$p(l | k, t) = \frac{\exp(V_l(t))}{\sum_{m \in L_j} \exp(V_m(t))}$$

where:

- $p(l|k,t)$ = probability of choosing link l for a vehicle arriving at node j at time t ;
- $L_j(t)$ = choice set of available outgoing links from node k ;
- $V_l(t)$ = systematic utility of a route containing link l ;

The systematic utility $V_l(t)$ of a link is a function of the perceived travel time on link l , the perceived travel time on the shortest path from node n (the downstream node of link l) to the destination, and of a diversion penalty for leaving a freeway link for a non-freeway link.

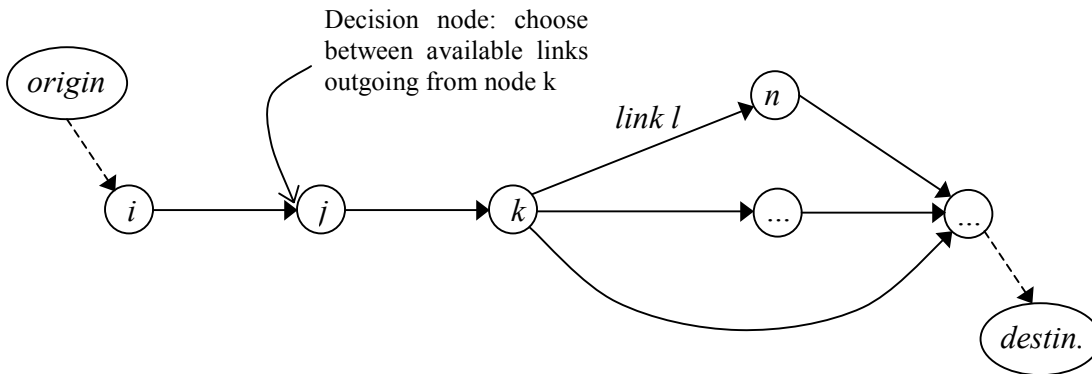


Figure H-2. An illustration of the route generation model.

The Route Switching Model

Drivers with pre-specified paths make their routing choices based on the route switching model. A vehicle that has a predefined, or habitual, path may ‘switch’ to alternative routes in light of events such as incidents or in response to travel delay information received from ATIS. Each time a vehicle enters a node j on any link ij , the driver will decide by the route generation logic which link to take at the downstream node of the following link jk (see Figure 3).

The route generation model is mathematically expressed as follows:

$$p(s | r, t) = \frac{\exp(V_s(t))}{\sum_{q \in S_r} \exp(V_q(t))}$$

where:

- $p(s|r,t)$ = probability of choosing path s for a vehicle arriving at node j via path r ;
- $S_r(t)$ = choice set of available outgoing paths from node k to the driver’s destination;
- $V_q(t)$ = systematic utility of path q ;

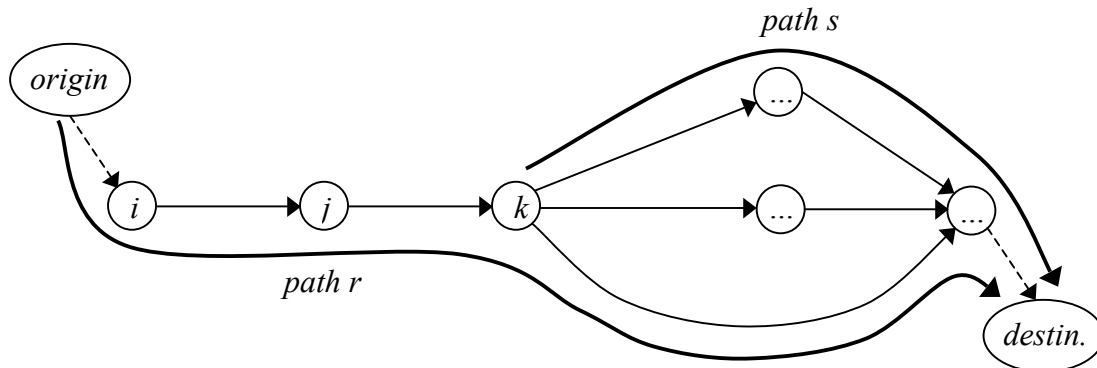


Figure H-3. An illustration of the route switching model.

The systematic utility $V_s(t)$ of a path is a function of the perceived travel time on path s and a diversion penalty if path s is different from the driver's current path r .

MITSIMLab Route Choice Implementation

The route choice model is carried out between several MITSIMLab source code files. Some of the more important files, and a description of their role in the route choice model, are listed below:

- `GRN/RN_Vehicle.cc` – contains the `OnRouteChoosePath` functions. Depending on whether a vehicle does or does not have a predefined path, these functions call either the `routeGenerationModel` function or the `routeSwitchingModel` function.
- `GRN/RN_Node.cc` – contains the `routeGenerationModel` and `routeSwitchingModel` functions, which calculate the utilities of available links or paths in a driver's choice set and dynamically generate the next link on a vehicle's path.
- `RN_Route.cc` – stores time invariant routing information.
- `RN_DynamicRoute.cc` – stores time variant shortest path information.
- `RN_LinkTimes.cc` – stores time variant link travel times

An understanding of the simulation logic leading into each of the route choice models is useful. Every time step (i.e. step size parameter specified in the `paralib.dat` file) in a MITSIMLab simulation, the `simulationLoop` function (`TS_Engine.cc`) is called. The `simulationLoop` function calls the `moveVehicles` function (`TS_Network.cc`), which is designed to 'physically' move the vehicles in the network based on their speed and acceleration. In the `moveVehicles` function, segment-by-segment and vehicle-by-vehicle, the `move` function (`TS_Vehicle.cc`) is called to move each vehicle in the network. From the `move` function, the logic takes two directions for vehicles with and without paths.

The utility of a link or path in MITSIMLab is primarily a function of path travel times, which are calculated from link travel times. When the `RN_LinkTimes.cc` file updates link travel times, it calls the `generalizeTravelTime` function stored in the `RN_Link.cc`, which applies a freeway bias to all freeway links. The travel times on freeway links are divided by the freeway bias, a number greater than or equal to one, in order to reduce the perceived travel time on freeway links, thus capturing drivers' apparent preference for travel on limited access freeways as opposed to other roadway types.

Figure H-4 illustrates the flow of control between various MITSIMLab files and functions. Each node in the diagram contains a function name and the name of the file where the function may be found. Each node represents a key function leading up to the route choice logic. Each arrow in the flow diagram represents a call from the function, or node, where the arrow originates to the function in the node at the arrow's head. A more detailed description of each model, beginning in the `move` function, follows. (Note: References to functions in the text may be followed by the file name in parentheses.)

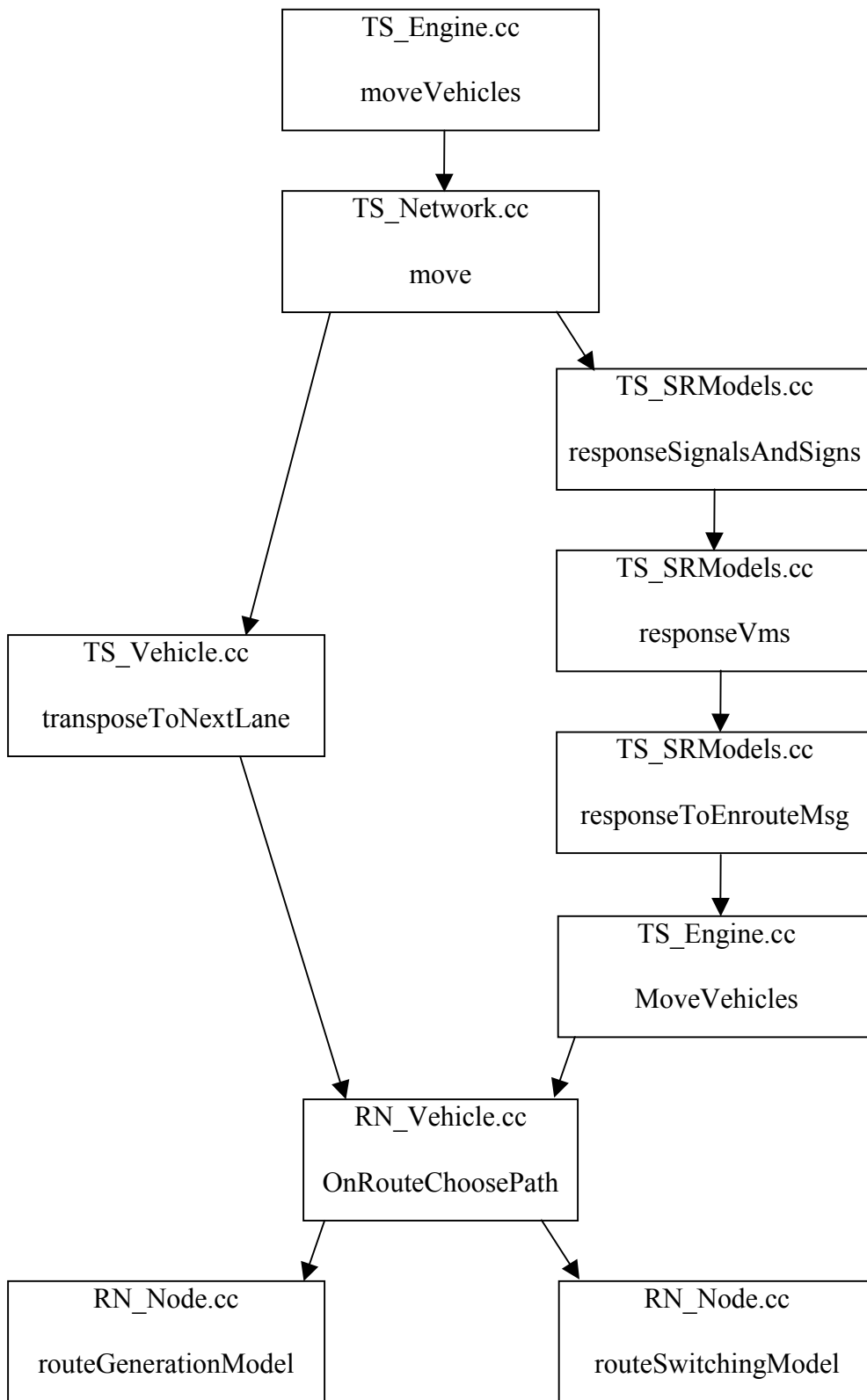


Figure 4. The relationship of route choice to the MITSIMLab file structure

The Route Generation Model

In the route generation model, vehicles decide at the upstream node of each new link which link will be taken at the downstream node of that link. The logic begins in the `transposeToNextLane` function (`TS_Vehicle.cc`), which is called from the `move` function IF the vehicle has reached the end of a segment. Each time a vehicle approaches a node, the `transposeToNextLane` function is called, marking the vehicle's entrance to a new link. MITSIMLab takes this opportunity to update the next link in the vehicle's path.

If the new link that the vehicle is about to enter is not the last link in the vehicle's path, `transposeToNextLane` will call the `OnRouteChoosePath` function (`RN_Vehicle.cc`). If the vehicle has no path, the `routeGenerationModel` (`RN_Node.cc`) will be called from `OnRouteChoosePath`. The `routeGenerationModel` retrieves the vehicle's routing information via the `routingInfo` function (`RN_Vehicle.cc`). The routing information is termed either `theGuidedRoute` or `theUnGuidedRoute` and determines the travel time table that will be used to determine the driver's perceived travel time in the choice model. If a guided vehicle has not accessed any information (e.g. from VMS), then the vehicle will travel according to `theUnGuidedRoute`, as will all unguided vehicles.

Once the routing information has been retrieved, control is passed back to the `routeGenerationModel`. Here, the utility computations begin with the calculation of `cost0`, the travel time on the shortest path from the downstream node of the new link to the vehicle's destination. The `routeGenerationModel` makes a call to `dnRouteTime` (`RN_Route.cc`), which checks whether the destination node exists. If so, another `dnRouteTime` function (`RN_DynamicRoute.cc`) is called. From `dnRouteTime`, a call is made to `upRouteTime` (`RN_DynamicRoute.cc`), which is designed to retrieve the travel time on the shortest path from the upstream node of the new link to the destination.

In `upRouteTime`, a variable `dt` is calculated by the following equation:

$$\text{offset} = [(\text{current time} - \text{infoStartTime}) / (\text{infoPeriodLength})] + 0.5$$

Depending on the value of `dt`, an offset is determined and a read function (`routeTimes_.read(offset)`) is called, which reads the shortest path travel time from the time table file at a location determined by the value of `offset`.

The `dnRouteTime` function then makes a call to `linkTime` (`RN_LinkTimes.cc`), which returns the expected link travel time at the given entry time to a link. If there are more than one `infoPeriods`, the variable `dt` described above is calculated again, and the expected link travel time returned by the function is dependent on `dt`. Otherwise, if there is only one `infoPeriod` (i.e. the MITSIMLab default value), the default `linkTimes` are returned. The default `linkTimes` values are determined prior to the start of the simulation and are calculated as the link length divided by the free flow speed (`calcStaticInfo` in `RN_Link.cc`). The difference between the travel time on the shortest path from the upstream node of the new link to the destination and the expected

travel time on the new link is returned to the `routeGenerationModel` and assigned to the variable `cost0`.

The variable `cost0` (i.e. the travel time on the shortest path from the downstream node of the new link to the destination) will be used along with the valid path factor to determine whether a link will be considered in the choice set. For each link outgoing from the downstream node of the new link, the following steps are performed in `routeGenerationModel`:

1. Call the `upRouteTime` function to determine the travel time on the shortest path from the upstream node of the link under consideration to the destination. The travel time is assigned to the variable `cost`.
2. IF the shortest path travel time for the link in question is less than the product of the valid path factor and `cost0`, THEN continue. Otherwise, this link is not included in the driver's choice set, so consider the next link.
3. The `linkTime` function is called to determine the expected travel time on the link in question. Assign the travel time to the variable `cost1`.
4. The difference `cost - cost1` (i.e. the travel time on the shortest path from the downstream node of the link in question to the destination) is assigned to the variable `cost2`.
5. If the vehicle is currently on a freeway link (i.e. the new link is a freeway link) and the link in question is not a freeway link (e.g. an exit ramp), then a third cost `cost3` is assigned the value of the diversion penalty for the route generation model.
6. The perceived `cost`, then, of using the link in question is the sum of `cost1`, `cost2` and `cost3`. The utility is the product of the coefficient `beta` and the ratio of `cost` to `cost0`, the travel time on the shortest path from the downstream node of the new link to the destination. The exponential of the utility is calculated as follows:

$$\exp [(\text{beta} * \text{cost}/\text{cost0})$$

where:

`cost` = (expected TT on link) +
(shortest path TT from downstream node of link to destination) +
(diversion penalty, if it applies);

`cost0` = (shortest path TT from upstream node of link to the destination).

7. A variable `sum` keeps track of the sum of the exponentials of the utilities of all links in the choice set (i.e. the denominator in the logit model probability calculation).

The `routeGenerationModel` calculates the probability of choosing each link as the ratio of the utility to the sum of the utilities of all available links and draws a random number

to determine the next link in the vehicle's path. The chosen path is then appended to the vehicle's path index via the `setPathIndex` function.

The Route Switching Model

There are two reasons why a vehicle with a predefined path might consider alternate paths (i.e. use the route switching logic). First, a vehicle will consider switching to a new path each path step size. The default path step size in MITSIMLab is 86,400 seconds, or one day. This day-by-day path reconsideration is meant to capture each driver's long-term route choice behavior given the day-to-day evolution of travel times in the network. A driver will also enter the route switching decision-making process in response to information such as signing (e.g. a VMS message indicating delays on the driver's path) and events (e.g. an incident impeding flow on the current path). The latter scenario is the focus of this discussion.

From the `move` function, if the vehicle has moved since the last iteration (i.e. time step), a call is placed to the `responseSignsAndSignals` function (`TS_SRModels.cc`). In the `responseSignsAndSignals` function, while there is a control device and it is within visibility, a different action is taken depending on the control type. For the case of VMS, a call is placed to the `responseVms` function (`TS_SRModels.cc`). In the `responseVms` function, different actions are taken depending on the type of VMS signing. For VMS signs of type `SIGN_ENROUTE`, the `responseToEnrouteMsg` function (`TS_SRModels.cc`) is called.

Recall that vehicles with paths will consider alternate paths in response to travel information. MITSIMLab uses the `ATTR_ACCESSED_INFO` attribute to tag vehicles that have accessed travel information. A vehicle with the `ATTR_ACCESSED_INFO` attribute 'turned off' will not enter the `routeSwitchingModel` function described below. The `ATTR_ACCESSED_INFO` attribute can be set in one of two functions: `responseToEnrouteMsg`, which is accessed via the sequence of events described above, or `changeRoute`, which is discussed next. In `responseToEnrouteMsg`, the attribute is set unconditionally and a call is placed to the `changeRoute` function (`TS_VehicleLoader.cc`).

MITSIMLab requires an encounter with a VMS message of type `SIGN_ENROUTE` in order for the `ATTR_ACCESSED_INFO` attribute to be set. In a system that is not VMS or beacon based, however, the requirement is bypassed in the `changeRoute` function, and the `ATTR_ACCESSED_INFO` attribute is 'turned on'. A call is then placed to the `OnRouteChoosePath`.

In the `OnRouteChoosePath` function, access to the `routeSwitchingModel` function is dependent on either of two conditions. IF the vehicle's path is not fixed (in general, it is not) OR if the vehicle has accessed information (i.e. the `ATTR_ACCESSED_INFO` attribute is 'turned on'), then the `routeSwitchingModel` is called. Otherwise, the vehicle advances on its current path unless it has reached its destination.

Like the `routeGenerationModel`, the `routeSwitchingModel` retrieves the vehicle's routing information via the `routingInfo` function (`RN_Vehicle.cc`). Like before, the routing information determines the travel time table that will be used to determine the driver's perceived travel time in the choice model. The `routeSwitchingModel` completes the following steps to generate a choice set:

1. Consider all paths to which the vehicle's current link belongs. If the path reaches the vehicle's destination, the vehicle is included in the choice set.
2. For each path connecting to the destination, a `cost` function (`RN_PathPointer.cc`) is called. The `cost` function, in turn, calls a `travelTime` function (`RN_Path.cc`), which loops through all links in the path not yet traveled and sums up the travel times on those links.
3. The link travel times are retrieved via a call to the `linkTime` function (`RN_LinkTimes.cc`), which reads the travel times from a time table or accepts the default, free flow travel times depending on the value of `dt` (which is a function of `infoPeriods`, `infoStartTime`, etc.) in the same manner as was described previously for the route generation model.
4. In the loop mentioned in step 2, the `routeSwitchingModel` keeps track of the minimum cost of all paths in the choice set in the variable `smallest`.

Once the four steps above have been completed, a path choice set has been generated. The next step in the `routeSwitchingModel` function is to calculate the utilities to be used in the logit choice model. If there is more than one path in the choice set, the utility parameters `beta` (the coefficient of the ratio of the path cost to the minimum path cost `smallest`) and `alpha` (the coefficient of the commonality factor) are retrieved and the following steps are performed for all paths in the choice set:

1. If the path in question is different from the vehicle's current path, the variable `cost` is assigned the value of the diversion penalty. Otherwise, `cost = 0`.
2. The path cost determined in steps 2 and 3 of the choice set generation procedure described above is added to `cost`.
3. The utility of the path is equal to the sum of the product of the cost coefficient `beta` and the ratio of the path cost to the minimum path cost and the product of the commonality coefficient `alpha` and the commonality factor. The exponential of the utility is calculated as follows:

$$\exp [(\text{beta} * \text{cost} / \text{smallest}) + \text{alpha} * \text{commonality factor}]$$

where

`cost` = (TT on the path in question);

`smallest` = (TT on shortest path from current position to the destination).

The commonality factor is calculated in `calcPathCommonalityFactors` (`RN_Link.cc`) as follows:

$$cf = \log \left(\frac{\sum_p \sum_l \delta_p N_l L_l}{\sum_p \sum_l \delta_p L_l} \right)$$

where:

\sum_p = summation over all paths p connected to node n ;

\sum_l = summation over all links l from the current position and the destination;

δ_p = 1, if path p is connected to node n ; 0, otherwise;

N_l = number of paths sharing link l ;

L_l = length of link l .

4. A variable `sum` keeps track of the sum of the exponentials of the utilities of all paths in the choice set (i.e. the denominator in the logit model probability calculation).

The `routeSwitchingModel` calculates the probability of choosing each link as the ratio of the utility to the sum of the utilities of all available paths and draws a random number to determine the vehicle's path. The chosen path is set via the `setPath` function.