DRAFT

Revised 11/2002

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

March 2001

Massachusetts Institute of Technology

Intelligent Transportation Systems Program

TABLE OF CONTENTS

TABLE OF CONTENTS **2**

OVERVIEW **4**

1. INTRODUCTION **4**

1.1 INSTALLING MITSIMLAB 4
1.2 RUNNING MITSIMLAB 5
1.3 RUNNING INSTRUCTIONS FOR MITSIMLAB 6

2. MITSIMLAB INPUT FILES **11**

2.1 MASTER FILES 11
2.1.1 Master.mitsim *13*
2.1.2 The `master.smc` file *17*
2.1.3 The `master.tms` file *18*
2.2 PARAMETER FILES 20
2.2.1 The `paralib.dat` file *20*
Example of `ctrllogic.dat` File *42*
2.3 THE NETWORK FILE 45
2.1.5.1 Network Data *45*
2.3.1 Nodes *45*
2.3.2 Links, Segments and Lanes *46*
2.3.3 The Lane Connectors *48*
2.3.4 Turn Prohibitors *49*
2.3.5 Sensors *49*
Traffic control devices *51*
Toll Plazas *52*
2.3.6 The Bus Stops *52*
2.4 DEMAND FILES 53
2.4.1. The demand.dat File *53*
2.4.2 The Path Definition File (`path.dat`) *56*
2.4.3 Link travel times *57*
2.5 TRAFFIC CONTROLLER ..... **ERROR! BOOKMARK NOT DEFINED.**
2.6 THE `INCIDENT.DAT` FILE 58
2.7 TRANSIT INPUTS 83
2.7.1 The Transit Network Representation File (`route.dat`) *83*
2.7.2 The Schedule Definition File (`schedule.dat`) *84*
2.7.3 The Bus Run Definition File (`run.dat`) *86*
2.7.4 The Bus Assignment File (`bus.dat`) *87*
2.7.5 The Transit Demand File (`passenger.dat`) *88*
2.7.6 The Priority File (`priority.dat`) *89*

3 MITSIMLAB OUTPUT FILES **91**

3.1 THE `SEGSTATS.OUT` FILE 91
3.2 THE `SEGTIME.OUT` FILE 92
3.3 THE LINK FLOW AND TRAVEL TIME DATA FILES 93

3.4 THE LINKTIME_MITSIM.OUT FILE 94
3.5 THE TRANSIT TRAJECTORY FILE 95
3.6 THE BUS STOP ARRIVAL FILE 95

4. THE ROAD NETWORK EDITOR (RNE) 96

HOW TO GET ON-LINE HELP? 96
HOW TO START THE ROAD NETWORK EDITOR? 97
HOW TO CHOOSE A DIFFERENT EDITING MODE? 98
HOW TO CREATE NETWORK OBJECTS? 98
Create Nodes 98
Create Links 98
Create Segments 99
Create Lane Connections 100
Create Toll Plazas 100
HOW TO CREATE SENSORS? 100
HOW TO CREATE TRAFFIC SIGNALS? 100
HOW TO EDIT PARAMETERS FOR AN OBJECT? 101
HOW TO SELECT FIGURES? 101
How to move figures? 101
How to measure the distance between two points? 101

APPENDIX A: SIMLAB CONSTANTS 102

CONTROLLER TYPES 102
CONTROL DEVICES 102
STATE FOR TRAFFIC SIGNALS 102
VARIABLE MESSAGE SIGNS 103
SURVEILLANCE SENSORS 105
VEHICLE TYPE 106
Vehicle Class 106
Vehicle Group 106
OUTPUT 106
PATH FLAGS 107

APPENDIX B: CALIBRATION 108

CALIBRATION MODULE 108
GENERAL ASSUMPTIONS 108
MITSIMLAB INPUT FILES ASSUMPTIONS 108
INPUT FILES 112

Overview

A MIcroscopic Traffic SIMulator (MITSIM) has been developed for modeling traffic flows in networks involving advanced traffic control and route guidance systems (Yang and Koutsopoulos, 1996). MITSIM represents networks at the lane level and simulates movements of individual vehicles using car-following, lane changing, and traffic signal response logic. Probabilistic route choice models are used to capture drivers' route choice decisions in the presence of real time traffic information provided by route guidance systems. MITSIM is designed as a testbed for evaluating traffic management systems with the following characteristics:

- it provides real-time sensor data that mimics the surveillance capacities of the traffic management systems in an ITS environment;

- it accepts traffic controls and routing information as input from traffic management systems and maintains the state of traffic signals and signs in the simulated network; and

- it calculates a set of measures of effectiveness (MOE) that represent the performance of the systems to be evaluated.

1. Introduction

This document instructs the user on running:

- MITSIMLab
- Road Network Editor

1.1 Installing MITSIMLab

Initially, a file called GUI_Libs.tar.gz needs to be uploaded at the sax ftp, under MITSIMLab. (ftp sax, login as anonymous, cd pub, cd MITSIMLab, get GUI_Libs.tar.gz).

Then we need to Copy, Unzip and Untar the file, by writing tar zxvf GUI_Libs.tar.gz. We go into the directory that will be generated (GUI_Libs) and run the script called Install (as root), for example by running ./Install . This installs all the necessary libraries.

Now, as users, we copy the executables (bin) in our account.

Note: When the libraries are installed, that happens at the directory /USR/LOCAL/LIB . Because this directory is not in the library path, we edit the file /ETC/LD.SO.CONF and we add the /USR/LOCAL/LIB path. After this, we do /SBIN/LDCONFIG in order to update the libraries.

Finally, we need to copy the PMV and PMV_HOSTS profile into the home directory of each one of the users.

The MITSIMLab directory contains the following sub-directories and files:

1. bin - the various MITSIMLab executables for Linux

2. PVM - PVM is a software that MITSIMLab requires to run. the software and a document explaing how to install it are in this directory.

3. Example - a set of example input files that may be used as templates when creating inputs for your own work.

4. MITSIMLabUserGuide.pdf - the MITSIMLab user guide.

If we use RedHat 7.x, and for an easier way to install PVM, we simply type up2date PVM and for more information we type man up2date.

1.2 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 and gives the option of being run with or without graphics also.

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

`xmitsim –m master.mitsim` (to run with graphics)
`mitsim –m master.mitsim` (to run without graphics)

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

- Run PVM:

  PVM facilitates communication between the packages MITSIM and TMS

- 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:

`xsmc –m master.smc` (to run with graphics)
`smc –m master.smc` (to run without graphics)

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
```

1.3 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.

When bus transit operations are being simulated in MITSIMLab, the following input files may be used to define various aspects of the bus transit system:

- ✓ `Route.dat`

  The **`route.dat`** file is used to define the physical components of the network (e.g. bus stops and links) used by the bus transit system.

- ✓ `Schedule.dat`

  The **`schedule.dat`** file is used to define the individual transit trips to which a bus may be assigned. The **`schedule.dat`** file assigns the trips to particular routes and specifies scheduled arrival times at the stops on the route.

- ✓ `Run.dat`

  The **`run.dat`** file is used to define the runs, or sequence of trips (defined in the **`schedule.dat`** file) to which a bus may be assigned.

- ✓ `Bus.dat`

  The **`bus.dat`** file assigns individual buses to runs defined in the **`run.dat`** file, along with the bus type and a departure time at which the bus enters the network to begin service of the trips in its run.

- ✓ `Passenger.dat`

  The **`passenger.dat`** file specifies the mean transit passenger arrival rates and the percentage of bus load alighting at each stop, for each route and by time of day.

- ✓ `Conditional.dat`

  The **`conditional.dat`** file contains the condition thresholds for conditional priority strategies.

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.

When bus transit operations are simulated in MITSIMLab using the bus transit input files described above, the user may choose to generate the following outputs:

- **transtraj.out**

The transit trajectory file, **`transtraj.out`**, file contains bus (vehicle) ID, route ID, trip ID, schedule deviation, prevailing time headway between successive buses, and passenger load at each trajectory time step (equal to the trajectory time step in the trajectory.out file) during the simulation.

- **`busstop.out`**

  The **`busstop.out`** file contains a record for each time that a bus arrives at a bus stop during a simulation. Each record includes the bus stop ID, the bus ID, the route ID, the schedule deviation for the arrival, the time headway between the current bus and the previous bus on the same route, dwell time, the number of passengers wishing to board, the number of passengers alighting and the number of passengers left behind.

2. MITSIMLab Input Files

2.1 Master 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`**)

MITSIMLab requires the following inputs :

- Surveillance and Control Device data including the logic controlling the Control Device
- Operational Environment under which the system will be evaluated

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`

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.

When bus transit operations are being simulated in MITSIMLab, the following input files may be used to define various aspects of the bus transit system:

- ✓ `Route.dat`

  The **route.dat** file is used to define the physical components of the network (e.g. bus stops and links) used by the bus transit system.

- ✓ `Schedule.dat`

  The **schedule.dat** file is used to define the individual transit trips to which a bus may be assigned. The **schedule.dat** file assigns the trips to particular routes and specifies scheduled arrival times at the stops on the route.

- ✓ `Run.dat`

  The **run.dat** file is used to define the runs, or sequence of trips (defined in the **schedule.dat** file) to which a bus may be assigned.

- ✓ `Bus.dat`

The **bus.dat** file assigns individual buses to runs defined in the **run.dat** file, along with the bus type and a departure time at which the bus enters the network to begin service of the trips in its run.

- ✓ `Passenger.dat`

  The **passenger.dat** file specifies the mean transit passenger arrival rates and the percentage of bus load alighting at each stop, for each route and by time of day.

- ✓ `Conditional.dat`

The **conditional.dat** file contains the condition thresholds for conditional priority strategies.

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** (See …….for details)

```
[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.

```
[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 discussed in detail in section 2.1.7.

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"
```

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 and output file, and another will have a 30-second stepsize 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
```

2.2 Parameter Files

2.2.1 The `paralib.dat` file

The **`paralib.dat`** file contains the values of all the parameters used within MITSIMLab. Sections from a sample **`paralib.dat`** 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 **`paralib.dat`** file contains parameters for the following:

- Constants used throughout the simulation
- Legend units
- Thresholds for viewing graphical objects
- Route choice parameters for unguided and guided vehicles
- Information update
- Update step size (reaction time)
- Vehicle loading parameters
- Bus Attributes
- Car following parameters
- Merging
- Lane-changing parameters
- Gap acceptance model for lane changing
- Parameters for yielding probabilities
- Parameters for nosing and yielding
- Parameter for discretionary lane change
- Start up delays
- Response to controls
- Parameters for headway variance
- Limiting parameters
- Distribution of driver groups
- Acceleration tables
- Limiting speed by vehicle type and grade
- Lane speed ratio

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
# meteric 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 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
```

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 next section contains information on the types of buses and bus characteristics, which are useful when simulating detailed bus operations for transit systems with different bus types. Since the physical characteristics of a bus can affect bus operational performance, MITSIMLab allows the user to specify a fleet of buses that may have varying characteristics that are specific to bus operations. Bus types and corresponding parameters may be defined in `paralib.dat` like so:

```
# BUS ATTRIBUTES

%   (1) Label (Type)
%   (2) Length (overrides lengths in Vehicle Classes) (ft)
%   (3) Seating capacity
%   (4) Total capacity
%   (5) Average boarding rate (sec/passenger)
%   (6) Average alighting rate (sec/passenger)
%   (7) Dead time lower bound at bus stop (sec)
%   (8) Dead time upper bound at bus stop (sec)
%   (9) Crowding factor (sec/passenger)

[Bus Classes] =
{
%   1                   2   3    4    5     6    7    8    9
    "Standard Bus"     40  32   70  2.6   1.7  2.0  5.0  0.5
    "Articulated Bus"  60  46  124  2.2   1.4  2.0  5.0  0.5
}
```

Bus operators may begin to make lane-changing maneuvers well in advance of bus stops in order to arrive in the appropriate lane to serve a stop. The “Bus LC Visibility” below

is the distance upstream of a bus stop at which a bus will begin to position itself in a lane to reach the bus stop. Bus operators know where their stops are without having to rely on "physical" visibility.

```
[Bus LC Visibility] = 1500  # feet
```

Often, when buses stop to serve passengers in the shared lane on a single-lane road, vehicles behind the bus stop "squeeze" around bus, using the available lane space next to the bus and occupying some of the space in the lane in the opposite direction. To represent this reduced capacity, wayside stops may be modeled, and, when the stop is occupied by a bus, the speed limit in the adjacent lane is multiplied by the "Bus Stop Squeeze Factor" to reflect the reduced capacity.

```
[Bus Stop Squeeze Factor] = 0.7  # 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:

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

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

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

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 aggresive merge from ramp
                # (var: aggresiveRampMergeProb)
}
```

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.

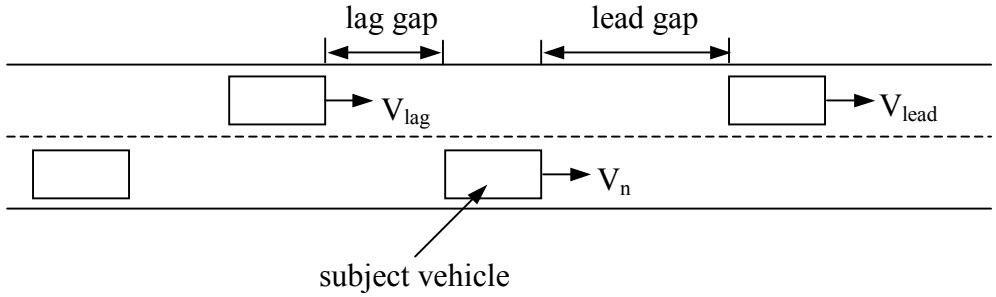
In MITSIMLab, drivers enter a hierarchical decision making process before making a lane change. The first decision that must be made is whether a vehicle is in a mandatory lane changing situation. A vehicle might be in a mandatory lane changing situation if its current lane is ending or if its current 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 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
}
```

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 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
}
```

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, lead_rel_spd) + b[3] * rem_dis_impact +
%           b[4] * gap + b[5] * nlanes
%
%   rem_dis_impact =  10 - 10 / (1 + exp(b[2] * dis))
%
% where:
% lead_rel_spd   = speed of subject vehicle subject to speed of lead
%                  vehicle in gap, i.e. Vn/Vlead
% 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* = (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*(Vsub-V*)+B4*dheavy+B5*dtail)}]
%
% where Vsub   = speed of subject vehicle
%       dheavy = 1, if subject is a heavy vehicle, 0, otherwise
%       dtail  = 1, if subject being tailgated, 0, otherwise
%
```

```
% Prob(choose left (or right) lane|driving conditions not
% satisfactory)=
%   1/[1+exp{-(B6+B7*(Vlead-V*)+B8*(Vfront-V*)+B9*(Vlag-Vsub))}]
%
% 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
    -1.0         # bus-following 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 Visiblity 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
}
```

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 VSLS (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
}
```

2.3 The `Network` 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 aids 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 connections
- Sensors
- Traffic controls
- Toll plazas
- Bus stops

2.3.1 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"}
```

2.3.2 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.)

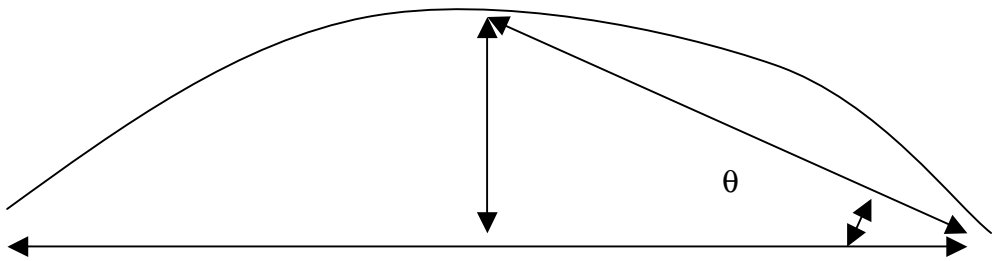


Figure 2.1: Illustrating Bulge for a Segment

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 : #Segments : #Lanes

```
[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}
   }
 }
.
.
.
```

2.3.3 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}        …
```

2.3.4 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}
}
```

2.3.5 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}.

Traffic control devices

Traffic control devices are defined according to the following format:

```
[Control Devices] :
{
# { Type |Visibility |Segment |Position
#   Code |Distance   |ID      |InSegment
#         { DeviceID | InitialState }
# TS, PS, VSLS, and VMS are link specific. It controls the
# traffic in all lanes of a link.
# LUS and Ramp Meters are lane specific.
#         { DeviceID | InitialState |LaneID }
# }
  {4 1000 5 0.602762
    {0 0x12200000}
  }
  {12 328 8 0.907322
    {1 0x3 22}
  }
  {12 328 3 0.875842
    {2 0x3 10}
  }
  {10 328 0 0.859956
    {3 0x3 0}
    {4 0x3 1}
    {5 0x3 2}
  }
  {10 328 1 0.587701
    {6 0x3 3}
    {7 0x3 4}
    {8 0x3 5}
    {9 0x3 6}
  }
  {10 328 0 0.290674
    {10 0x3 0}
    {11 0x3 1}
    {12 0x3 2}
  }
  {1 328 3 0.355527
    {13 0x3}
  }
}
```

A control device is associated with a device ID (**`DeviceID`**), type code (**`TypeCode`**), initial state (`InitialState`), visibility distance (`VisibilityDistance`), 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). Control devices can be specific to either a link or a lane.

The type code can be one of the following:

```
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)
```

The type code is the number corresponding to the device type. In addition, the sensor can either be specific to a lane, or can cover the entire width of the link. Traffic signals, portal signals, variable speed limit signs and variable messages signs are link specific. Lane use signs, ramp meters, and toll booths are lane specific.

Initial state can be one of the following:

```
0x0 Blank
0x1 Red
0x2 Yellow
0x3 Green
0x4 Arrow (protected)
0x8 Flashing (yield)
```

Toll Plazas

Toll plazas are defined according to the following format:

```
[Toll Plazas] :
{
# Each toll plaza consists of one or more toll booths. Toll
# booths are treated as lane specific control devices
# { Visibility |Segment |Position
#   Distance   |ID      |InSegment
#         { Booth |Initial |Lane |LaneUse |Speed |Flow
#           ID    |State   |ID   |Rules   |Limit |Rate }
# }
}
```

A toll plaza is associated with a booth ID (**`BoothID`**), visibility distance (`VisibilityDistance`), segment ID (**`SegmentID`**, the ID of the segment on which the sensor is located), a position in the segment (**`PosInSegment`**, expressed as a fraction from the downstream node), initial state (`InitialState,` the same initial states as control devices), lane ID (`LaneID`), speed limit (`SpeedLimit`), and flow rate (`FlowRate`).

2.3.6 The Bus Stops

The next section in the file describes parameters of bus stops in the network. The number following **`[Bus Stops]`** specifies the total number of bus stops defined. Below is a

sample portion of a `network.dat` file, demonstrating how bus stops might be declared for the network in Figure A-1:

```
[Bus Stops] : 7
{
  {328 20 0.13
    {10 0x3 31 0 100 6 0}
  }
  {328 31 0.3
    {11 0x3 44 128 75 6 0}
  }
  {328 42 0.18
    {15 0x3 67 0 75 6 0}
  }
  {328 45 0.6
    {16 0x3 75 0 75 6 0}
  }
  {328 69 0.55
    {17 0x3 128 0 75 6 0}
  }
  {328 45 0.6
    {18 0x3 75 0 75 6 0}
  }
  {328 69 0.55
    {19 0x3 128 0 75 6 0}
  }
}
```

Visibility Distance (ft) | Segment ID | Position in Segment

Bus stop ID | Initial State | Lane ID | Lane Use Rules

Stop Length (ft) | Squeeze Dummy | Wayside Dummy

The stop length is the distance upstream of the bus stop "stop line" within which a stopped bus (whether due to congestion or due to another bus already at the stop) may serve passengers and proceed. The placement of the bus stop in a wayside bay (i.e. a second lane) and the use of the squeeze dummy and the "Bus Stop Squeeze Factor", specified in the **`paralib.dat`** file, are meant to reflect conditions where the stop is located in the mixed traffic lane on a single-lane road. The squeeze dummy is set equal to 1 if the stop is in a shared lane on a single-lane road, and zero otherwise. When the squeeze dummy is equal to one, the speed of vehicles in the lane adjacent to the lane containing the bus stop is multiplied by the squeeze factor. The wayside dummy is equal to 1 if the stop is a wayside stop, and 0 otherwise. The specification of a stop as wayside triggers certain bus operator behavior when the bus is ready to depart from the stop to merge with the adjacent traffic stream.

2.4 Demand Files

2.4.1. 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.

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.

```
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  }
}
```

Modify this **Scaling Factor** to alter the demand level. For example, a scaling factor of 0.80 represents 80% of the demand that follows.

Vehicle type code for the type of vehicle to which the O-D flows pertain

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

(Origin,Destination,Demand (veh/hr))

.
.
.

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.

Below is an illustration of the data contained within the demand file.

```
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   }
}
```

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.

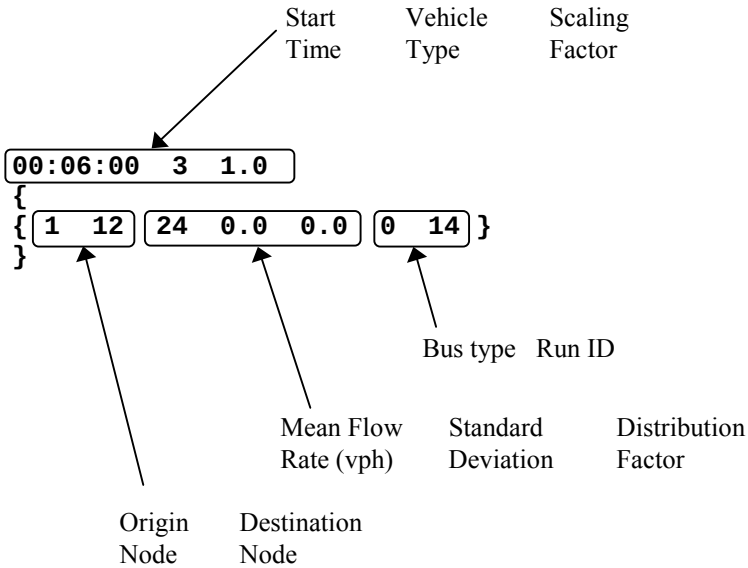
Additional parameters may be specified for each OD pair: the standard deviation of the departure rate (demand), and a distribution factor. In the example below, the standard deviation is 0.0 and the distribution factor is 1.0 for both OD pairs.

```
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.

Bus assignments may also be specified in MITSIMLab's **`demand.dat`** file, where the user may specify the origin (i.e. the upstream node of the first link on the first trip), the destination (i.e. the downstream node of the last link on the final trip), the flow rate (vehicles per hour), the variance of the time headway, the distribution, the bus type and the run ID. Below is an example portion of a **`demand.dat`** file that assigns buses to bus runs, which are defined in the **`run.dat`** file (see section 2.1.11), at an average headway of 2.5 minutes (i.e. 24 vph):



- According to the input specification above, the flow rate of 24 buses per hour begins at 6 A.M. Thus, at a mean time headway of of 2.5 minutes (24 vehicles per hour), buses of type 3 will be generated in the network.
-

2.4.2 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 }
}
```

number of paths, each row is a path

path travel time

% Converted from:

```
%   ptabsetpath35sens_1.out
% Using the program PathSorter by Qi Yang
```

2.4.3 Link travel times

2.1.8 The `linktime.dat` and `update.dat` Files

The `linktime.dat` is used to input historical link travel times which are used by the route choice model for unguided vehicles. The `update.dat` contains the travel times for guided vehicles but is not real-time.

`Linktime.dat` is also used as a template file. It specifies the number of periods and period duration to report the data within the `lft_i.out` and `linktime_mitsim.out` output files. An example of such 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 part of each of the above sentences after the symbol '%' is a comment.
The sixth line gives the order in which the values will be written into the output files. **`ID`** refers to the link ID, while **`type`** and **`length`** refer to the type of the link (1 – freeway, 2 – ramp, 3 – urban road) and length of the link respectively.

The 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
```

2.5 Traffic Control

2.5.1 Overview:

The TMS module in MITSIM can simulate a wide range of traffic control and advisory devices. Examples include:

- Intersection controls: traffic signals, yield and stop signs (TS).
- Ramp controls: ramp metering and speed limit signs.
- Mainline controls: lane use signs (LUS), variable speed limit signs (VSLS),variable message signs (VMS), portal signals at tunnel entrances (PS).

These signals and signs are controlled by four types of traffic signal controllers:

- Static Controllers
- Pre-timed Controllers
- Adaptive Controllers
- Metering Controllers
- Activated/Generic Controllers

As the simulation proceeds a controller can switch from one type to another based on the implemented logic. For example, a controller may switch to activated control in the off-peak period and to pre-timed control in the peak period.

Each controller is characterized by data items such as :

- Controller type
- Signal type(e.g., traffic signal at intersections (TS)
- Variable speed limit signs (VSLS)
- Lane use signs (LUS), etc.)
- Number of egresses (i.e. the out-degree of the intersection node)
- IDs of signals it controls
- Timing table, etc.

The representation of the control devices as network elements is described in Section 3.4.2. In the following sections, traffic signals at an intersection are used as an example to describe how traffic signal operations are represented in TMS. The same methodology is applied to other control elements (e.g., LUS, VSLS, VMS, etc.).

2.5.2 Static Controllers:

Static controllers represent the traffic signal and signs whose state does not change during the simulation. Examples include yield and stop signs, fixed speed limit signs and message signs. Each static controller contains a list of traffic signals or signs under its

control and uses a vector of signal states to establish the right-of-way for each turning movement. This state vector does not change during the course of simulation.

2.5.3 Pretimed Controllers

Pretimed controllers simulate the operation of signals and signs whose states change according to a pre-determined sequence. This type of controller is typically used to represent pretimed intersection signal controls. The control logic for a pretimed controller is specified by an offset and a timing table, which consists of a set of phases and control intervals .

Offset:
Offset is used to coordinate operations of the traffic signals between adjacent intersections in the network. It is defined as the time difference between the beginning of the first phase and a given master clock time (i.e. start time of the simulation).In order for pretimed signal controls to be coordinated, it is necessary to have the signals operate on a common cycle time (or in rare cases on the modulus of a common cycle time).

Timing Table:
A set of phases represents the timing table for a pretimed controller. Each phase is further divided into control intervals. For traffic signals at intersections, three intervals are typically used: green, yellow and red clearance. Each control interval represents a period of time during which states of all signals remain constant. The data items describing a control interval are length of the interval and a vector of signal states. The length species the time duration of the interval and the signal states specify the right-of-way for various turning movements.

The number of bits required to represent each signal state variable is equal to 4. Number of egress links 1 .Each 4 bits of this variable represents the right-of-way for a particular movement, and may have the following values:

0 = Blank: The signal is not used (no regulation for the movement);
1 = Red: Movement is not allowed or vehicles should stop before moving again (i.e. flashing red, see below);
2 = Yellow: Vehicles should prepare to stop if time headway from the signal is longer than a threshold;
3 = Green: The movement is allowed.
One of the following two tags can be added to a signal \color value" to indicate whether the movement is protected or has a lower priority.
4 = Arrow: The movement is protected;
8 = Flashing: The movement has a lower priority.

The signal states for up to 8 movements from a given approaching link is fully represented by a single 32-bit integer number. The implementation allows the right-of-way for each turning movement to be represented by a single hexadecimal digit and the sophisticated signal sequencing to be coded in a very compact and organized fashion. For

example, a flashing red signal (the equivalent of a stop sign) is assigned a state value of 9; a green arrow for protected turning movement is assigned a state value of 7.

2.5.4 The Adaptive Controller:

Traffic Adaptive controllers use real-time data from surveillance detectors and pre specified control laws. According to the particular system to be evaluated, the control logic may require coding as a customized module and interfacing with the TMS, or it may be implemented as a new controller type in TMS. Depending on the particular system to be evaluated, the control logic may require coding as a customized module and interfacing with the TMS, or it may be implemented as a new controller type in TMS. The modular design and object oriented implementation facilitate the addition of new types of controllers into the system with minimal effort.

The functionality of the system will be demonstrated through the implementation of the default traffic adaptive controller in TMS. The default controller is described by three data blocks of: (i) signal records; (ii) phase records; and, (iii) detector records.

Signal records: A signal record provides information on signal ID, maximum red time, and the ID of the phase to be called in the event that continuous red time for the signal has reached maximum value. As in pretimed controllers, a composite signal is used to control all movements from a particular approaching link.

Phase records: As with pretimed controllers, each traffic adaptive controller consists of a set of phases, and each phase is further divided into control intervals. The data items for the control intervals are the same as for pretimed controllers (i.e. each control interval has a length and a state vector for the signals defined in signal records). The difference is that, for the traffic adaptive controllers, a phase can be either extendible, called, or both. A phase is extendible if its green interval can be extended when detector data satisfies certain criteria and the continuous red time for any conflicting movement has not reached its maximum value. A phase is called if, after completion of an earlier phase, signal operations can be switched (shortcut) to this phase without completing the subsequent phases in the cycle. Based on the real-time traffic information obtained from the detectors and the pre-specified control logic, the green interval of a phase may be extended or terminated. The actual length of the green interval of an extendible phase and the sequence of phases may change as a result of the extension and calling logic. The logic for extending and calling a phase is specified by the detector records and the following variables:

Extension time: In a traffic adaptive signal controller, the length of the green interval is the minimum green time for a phase. An extension time (e.g. seconds) can be specified to extend the current phase incrementally. If the extension time is non-zero, detector measurements and continuous red time for each movement are examined at the green interval's end to decide whether the current phase should be extended. If so, the green interval continues for a number of seconds as specified by the extension time.

Maximum green time: When the length of the green interval reaches the maximum green time, it switches to the next interval without further extension,

regardless of traffic conditions measured by the available detectors.

Reset flags: They determine whether the time of the green interval of a phase should be reset to its initial value when the subject phase is completed.
When the last interval (i.e. red clearance) of a phase is completed, the conditions for calling an alternative phase are examined. If the conditions warrant switching to a called phase, the signal states are switched to the first interval of that phase; otherwise the signal operation moves to the next phase in the cycle.

Detector records: Detector records specify the logic for extending the current phase and calling a new phase. A controller may contain any number of detector records, each corresponding to a single detector. These records contain the following information: detector ID, a vector of speciation flags for extendible phases, and a vector of speciation flags for called phases. Each speciation flag indicates under what conditions the corresponding phase should be or should not be extended or called (see Table 4.2). In the current implementation, a speciation flag consists of three pieces of information: state indicator, selection indicator, and group indicator.

The State indicator describes the required state of the detector for the phase under consideration and may have one of the following values:

0000 = irrelevant to the phase;
1000 = idle state is required; and
2000 = activated state is required.

The selection and group indicators determine whether the detectors are considered inclusively or exclusively. The selection indicators may have one of the following values:

0100 = preemptive (if the detector has the required state, no other records in any group will be examined further);
0200 = any one in the group (the condition is satisfied if any detector in the group has the required state);
0300 = every one in the group (the condition is satisfied only if all the detectors in the group have the required state).

A group indicator, represented by the last two digits of hexadecimal speciation flag, may have a value between 1 and 255. Records in the same group are treated as a family and the conditions for whether to call (or extend) the phase or not are examined together.

Using the notation described above, a variety of adaptive signal control logic can be represented.

2.5.5 Metering Controllers

Ramp and mainline metering can be represented by either pretimed or traffic adaptive controllers if the timing table is predetermined or a direct mapping from detector state to signal state is available. Otherwise, a metering controller can be used to simulate the signal operations whose metering rate is adjusted based on measured or predicted downstream traffic conditions. The following parameters are used for metering controllers (their values are obtained from a control logic le and can be updated periodically):

Cycle length: This defines the time in seconds required by the metering signals to complete a green-yellow-red cycle. The time splits between green and red intervals are calculated based on the current metering rate and the given cycle length.
Limiting parameters: The metering rate must be chosen from a feasible range because of capacity constraints, driver behavior, safety considerations, etc. These constraints are represented by a set of limiting parameters, including minimum green time, yellow time, and maximum red time. These parameters are the basis for translating metering rates into an implementable timing table.
Regulators and desired state: These define the parameters used in Eq (4.4), including the regulator parameter _, the detectors that measure the network state x(t), and the default value of desirable state.

Queue detectors: To prevent queue caused by metering from spilling back to the upstream intersection and blocking cross traffic, a queue over-writing logic is often used in practice. The metering controllers in TMS can be assigned a list of queue detectors and an occupancy threshold to determine whether the queue length has become critical. When the measured occupancy exceeds the threshold, the metering rate is increased to the maximum rate in order to reduce the queue length.

Step size for updating metering rate: The metering rate and signal timing table may be updated periodically based on sensor measurements. A step size is used to specify this updating frequency. Between two updates the metering rate remains constant and ramp signals operate according to the timing table. When the metering rate changes, the corresponding timing table will not be implemented until the current phase is completed. This allows a control interval to complete its minimum length.

2.5.6 The Actuated/Generic Controller :

The Generic Controller has been designed to be able to simulate the widest possible range of signal controllers. It is applicable to any case, regardless of the specific control system in use. This is especially important because MITSIM is applied in multiple countries with different signal controller standards.

Clearly, it is impossible to capture the precise details of every possible controller type, but all controllers have a common base of core functions that can be identified. The approach of the generic controller is to break down control strategies into these basic logic elements and to implement these elements within a modular framework. Specific control logic can then be recreated from these basic components, and the modular structure allows any specialized features to be implemented easily.

*Control Logic:*Regardless of the complexity of the control system in use, the output of the control logic that is seen by a driver is extremely simple: a color indication on a signal face. Therefore, when a control logic is viewed on an individual signal group basis, the logic reduces to the single decision of whether to hold the current indication or to change to a different one. Input variables for this logic are also relatively limited. These include time-based inputs, data from sensors on the network, and data from other signals and controllers.
The most basic logic elements are time-based, either relative to a local timer or to a system clock time. A local timer is used if a signal has restrictions on the duration for which an indication is displayed, such as minimum and maximum lengths. For example, the logic may specify that a green signal will be held if it has been active for less than six seconds and will be forced to change to yellow if it has been active for more than 30 seconds. Alone, these logic units can simulate a pretimed controller by specifying the duration of each interval for every signal. Clock time is used as an input if actions must occur at specific times, such as when signals are running in coordination. In these cases, start and end times for signal intervals may be specified relative to a system clock.

More advanced logic elements use sensor readings from the network as inputs and perform different actions depending on those values. For example, a vehicle detected on a side-street approach may force the primary-street signal to change from green to yellow in preparation for serving the newly arrived vehicle. In another example, a signal group may skip its green interval if detectors on its approach are unoccupied.

States of other signals in the intersection can also be used as logic inputs, allowing the definition of complementary or conflicting signal groups or the specification of a phase plan. For example, one signal group may be held in green as long as another complementary group is still green. In another case, a signal group may be held in red if a conflicting group is active, preventing conflicting movements from being allowed simultaneously. Moreover, by defining conflicting movements and identifying which signal groups wait for which other groups, an implicit phase order can be defined.

Just as the examples described above can be combined to form a description of a control strategy, the basic logic elements can be combined to construct a full control logic. The generic controller models these basic elements and provides an interface to allow these components to be ordered and combined. Unlike the existing controller types, however, the logic for the generic controller is specified in terms of signal groups instead of signal phases. This distinction is very significant. Specifying logic in terms of signal phases divides the signal cycle into distinct time periods. In operation, the controller progresses in time, displaying phases in a specified sequence. By specifying logic in terms of signal

groups, the signal cycle is divided by groups of vehicle movements. In operation, these signal groups progress individually in time, changing states when specified. Specification in terms of signal groups allows much greater flexibility in defining the controller logic, as all the possible phases need not be enumerated.

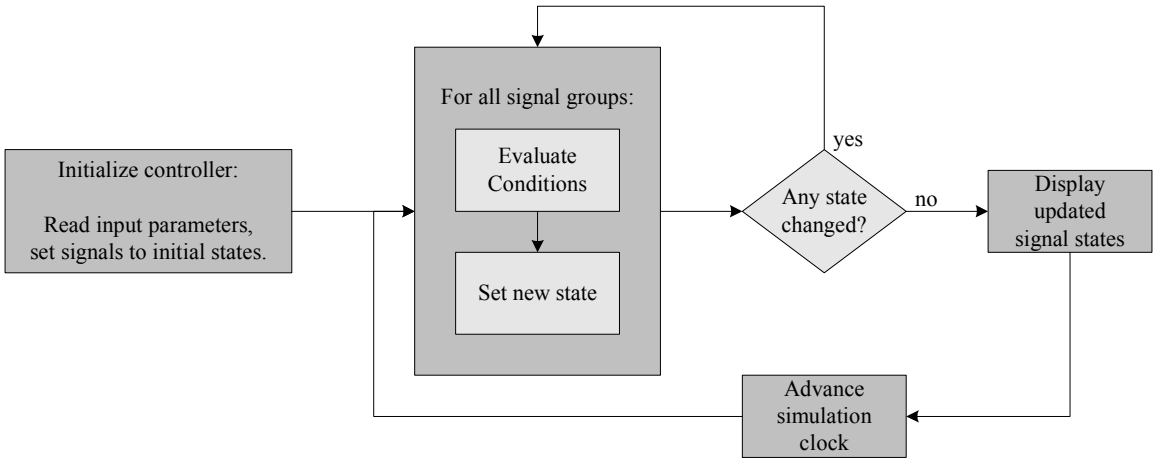


Figure -1: Overall logic of the generic controller.

The first step is the initialization of the controller, in which the parameters for the controller are read in from the signal input file. These parameters include general data—the IDs of the signals that the controller directs, the movements controlled by each signal group, and the initial state of each signal group—as well as the conditions which specify the control logic for each signal group. As part of this step, each group is also set to its specified initial state. Next the controller iterates through all the signal groups. For each signal group, it evaluates the logic conditions and determines whether the group's state should be updated. Because the group states may be interdependent, with the state of one group as an input to the logic of another group, this evaluation step must be iterative. Therefore, when the state of all the signal groups has been determined, the controller again cycles through the groups to check if the new states cause more signal groups to change. This process is repeated until the states of all signal groups are stable, at which point the controller displays the updated states of the traffic signals in TMS. This process is repeated each time step, typically 0.1 seconds of simulation time.
This iterative process is required because the controller evaluates the signal groups in the order in which they are defined, an order that may be arbitrary. In order for the evaluation order not to affect the control logic, this iteration step allows all signal groups to be updated to their final states before the clock time advances. With properly defined logic, the signal group states will stabilize after a number of iterations, the updated states will be displayed, and the clock time will be advanced.

The most critical part of this process is the evaluation of the conditions that may apply for each signal group. The conditions are evaluated in the order in which they appear in the signal input file. Unlike for the signal groups, the order in which the conditions are listed is significant, as this is an integral part of the definition of the control logic. There are four types of conditions, corresponding to different actions that each produces: general conditions perform miscellaneous functions, change conditions advance the signal group

to the next period, hold conditions keep the group in the current period, and skip conditions indicate that the next specified number of conditions should be skipped. Figure 1□3 shows the detailed steps for the evaluation of conditions for each signal group.

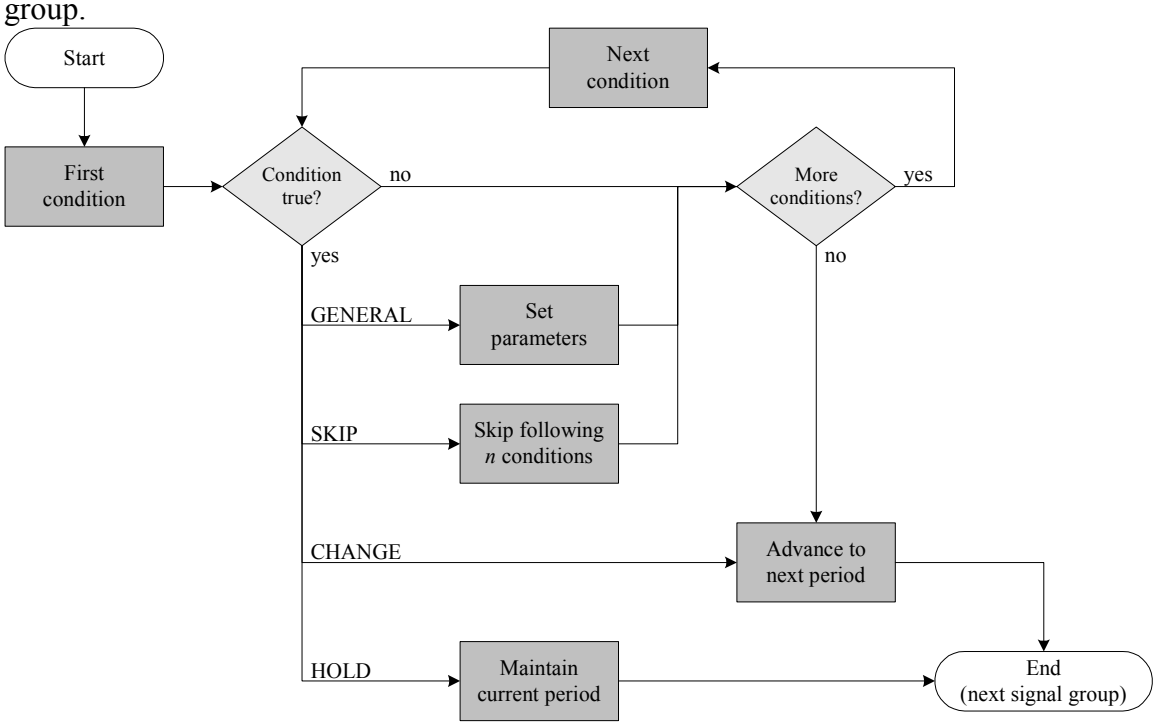


Figure 1□3: Condition evaluation logic within signal group of generic controller.

Logic Conditions: General conditions are typically listed and evaluated first, as they set various parameters and perform calculations that must occur every time step. For example, a gap timer which indicates the time since the last vehicle detection can be implemented as a general condition, as this timer must be either incremented or reset at each time step.

Change conditions specify the conditions that will force the signal group to change to a specified state. For example, a signal may be forced to change indication after a specified maximum duration or, if operating in coordination, at a specific clock time. If a change condition is met, the new state will be displayed and the remaining conditions will be skipped.

Hold conditions specify the conditions that will keep the signal group in its current state. For example, a signal may be prevented from changing if it has been active for less than a minimum duration, if specified detectors are occupied, or if conflicting movements are active. If a hold condition is met, the current state will be maintained and the remaining conditions will be skipped.

Skip conditions are used to create more advanced logic using combinations of conditions. If a skip condition is met, the next specified number of conditions will be skipped. This type of condition can be used to specify an alternate logic for different circumstances.

For example, a group may have a minimum time of 15 seconds under normal conditions but a minimum time of only eight seconds when demand is present on other approaches. A skip condition can therefore be used to ignore the longer minimum time when specified detectors are occupied.

For each condition, the following information is given:

1. *Code*: a unique three-digit code that identifies the condition in the signal input file. The first digit of the code indicates the condition type:
   0XX = general
   1XX = change
   2XX = hold
   3XX = skip
2. *Description*: a brief description of the logic of the condition.
3. *Number of Parameters*: the number of input parameters that the condition takes. "N" indicates a number that will vary based on the input specification.
4. *Parameters*: the input parameters required for specifying the condition.

Table A.1 and Table A.2 list the conditions used for basic signal control functions, as described in Section **Error! Reference source not found.**. Table A.3 lists the conditions used for transit signal priority functions, as described in Section **Error! Reference source not found.**.

Table A.1: Basic logic conditions (general and change conditions).

Code	Description	Number of Parameters	Parameter [0]	Parameter [1]	Parameter [2]	Par
000's	(General Conditions):					
001:	Next Period - sets [next period] to follow [current period]	2	current period	next period		
002:	Delayed Start (Offset) - group starts with signals blank - is initialized after a delay of [lag time]	1	lag time			
003:	Advance Start - group is started [lead time] seconds into the initial period	1	lead time			
004:	Gap Timer - resets gap timer if any of [detectors] is actuated	N	detectors[N]			
100's	(Change Conditions):					
101:	No Demand - [current period] changes to [next period] if none of [detectors] is occupied	N+2	current period	next period	detectors[N]	
102:	Force-Off - [current period] changes to [next period] if "time in cycle" equals [force-off time] where "time in cycle" = (time_since_initialization % [cycle length]) - [offset]	5	current period	next period	force-off time	cycl
103:	Maximum Time - [current period] changes to [next period] if active for more than [maximum time]	3	current period	next period	maximum time	
104:	Follow Other Group - [current period] changes to [next period] [delay] seconds after [sync group] changes to [sync period]	5	current period	next period	sync group	syn

Table A.2: Basic logic conditions (hold and skip conditions).

Code	Description	Number of Parameters	Parameter [0]	Parameter [1]	Parameter [2]	Par
200's	(Hold Conditions):					
201:	Minimum Time - [current period] held if active for less than [minimum time]	2	current period	minimum time		
202:	Extension Time - [current period] held if gap timer exceeds [extension time] - not held if period has been active for more than [maximum time]	3	current period	extension time	maximum time	
205:	Complementary Groups - [current period] held if [complementary groups] still active	N+1	current period	complementary groups[N]		
206:	No Conflicting Calls - [current period] held if none of [detectors] is actuated	N+1	current period	detectors[N]		
207:	Conflict Clearance - [current period] held if [conflicting groups] not completed - held until conflicts have been complete for [clearance time]	N+2	current period	clearance time	conflicting groups[N]	
208:	Hold Indefinitely - [current period] held	1	current period			
210:	Hold in window - [current period] held if "time in cycle" is between [start] and [end] where "time in cycle" = (time_since_initialization % [cycle length]) - [offset]	5	current period	start window	end window	cycl
300's	(Skip Conditions):					
301:	Detector Occupied - Skip next condition if any of [detectors] is occupied	N	detectors[N]			
302:	Detector Not Occupied -Skip next condition if none of [detectors] is occupied	N	detectors[N]			
303:	Skip next [num. to skip] conditions if [group]'s [flag] is [on/off]	4	num. to skip	group	flag	c

Table A.3: PRIBUSS logic conditions.

Code	Description	Number of Parameters	Parameter [0]	Parameter [1]	Parameter [2]	Parameter [3]
000's	(General Conditions):					
006:	Bus priority check-in - increments check-in counter if [detector] is actuated	1	detector			
007:	Bus priority check-out - decrements check-in counter if [detector] is actuated	1	detector			
008:	Bus priority: extension - set flag if bus checks-in and if "time in cycle" is between [start window] and [end window] - unset flag if check-in counter is zero or if "time in cycle" is greater than [end extension] where "time in cycle" = (time_since_initialization % [cycle length]) - [offset]	5	start window	end window	end extension	cycle length
009:	Bus priority: insertion - set flag if check-in counter is > 0 and if "time in cycle" is between [start window] and [end window] - unset flag if check-in counter is zero or if "time in cycle" is greater than [end insertion] where "time in cycle" = (time_since_initialization % [cycle length]) - [offset]	5	start window	end window	end insertion	cycle length
010:	Set bus insertion flags for preceding groups - only valid in window - resets flags at reset time	N+5	start window	end window	reset time	cycle length
011:	Set bus insertion flags for following groups - only valid in window - resets flags at reset time	N+5	start window	end window	reset time	cycle length
012:	Bus priority: early start - set flag if check-in counter is > 0 and if "time in cycle" is between [start window] and [end window] - unset flag if check-in counter is zero or if "time in cycle" is greater than [reset time] where "time in cycle" = (time_since_initialization % [cycle length]) - [offset]	5	start window	end window	reset time	cycle length
013:	Set shortened flags for preceding groups - only valid in window - resets flags at reset time	N+5	start window	end window	reset time	cycle length
014:	Reset check-in counter - resets counter to zero at specified time in cycle	3	reset time	cycle length	offset	

Example of Signal Input File: Phase-Based Specification

This section presents an example signal timing plan for an eight-phase dual-ring traffic signal controller and the corresponding MITSIMLab signal input file. The timing plan is for actuated-coordinated operation . Table B.1 shows the signal timing data for each phase.

Table B.1: Signal timing data for eight-phase controller.

	Phase:	1	2	3	4	5	6	7	8
	type	fixed	sync		fixed	fixed	sync		
minimum green	(sec)			5				5	10
extension time	(sec)			3				3	5
yellow time	(sec)	3	4	3	4	3	4	3	4
red clearance	(sec)	1	1	1	1	1	1	1	1
split	(%)	20	35	20	25	20	35	20	25
split time	(sec)	24	42	24	30	24	42	24	30
phase sequence:	ring 1	1	2	3	4				
	ring 2					2	1	3	4

cycle length = 120 sec

offset = 20 sec

Notes:

1) Sync phases and fixed phases are not subject to extension, so minimum green and extension time are not needed.

2) Splits are given as percentages of the cycle length; multiplying the split by the cycle length gives the split time.

Figure B-1 shows the phase diagram for the controller with all necessary timing data. The corresponding MITSIMLab signal input file follows.

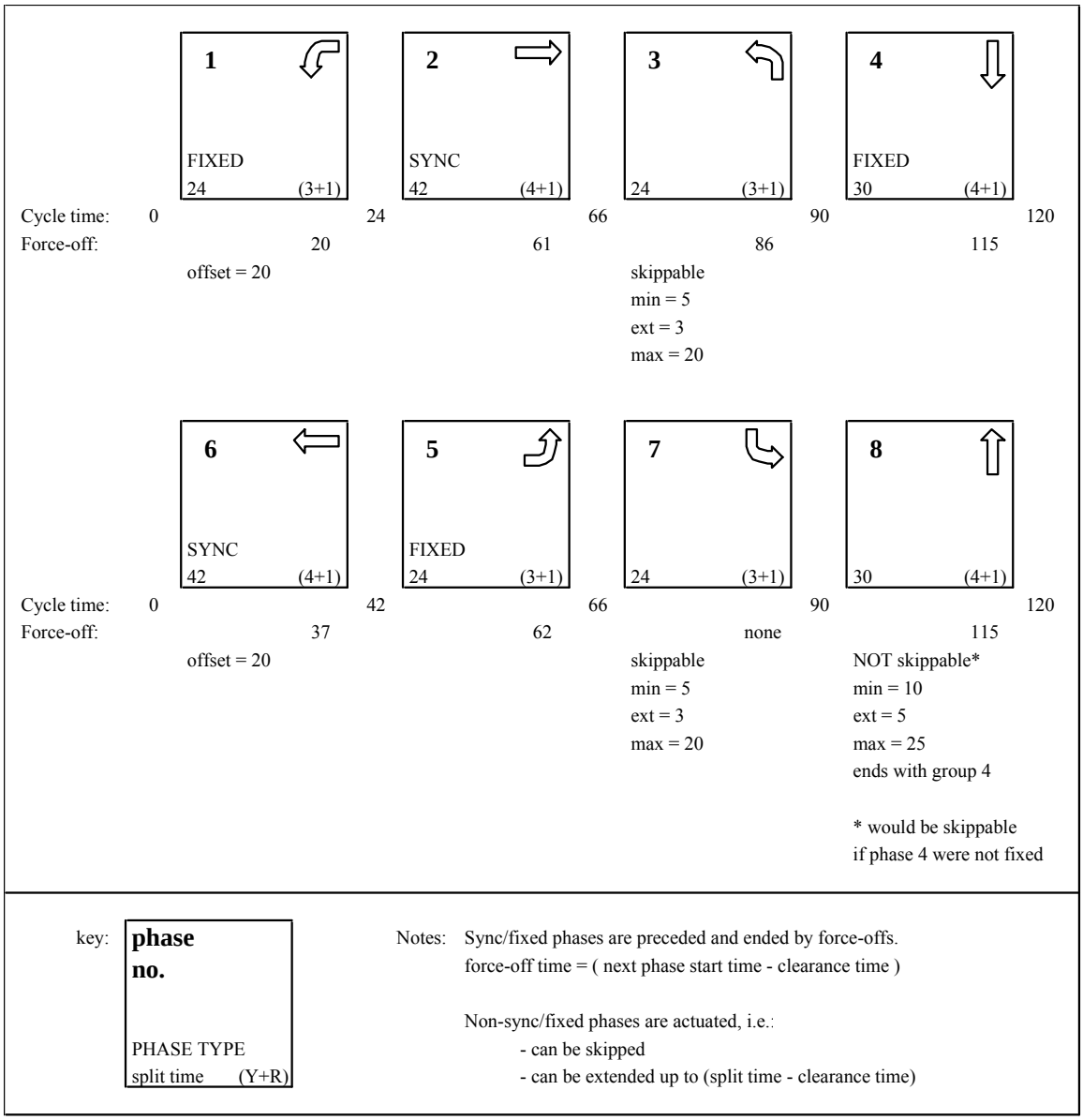


Figure B-1: Phase and timing diagram for eight-phase controller.

```
00:00:00        # Signal plan start time
{

  385           # ControllerID
  20 1 3        # ControllerType SignalType NumEgresses
  {
    4   # NumSignals
    {
      150 { 7 4 4 }     # SignalID { GroupID for each movement }
      151 { 5 2 2 }
      152 { 3 8 8 }
      153 { 1 6 6 }
    }

    8   # NumSignalGroups
    {
      1 3 5     # GroupID InitialPeriod NumConditions
      {
        # ConditionCode [NumParameters] { Parameters }
        002   { 20. }                   # delayed start (offset)
        102   { 3 2 20. 120. 20. }      # G: force-off
        208   { 3 }                     # G: hold
        201   { 2 3. }                  # Y: min
        207 3 { 1 1. 4 }                # R: conflict clearance
      }

      2 1 4
      {
        102   { 3 2 61. 120. 20. }      # G: force-off
        208   { 3 }                     # G: hold
        201   { 2 4. }                  # Y: min
        207 3 { 1 1. 1 }                # R: conflict clearance
      }

      3 1 7
      {
        004 1 { 260 }                   # gap timer
        101 3 { 3 1 260 }               # G: no demand
        102   { 3 2 86. 120. 20. }      # G: force-off
        201   { 3 5. }                  # G: min
        202   { 3 3. 20. }              # G: extension
        201   { 2 3. }                  # Y: min
        207 3 { 1 1. 2 }                # R: conflict clearance
      }

      4 1 4
      {
        102   { 3 2 115. 120. 20. }     # G: force-off
        208   { 3 }                     # G: hold
        201   { 2 4. }                  # Y: min
        207 3 { 1 1. 3 }                # R: conflict clearance
      }

      6 3 5
      {
        002   { 20. }                   # delayed start (offset)
        102   { 3 2 37. 120. 20. }      # G: force-off
        208   { 3 }                     # G: hold
        201   { 2 4. }                  # Y: min
        207 3 { 1 1. 8 }                # R: conflict clearance
      }

      5 1 4     # GroupID InitialPeriod NumConditions
      {
        102   { 3 2 62. 120. 20. }      # G: force-off
        208   { 3 }                     # G: hold
        201   { 2 3. }                  # Y: min
```

```
      207 3 { 1 1. 6 }                # R: conflict clearance
    }

    7 1 6
    {
      004 1 { 257 }                   # gap timer
      101 3 { 3 1 257 }               # G: no demand
      201   { 3 5. }                  # G: min
      202   { 3 3. 20. }              # G: extension
      201   { 2 3. }                  # Y: min
      207 3 { 1 1. 5 }                # R: conflict clearance
    }

    8 1 6
    {
      102   { 3 2 115. 120. 20. }     # G: force-off
      201   { 3 10. }                 # G: min
      202   { 3 5. 25. }              # G: extension
      205 2 { 3 4 }                   # G: complementary group
      201   { 2 4. }                  # Y: min
      207 3 { 1 1. 7 }                # R: conflict clearance
    }

  } # End of all Signal Groups

 } # End of Controller
} # End of Time Period
```

Signal Group Specification:
This section presents a signal timing plan from the Hornstull case study network (see Chapter 5 for description) and the corresponding MITSIMLab input file. Section C.1 presents the base timing plan, and Section C.2 presents the plan with PRIBUSS functions implemented.

C.1 Pretimed-Coordinated Control

The signal group diagram containing the basic timing data is shown in Figure C-1. The corresponding MITSIMLab signal input file follows.

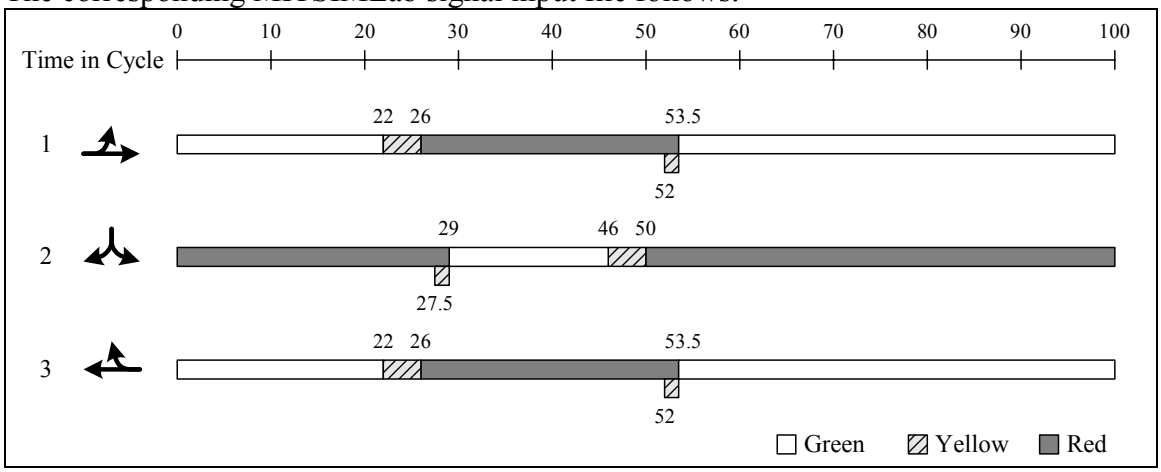


Figure C-1: Signal group diagram (Hornsgatan-Varvsgatan intersection).

```
06:30:00        # Signal plan start time
{

9               # ControllerID
  20 1 2        # ControllerType SignalType NumEgresses
  {
    3   # NumSignals
    {
      21 { 2 2 }        # SignalID { GroupID for each movement }
      22 { 1 1 }
      23 { 3 3 }
    }

    3   # NumSignalGroups
    {

      1 3 6     # GroupID InitialPeriod NumConditions
      {
        # ConditionCode [NumParameters] { Parameters }
        102   { 3 2 22. 100. 0. }       # G: force-off
        208   { 3 }                     # G: hold
        201   { 2 4. }                  # Y: min
        001   { 1 18 }                  # R: next period
        207 3 { 1 2. 2 }                # R: conflict clearance
        201   { 18 1.5 }                # R/Y: min
      }

      2 1 6     # GroupID InitialPeriod NumConditions
      {
        # ConditionCode [NumParameters] { Parameters }
        102   { 3 2 46. 100. 0. }       # G: force-off
        208   { 3 }                     # G: hold
        201   { 2 4. }                  # Y: min
        001   { 1 18 }                  # R: next period
        207 3 { 1 1.5 1 }               # R: conflict clearance
        201   { 18 1.5 }                # R/Y: min
      }

      3 3 6     # GroupID InitialPeriod NumConditions
      {
        # ConditionCode [NumParameters] { Parameters }
        102   { 3 2 22. 100. 0. }       # G: force-off
        208   { 3 }                     # G: hold
        201   { 2 4. }                  # Y: min
        001   { 1 18 }                  # R: next period
        207 3 { 1 2. 2 }                # R: conflict clearance
        201   { 18 1.5 }                # R/Y: min
      }

    } # End of signal groups

  } # End of controller

} # End of Time Period
```

C.2 Pretimed-Coordinated Control with Bus Priority

In this example, priority-equipped buses can call for green extension or phase shortening. The parameters for the priority logic are shown in Table C.1 (see Section **Error! Reference source not found.** for parameter descriptions). All values are times relative to time 0 in the cycle. The MITSIMLab signal input file incorporating the PRIBUSS functions follows.

Table C.1: PRIBUSS parameters (Hornsgatan-Varvsgatan intersection).

	Group	Call Window Start (A)	Call Window End (B)	Time-out (D)
Green Extension	1	99	22	35
	3	1	24	35
Phase Shortening	1	33	46	53.5
	3	33	46	53.5

```
06:30:00        # Signal plan start time
{

9              # ControllerID
 20 1 2        # ControllerType SignalType NumEgresses
 {
  3   # NumSignals
  {
    21 { 2 2 }        # SignalID { GroupID for each movement }
    22 { 1 1 }
    23 { 3 3 }
  }

  3   # NumSignalGroups
  {

    1 3 14    # GroupID InitialPeriod NumConditions
    {
      # ConditionCode [NumParameters] { Parameters }
      006   { 109 }                    # check-in
      007   { 110 }                    # check-out
      014   { 53.5 100. 0. }           # reset check-in counter
      008   { 99. 22. 35. 100. 0. }    # bus extension flag
      012   { 33. 46. 53.5 100. 0. }   # bus shortening flag
      013 6 { 33. 46. 53.5 100. 0. 2 } # bus shortening flag preceding
      210   { 3 40.5 22. 100. 0. }     # G: hold in window
      303   { 1 1 64 0 }               # skip if this group not extended
      208   { 3 }                      # G: hold indefinitely
      205 2 { 3 3 }                    # G: complementary group
      201   { 2 4. }                   # Y: min
      001   { 1 18 }                   # R: next period
      207 3 { 1 2. 2 }                 # R: conflict clearance
      201   { 18 1.5 }                 # R/Y: min
    }

    2 1 8     # GroupID InitialPeriod NumConditions
    {
      # ConditionCode [NumParameters] { Parameters }
      102   { 3 2 46. 100. 0. }        # G: force-off
      303   { 1 2 2048 1 }             # skip if this group shortened
      208   { 3 }                      # G: hold
      201   { 3 4. }                   # G: min
      201   { 2 4. }                   # Y: min
      001   { 1 18 }                   # R: next period
      207 3 { 1 1.5 1 }                # R: conflict clearance
```

```
      201   { 18 1.5 }                 # R/Y: min
    }

    3 3 14    # GroupID InitialPeriod NumConditions
    {
      # ConditionCode [NumParameters] { Parameters }
      006   { 125 }                    # check-in
      007   { 126 }                    # check-out
      014   { 53.5 100. 0. }           # reset check-in counter
      008   { 1. 24. 35. 100. 0. }     # bus extension flag
      012   { 33. 46. 53.5 100. 0. }   # bus shortening flag
      013 6 { 33. 46. 53.5 100. 0. 2 } # bus shortening flag - preceding
      210   { 3 40.5 22. 100. 0. }     # G: hold in window
      303   { 1 3 64 0 }               # skip if this group not extended
      208   { 3 }                      # G: hold indefinitely
      205 2 { 3 1 }                    # G: complementary group
      201   { 2 4. }                   # Y: min
      001   { 1 18 }                   # R: next period
      207 3 { 1 2. 2 }                 # R: conflict clearance
      201   { 18 1.5 }                 # R/Y: min
    }

  } # End of signal groups

 } # End of controller

} # End of Time Period
```

Example of a General Signal Input Files:
A. Sample program segment :

```
/*
 * SIGNAL TIMING

 * NOTE: Tables for signal plans MUST be sorted by time.  Each table
 * can contains multiple controllers.  Each controller can be one of
 * the following types:

 *   1 = TS
 *   2 = PS
 *   3 = VSLS
 *   4 = VMS

 *  10 = LUS
 *  12 = Ramp Meter
 *  16 = Toll Booth

 * and control logic can be one of the followings:

 *   0 = Unsignalized (flash operation, for example)
 *   1 = Pretimed
 *   2 = Activated
 *   3 = Metering
*  20=Generic Control

 * Controllers in the tables can be put in arbitrary order.
 *

 * StateIndicators in the Sequence Table, represented by the state IDs
 * (i.e. color) and operational mask of each signal or signs.  For
 * traffic signals such as TS, PS, LUS, Ramp Meter and Toll Booth, the
 * StateIndicators specifies the right-of-way for turning movements
 * from a particular approach during a given time interval.  The color
 * IDs of defined as the followings

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

 * Two operational masks can be attached to these colors:

 *   4 = Arrow mask
```

```
*   8 = Flashing mask

* State Indicators is specified in hex format.  For example, red arrow
* is 5 (1+4), flash yellow is A (2+8), and green arrow is 7 (3+4).
* However, the hex number F (3+4+8) does not means a Green flashing
* arrow, instead, it means that the corresponding turning movement is
* not allowed at all.

* SignalIndicators for the outgoing links of each link is represented
* by one digit.  These numbers are listed without space with the
* right most number corresponds the right most outgoing link, and the
* left most number corresponds the left most outgoing link (possibly
* a U turn).  The maximum number of outgoing links is 8.

* DetectorIndicators in the Detector Table specify the logic for
* extending the current phase or calling a new phase.  The value for
* a DetectorIndicator can be any of the followings:

*   0 = Do not care
*   1 = Must be idle
*   2 = Must be activated
*   3 = Preemptive if activated

* The example listed below has 3 control periods and 1 controller.
* During the 1st period (6:00-6:30am), flash operation is used
* (e.g. night operation).  During the 2nd period (6:30-7:30), a
* pretimed plan is used (e.g. peak period operation).  After 7:30am,
* a full-activated plan is used (e.g. mid-of-day operation).
*/

07:15:00        # Becomes in effect at 7:15AM
{
  1 0 1 4 {         # ID ControllerType SignalType NumEgresses
    4 { 30 32 31 33 }        # SignalIDs
    1 {       # Signal phase
      { 1
        0 { AAA AAA 999 999 }        #  Signal interval/Flash operations
      }
    }
  }              # End of a controller
}

07:20:00        # Becomes in effect at 7:30AM
{
  1 1 1 4 {         # ID ControllerType SignalType NumEgresses
    4 {             # {SignalIDs}
```

```
      30 # (F,L,N)
      32 # (M,G,E)
      31 # (B,H,J,K)
      33 # (A,I,C,D)
     }
   0 4 {       # Offset NumPhases
     {        3       # NumIntervals
       9 { 111 113 333 111 }
       3 { 111 112 222 111 }
       1 { 111 111 111 111 }
     }
     {        3
      15 { 111 111 111 333 }
       3 { 111 111 111 222 }
       1 { 111 111 111 111 }
     }
     {        3
      17 { 111 333 111 113 }
       3 { 111 733 111 112 }
       1 { 111 333 111 111 }
     }
     {        3
      18 { 333 333 111 111 }
       3 { 222 222 111 111 }
       1 { 111 111 111 111 }
     }
   }
 }             # End of a controller
}

08:30:00       # Becomes in effect at 8:30am
{
 1 2 1 4 {          # ID ControllerType SignalType NumEgresses
   4 {              # {SignalID MaxRed RedCalledPhase}
     { 30 120 4 } # (F,L,N)
     { 32 120 3 } # (M,G,E)
     { 31 120 1 } # (B,H,J,K)
     { 33 120 2 } # (A,I,C,D)
   }
   4 {       # NumPhases
     {  2 9 0       3 # Extension MaxGreen Reset NumIntervals
       5 { 111 113 333 111 }
       3 { 111 112 222 111 }
       1 { 111 111 111 111 }
     }
     {  2 15 0 3
```

```
      5 { 111 111 111 333 }
      3 { 111 111 111 222 }
      1 { 111 111 111 111 }
    }
    {  2 17 0 3
      5 { 111 333 111 113 }
      3 { 111 733 111 112 }
      1 { 111 333 111 111 }
    }
    {  2 18 1 3
      8 { 333 333 111 111 }
      3 { 222 222 111 111 }
      1 { 111 111 111 111 }
    }
  }
  1 {       # Called phases
   2
  }
  2 {       # Extended phases

   2  3
  }
  14 {       # Detector Specifications
   # {ID {CalledPhaseSpecs} {ExtendesPhaseSpecs}}
   { 11  { 2 }  { 2 2 } }  # 1
   { 13  { 2 }  { 2 2 } }  # 1
   { 15  { 2 }  { 2 2 } }  # 1
   { 17  { 2 }  { 2 2 } }  # 1
   { 10  { 2 }  { 2 1 } }  # 2
   { 12  { 2 }  { 2 1 } }  # 3
   { 14  { 2 }  { 2 1 } }  # 4
   { 16  { 2 }  { 2 1 } }  # 4

  { 20  { 1 }  { 1 1 } }  # 5
   { 21  { 1 }  { 1 1 } }  # 5
   { 22  { 1 }  { 1 1 } }  # 5

   { 30  { 1 }  { 1 1 } }  # 6

   { 40  { 1 }  { 1 1 } }  # 7
   { 41  { 1 }  { 1 1 } }  # 7
  }
 }            # End of a controller
}
```

2.6 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}
}
```

2.7 Transit Inputs

2.7.1 The Transit Network Representation File (`route.dat`)

The transit network representation file, **`route.dat`**, specifies those portions of the general network that are used by particular transit routes. Below is a sample **`route.dat`** file:

```
[Bus Route Table] : 2
{
{ 86
      { 223 224 225 226 227 228 229 230 231 232 233 }
      { 10  11  12  13  14  15 }
}

{ 93
      { 471 472 473 474 475 476 477 478 479 480 481 }
```

Number of routes defined in the file

```
      { 15  16  17  18  19  10 }
}
}
```

Sequence of link IDs

Sequence of bus stop IDs

Route ID

The example shown above defines two bus routes in MITSIMLab, one "inbound" and one "outbound" on the same real world route. The user may also include a design headway in the definition of a route, as shown below.

```
[Bus Route Table] : 4
{
{ 86   300.0
      { 223 224 225 226 227 228 229 230 231 232 233 }
      { 10  11  12  13  14  15 }
}

{ 93   300.0
      { 471 472 473 474 475 476 477 478 479 480 481 }
      { 15  16  17  18  19  10 }
}
}
```

Design headway (sec)

Specifying a design headway is useful for defining frequent services (e.g. bus rapid transit), which do not have scheduled arrival times at stops. The design headway may be used when evaluating APTS that must determine the deviation from a desired headway in order to devise a real-time strategy for achieving a predefined level of service.

2.7.2 The Schedule Definition File (`schedule.dat`)

ROUTE 9 RIO GRANDE												
	Southbound						Northbound					
	10	11	12	13	14	15	15	16	17	18	19	10
AM							6:15	6:27	6:42	6:47	6:57	7:15
							6:45	6:57	7:12	7:17	7:27	7:45
	6:00	6:35	6:45	6:50	7:03	7:15	7:15	7:27	7:42	7:47	7:57	8:15
	6:50	7:05	7:15	7:20	7:33	7:45	7:45	7:57	8:12	8:17	8:27	8:45
	7:20	7:35	7:45	7:50	8:03	8:15	8:15	8:27	8:42	8:47	8:57	9:15
	7:50	8:05	8:15	8:20	8:33	8:45	8:45	8:57	9:12	9:17	9:27	9:45
	8:20	8:35	8:45	8:50	9:03	9:15	9:15	9:27	9:42	9:47	9:57	10:15
	8:50	9:05	9:15	9:20	9:33	9:45	9:45	9:57	10:12	10:17	10:27	10:45
	9:20	9:35	9:45	9:50	10:03	10:15	10:15	10:27	10:42	10:47	10:57	11:15
	10:20	10:35	10:45	10:50	11:03	11:15	11:15	11:27	11:42	11:47	11:57	12:15
	11:20	11:35	11:45	11:50	12:03	12:15	12:15	12:27	12:42	12:47	12:57	1:15

Table 2.1: A hypothetical bus schedule timetable.

The schedule definition file is used to define work trips to which buses may be assigned. A trip is a single, one-way, end-to-end movement along a single route. The schedule definition file allows the user to define trips with scheduled arrival times at stops along a route. Table 2.1 is a hypothetical real-world schedule timetable.

The shaded portion of the AM schedule in Table 2.1 is defined in the MITSIMLab input format below:

Number of trips defined in the file

Route ID

Sequence of arrival times

Trip IDs

```
[Bus Schedule Table] : 9
{
{ 86
  186  { 00:06:00  00:06:35  00:06:45  00:06:50  00:07:03  00:07:15 }
  286  { 00:06:50  00:07:05  00:07:15  00:07:20  00:07:33  00:07:45 }
  386  { 00:07:20  00:07:35  00:07:45  00:07:50  00:08:03  00:08:15 }
  486  { 00:07:50  00:08:05  00:08:15  00:08:20  00:08:33  00:08:45 }
  586  { 00:08:20  00:08:35  00:08:45  00:08:50  00:09:03  00:09:15 }
  686  { 00:08:50  00:09:05  00:09:15  00:09:20  00:09:33  00:09:45 }
  786  { 00:09:20  00:09:35  00:09:45  00:09:50  00:10:03  00:10:15 }
  886  { 00:10:20  00:10:35  00:10:45  00:10:50  00:11:03  00:11:15 }
  986  { 00:11:20  00:11:35  00:11:45  00:11:50  00:12:03  00:12:15 }
       { 10  11  12  13  14  15 }
}
```

Sequence of bus stop IDs

The schedule input can be specified in a number of ways to define various types of services. The sequence of bus stop IDs is optional. If no bus stop IDs are specified, then the buses that serve those trip IDs will stop at the bus stops that are specified in the route input in the transit network representation file (**`route.dat`**). The user may also give a reduced set of bus stop IDs in order to define express services that only serve certain stops. For example, if, from 8:20 AM onward (i.e. trips 586-986), the southbound trips are assigned to an express service that includes only stops 10, 11, 14, and 15, the input may be specified like so:

```
[Bus Schedule Table] : 9
{
{ 86
  186  { 00:06:00  00:06:35  00:06:45  00:06:50  00:07:03  00:07:15 }
```

```
  286  { 00:06:50  00:07:05  00:07:15  00:07:20  00:07:33  00:07:45 }
  386  { 00:07:20  00:07:35  00:07:45  00:07:50  00:08:03  00:08:15 }
  486  { 00:07:50  00:08:05  00:08:15  00:08:20  00:08:33  00:08:45 }
}
{ 86
  586  { 00:08:20  00:08:35  00:09:03  00:09:15 }
  686  { 00:08:50  00:09:05  00:09:33  00:09:45 }
  786  { 00:09:20  00:09:35  00:10:03  00:10:15 }
  886  { 00:10:20  00:10:35  00:11:03  00:11:15 }
  986  { 00:11:20  00:11:35  00:12:03  00:12:15 }
         { 10  11  14  15 }
}
```

Trips 186 to 486 serve all routes on the stop, so no bus stop IDs were specified. The schedule definition input file may also look very different if frequent services are defined. In order to specify trips that have no scheduled arrival times, the user need only specify a trip ID and a route ID. For example, if a route were a bus rapid transit route with no scheduled arrival times, the schedule input file may be defined as shown below:

```
[Bus Schedule Table] : 2
{
{ 86 186 }
{ 93 193 }
}
```

Since there are no scheduled arrival times, which distinguish one trip from another, only a trip ID and a route ID are required. Thus all southbound buses on the route may be assigned to trip ID 186, and all northbound buses to trip ID 193.

2.7.3 The Bus Run Definition File (`run.dat`)

A run in MITSIMLab is a sequence of trips, defined in the schedule definition file described in section 2.1.10, to which a bus may be assigned. Figure 2.2 highlights a sequence of alternating southbound and northbound trips from Table 2.1.

ROUTE 9	RIO GRANDE											
	Southbound						Northbound					
	10	11	12	13	14	15	15	16	17	18	19	10
							6:15	6:27	6:42	6:47	6:57	7:15
							6:45	6:57	7:12	7:17	7:27	7:45
	6:00	**6:35**	**6:45**	**6:50**	**7:03**	**7:15**	**7:15**	**7:27**	**7:42**	**7:47**	**7:57**	**8:15**
AM	6:50	7:05	7:15	7:20	7:33	7:45	7:45	7:57	8:12	8:17	8:27	8:45
	7:20	7:35	7:45	7:50	8:03	8:15	8:15	8:27	8:42	8:47	8:57	9:15
	7:50	8:05	8:15	8:20	8:33	8:45	8:45	8:57	9:12	9:17	9:27	9:45
	8:20	**8:35**	**8:45**	**8:50**	**9:03**	**9:15**	**9:15**	**9:27**	**9:42**	**9:47**	**9:57**	**10:15**
	8:50	9:05	9:15	9:20	9:33	9:45	9:45	9:57	10:12	10:17	10:27	10:45
	9:20	9:35	9:45	9:50	10:03	10:15	10:15	10:27	10:42	10:47	10:57	11:15
	10:20	**10:35**	**10:45**	**10:50**	**11:03**	**11:15**	**11:15**	**11:27**	**11:42**	**11:47**	**11:57**	**12:15**
	11:20	11:35	11:45	11:50	12:03	12:15	12:15	12:27	12:42	12:47	12:57	1:15

Trip 186
Trip 393
Trip 586
Trip 793
Trip 886
Trip 1093

Figure 2.2: Defining a MITSIMLab bus run from a real-world schedule

The run example in Figure 2.2 is a relatively short run assignment for a bus, with only three round trips on the route and a time span of merely 4 hours and 15 minutes. Nonetheless, the run is used here to demonstrate run definition in MITSIMLab. The sample input file below defines two bus runs from the schedule in Table A-1 on the route in Figure A-1.

```
[Bus Schedule Table] : 2
{
{ 13  { 186  393  586  793  886  1093 }
{ 14  { 286  493  686  893  986  1193 }
}
```

Sequence of trip (schedule) IDs

Run ID

2.7.4 The Bus Assignment File (`bus.dat`)

The bus assignment file is used to assign buses to bus runs and to specify when those buses enter the network. The bus assignment file specifies a start time, the ID of the bus, the run ID to which the bus is assigned, and the bus type. The bus assignment file also gives the user the option to specify the passenger load on the bus when the bus first enters the simulation. An example of a bus assignment file is given below:

```
00:06:00
{
{ 100  0  13  10
}

00:06:50
{
{ 200  0  14 }
}
```

Time of entry into network

Load at start (optional)

Run ID

Bus type

Bus ID

Both buses are of type 0. A bus' type has a number of characteristics associated with it, which are specified in MITSIMLab's parameter file, **`paralib.dat`**, described in section 2.1.4.

2.7.5 The Transit Demand File (`passenger.dat`)

A single file is used to specify the time-varying, stop-based and route-based distribution of passengers in MITSIMLab. The demand input file, **`passenger.dat`**, is used to define the arrival and alighting rates of passengers at each stop in the network by route and time of day. A sample **`passenger.dat`** file is given below:

```
00:06:00
{
  10  { 86  25.0  0.20 }
      { 93  50.0  0.20 }
  11  { 86  40.0  0.15 }
      { 93  30.0  0.15 }
  12  { 86  20.0  0.10 }
      { 93  10.0  0.10 }
  13  { 86  35.0  0.05 }
      { 93  50.0  0.10 }
  14  { 86  25.0  0.05 }
      { 93  25.0  0.05 }
  15  { 86  15.0  0.25 }
      { 93  45.0  0.10 }
  16  { 86  40.0  0.05 }
      { 93  20.0  0.15 }
  17  { 86  30.0  0.10 }
      { 93  65.0  0.10 }
  18  { 86  10.0  0.15 }
      { 93  10.0  0.15 }
  19  { 86  20.0  0.20 }
      { 93  35.0  0.20 }
}

00:07:30
{
  10  { 86  80.0  0.15 }
      { 93  50.0  0.15 }
  11  { 86  60.0  0.20 }
      { 93  90.0  0.05 }
  12  { 86  85.0  0.10 }
      { 93  55.0  0.15 }
  13  { 86  65.0  0.20 }
      { 93  40.0  0.15 }
  14  { 86  45.0  0.05 }
      { 93  60.0  0.05 }
  15  { 86  70.0  0.10 }
      { 93  35.0  0.15 }
  16  { 86  75.0  0.25 }
      { 93  25.0  0.20 }
  17  { 86  15.0  0.15 }
      { 93  45.0  0.20 }
  18  { 86  55.0  0.20 }
      { 93  60.0  0.10 }
  19  { 86  55.0  0.10 }
```

Start time

Two bus routes (86, 93) serve stop 10

Bus stop ID

Route ID | Arrival Rate (pass/hr) | Alighting Fraction

```
      { 93  50.0  0.15 }
}
```

Within a given time period, and for a given bus stop and route, the user may specify a mean passenger arrival rate for passengers wishing to board at the stop and the fraction of the passenger load on the bus that wishes to alight at the stop.

When passenger arrival and alighting data is not available, the user may specify dwell time upper and lower bounds for each stop. When the dwell time lower and upper bounds are defined, the dwell time is generated from a random, uniform distribution between the lower and upper bounds. An example is shown below:

```
00:07:30
{
  10  { 15.0  45.0 }
  11  { 20.0  50.0 }
  12  { 25.0  60.0 }
}
```

Bus stop ID

Lower Bound (sec) Upper Bound (sec)

2.7.6 The Priority File (`priority.dat`)

The signal priority input file allows the user to define the thresholds that are the basis for determining whether priority shall be granted to an approaching bus and/or the routes to which unconditional priority is given. The conditions specified in the signal priority file, **`priority.dat`**, can be designated to a specific route so that only buses serving a particular route may be considered for priority. The conditions may also vary by time of day, so that either the value of the thresholds or the very availability of priority may change with time. MITSIMLab is able to support five types of conditional signal priority: signal priority that depends on

- load,
- headway,
- schedule adherence,
- load and headway,
- and load and schedule adherence.

The combinations of conditions listed above are checked when a bus is detected on the approach to a signal where priority is implemented. The combination of conditions is identified in the priority input file by a condition code, which are defined as follows:

Table 2-2: Condition codes and corresponding thresholds

Condition Code	Threshold(s)
0	Unconditional priority
1	Minimum load

2	Maximum schedule deviation
3	Maximum headway
4	Minimum load & maximum schedule deviation
5	Minimum load & maximum headway

Below is a sample conditional.dat file that specifies each of the types of conditions shown in Table D-1.

```
06:00:00{
   0  { 20 } { 86  93  44  59 }
}

08:00:00
{
   1  { -30.0 } { 86  93 }
   2  { 330.0 } { 44  59 }
}

10:00:00
{
   3  { 20 -30.0 } { 86  93 }
   4  { 20 330.0 } { 44  59 }
}

<END>
```

Start time

Array of bus routes to which the condition applies

Threshold (minimum bus load)

Condition code (0 = load only)

The first condition, which is valid from 6 AM until 8 AM, applies to routes 86, 93, 44 and 59 and requires only that an approaching bus have at least 20 passengers on board in order to be granted priority. The first condition under 8 AM says that a bus on either route 86 or 93 must be behind schedule by 30 seconds or more in order to qualify for priority. The second condition under 8 AM says that the headway between the subject bus and the one that preceded it must be greater than 330 seconds in order to satisfy the priority condition. After 10 AM, both conditions must be met in order to be granted priority. The headway conditions generally apply to frequent services where a desired headway is specified, but may also apply to routes with scheduled arrival times. The schedule deviation condition, however, should be used only with services with scheduled arrival times. Unconditional priority is defined by leaving the condition brackets empty.

3 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`**
- Transit Trajectory File : **`transtraj.out`**
- Bus Stop Arrival File : **`busstop.out`**

The details of these files are discussed below.

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

3.1 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}
  }
.
.
.
```

3.2 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 0 5.4

1 3 112.363 0 2.6 0 2.6 0 2.6 0 2.6 0 2.6 0 2.6 0 2.6 0 2.6 0 2.6 0 2.6
0 2.6 0 2.6 0 2.6 0 2.6 0 2.6 0 2.6 0 2.6 0 2.6
.
.
.
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
.
.
.
```

Note: Some of the segments in the file above have zero flows and free flow travel times reported for the entire simulation period. This is because the corresponding links did not form a part of any path connecting an OD pair with non-zero flow.

3.3 The Link Flow and Travel Time Data Files

The link flow and travel time data files contain link-specific flow volumes and travel times data. The file **`lft_i.out`** is created by MITSIMLab.

The file is organized as shown below. The file header contains the start time of the simulation, the number of periods within the simulation window, and the time length (in seconds) of each period. The travel times and flows reported are the averages in each period. In the example below, there are 18 periods of 240 seconds duration, thus totaling a simulation window of 18*240 = 4320 minutes i.e. 72 minutes (remember that the simulation period is from 7:00 to 8:12). The template file that determines the frequency of collecting the information of interest is specified in the **`master.mitsim`** file (see section 2.2.5).

```
% LINK TRAVEL TIMES
% Generated on Mon Aug 28 17:28:17 2000

0     % Start period (07:00:00)
18    % Number of periods
240   % Seconds per period
```

The next section lists the link ID, link type and link length (in feet). The number of entries beyond the link length equals twice the number of time periods in the simulation. This corresponds to alternating flow and travel time entries.

```
0 3 350 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8
0 8 0 8

1 1 426.7 0 4.5 0 4.5 0 4.5 0 4.5 0 4.5 0 4.5 0 4.5 0 4.5 0 4.5 0 4.5 0
4.5 0 4.5 0 4.5 0 4.5 0 4.5 0 4.5 0 4.5 0 4.5


2 2 196 0 4.5 0 4.5 0 4.5 0 4.5 0 4.5 0 4.5 0 4.5 0 4.5 0 4.5 0 4.5 0
4.5 0 4.5 0 4.5 0 4.5 0 4.5 0 4.5 0 4.5 0 4.5
.
.
.
5 2 131 51 3.7 84 3.6 78 3.7 100 3.7 102 3.6 112 3.6 98 3.7 91 3.6 76
3.6 86 3.7 77 3.6 74 3.6 62 3.6 71 3.7 61 3.6 62 3.6 47 3.7 53 3.6
```

```
6 9 6041.3 51 111.5 83 112 78 114.4 100 112.6 102 112.4 110 112.7 99
111.9 92 115.9 75 109.9 86 111.3 79 108.3 73 107.6 62 108 70 109.1 62
107.9 62 108 48 110.4 53 109.3

7 1 1523.7 7 35.7 12 35.1 8 35.7 9 33.6 4 36.7 8 35.5 17 34.1 11 35.7
15 34.5 15 36.7 15 36.9 11 37.2 18 36.9 14 33.9 15 36.5 12 34.1 17 36.4
15 36.2
.
.
.
```

3.4 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 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 . . .
.
.
.
```

3.5 The Transit Trajectory File

The transit trajectory file is used to determine vehicle-based performance data. At every trajectory time step during the simulation (default = 1 sec), a record is written to the **`transtraj.out`** file for each bus for which the user has defined a bus assignment. The transit trajectory file contains the following pieces of information (in order by column number in the output file):

Column	Data item
1	Time (seconds)
2	Vehicle ID
3	Route ID
4	Trip ID
5	Distance traveled from the origin (meters)
6	Passenger load
7	Schedule deviation at the last stop (seconds)
8	Headway between the bus and the preceding bus at the last stop (seconds)

The transit trajectory file is only created if the **`trajectory.out`** file is also selected in the **`master.mitsim`** file.

3.6 The Bus Stop Arrival File

The bus stop arrival file, **`busstop.out`**, stores a record for each time that a bus, for which an assignment has been defined, arrives at a stop. The bus stop records may be used to determine bus stop-level and route-level measures of performance, such as headway variability. The user may also observe the relationships between passenger boarding and alighting activity and dwell times. The information stored in each record in the **`busstop.out`** file includes:

Column	Data item
1	Time (seconds)
2	Bus stop ID
3	Route ID
4	Bus (vehicle) ID
5	Scheduled arrival time
6	Time headway between this bus and the last on the same route (seconds)
7	Dwell time (seconds)
8	Number of passengers wishing to board
9	Number of passengers alighting
10	Number of passengers left behind

4. The Road Network Editor (RNE)

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	
^h or ?	display this message
^o	display the options menu
^p	print canvas (not implemented)
^s	save all databases
1	append nearest existing point to figure
2	append new point to figure
3	alter last figure point
4	select previous figure
5	select current figure
6	select next figure
<Enter>	complete current figure or place selected point
N	mark current figure (link) incomplete
<Del>	delete current figure
.	delete last point from figure or delete selected point
<LMB>	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 imcomplete 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 32 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 nearese 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 inherits 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 <Del>.

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: 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
0x0000FFF0 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 B: Calibration

Calibration Module

This appendix gives details of each component of the calibration module and the instruction for using the module.
The entire module has been written in MATLAB and the following assumptions are made:

General Assumptions

- Counts data is available at the sensor locations.
- The weight matrix is diagonal.
- The OD time interval and the sensor interval are equal.

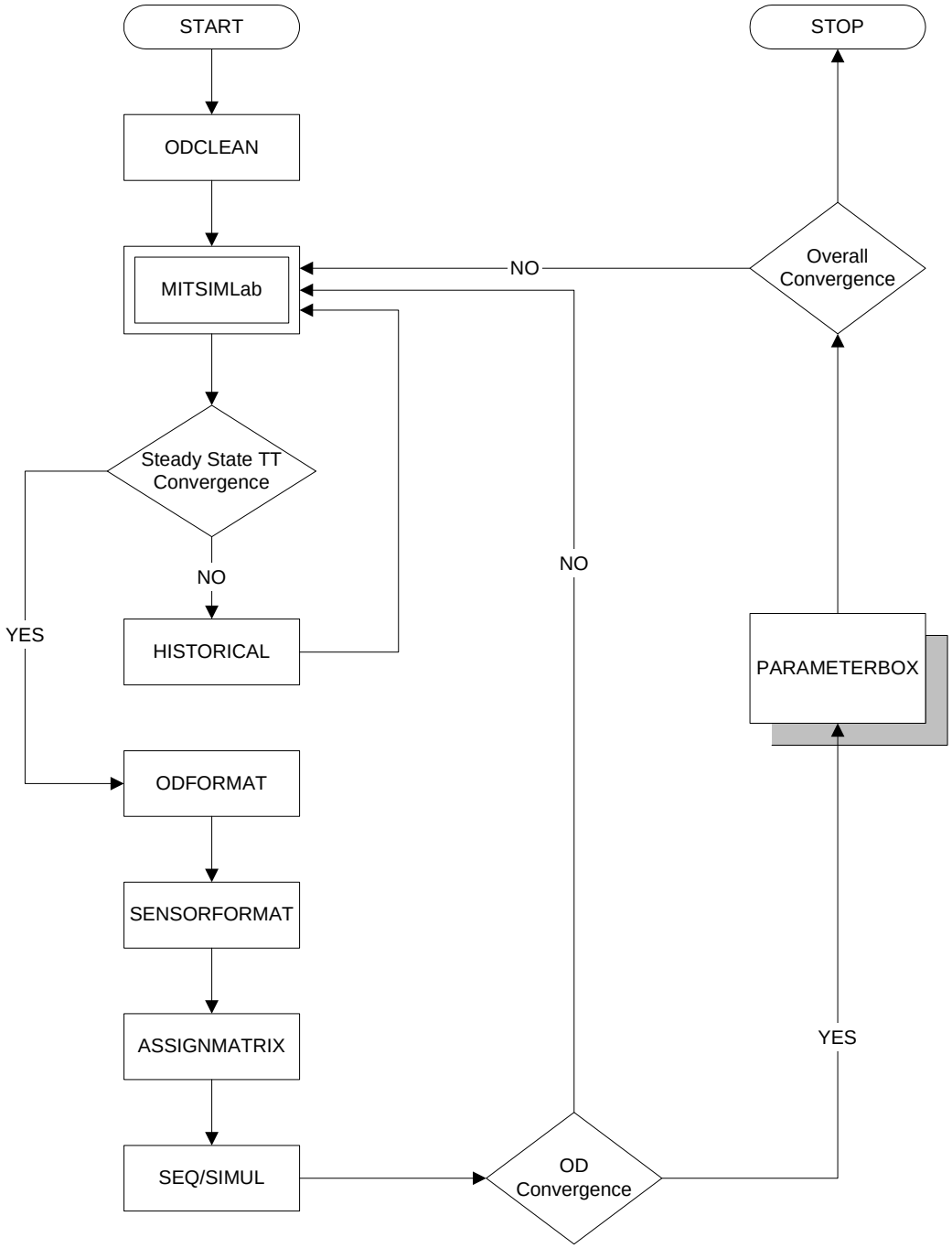
MITSIMLab Input Files Assumptions

- All input and output files are in the same directory.
- The OD file is called '**od.dat**'.
- The sensor output file is called '**sensor.out**'.
- The assignment matrix file is called '**assignment_matrix.out**'.
- The input historical travel time file is called '**linktime.dat**'.
- The output link travel time file is called '**linktime.out**'.
- The parameter file is called '**paralib.dat**'.
- The OD interval start time (in **od.dat**) is specified in seconds.

The module needs the following input files:

- Observed field counts file (**counts.dat**).
- Weights file (**weights.dat**)
- Seed OD file (**seed-od.dat**)
- Any other file required for calculating the objective function value.

The main program of the module is depicted in Fig. A.1:



A.1 Main program of the module

The module has many sub-modules and has checks for convergence at various stages. A description of each of the sub- modules and the convergence criteria used, is given below:

ODCLEAN: This module cleans the seed OD (**seed-od.dat**) file by removing comments from the file and writing the pertinent data to '**od.dat'**. This makes processing easier.

ODFORMAT: This module covert the OD file (**od.dat**) into a matrix format which makes it easier to manipulate.

```
27000 0 1
{
{ 1  4  2409 }
{ 1  7  453 }
{ 1  10  411 }
}
```

Format of OD

```
27000  1  4  2409
27000  1  7  453
27000  1  10  411
```

Format after ODFORMAT module

SENSORFORMAT: Similar to ODFORMAT module, this module coverts the sensor output file (**sensor.out**) into matrix format.

```
27900 {
      0 1 372
      1 1 440
      2 1 250
      3 1 42
}
```

Format of sensor.out

```
27900  0  372
27900  1  440
27900  2  250
27900  3  42
```

Format after SENSORFORMAT

ASSIGNMATRIX: This module generates the Assignment Matrix, which maps OD flows to sensor counts. Assignment Matrix represents the fraction of demand for a particular OD contributing to sensor counts by time period. It uses the '**assignment_matrix.out**' generated by MITSIMLab and the output files generated by ODFORMAT and SENSORFORMAT modules to generate the Assignment Matrix.

SEQ: This module uses the Assignment Matrix generated by ASSIGNMATRIX module, output file of ODFORMAT module, field sensor count file (**counts.dat**) and the weights file (**weights.dat**) to estimate the OD matrices. Constrained OD estimation is performed sequentially and the

estimated OD matrices are written back in MITSIMLab compatible format to '**od.dat'**.

SIMUL: This module is similar to the SEQ module. The only difference is that it estimates the OD matrices simultaneously.

HISTORICAL: This module calculates the weighted average of link travel time generated by MITSIMLab (**linktime.out**) and input travel time (**linktime.dat**) to produce a new historical travel time file (overwrite **linktime.dat**).

PARAMETERBOX: This module applies the Box algorithm and returns optimized value of the parameters. This module calls the BOX module, which in turn calls the CHECK module, INITIALPARAMETERS module, CHANGEPARAMETERS module and FUNC module. The description of these modules is given below:

BOX: In this module, the Box algorithm is implemented. It generates the initial (K) sets of chosen parameters. It calls the INITIALPARAMETERS module to get the initial value of the chosen parameters (specified in the parameter file). It then evaluates the function value associated with each set of parameters. To implement this, it first writes these values in the parameter file using the CHANGEPARAMETER module and then calls the FUNC module to evaluate the function value associated with them. Then based on the Box algorithm, it generates new set of parameters and repeats the above-mentioned steps till the convergence criterion is met.

INITIALPARAMETERS: This module passes the initial set of value from the parameter file to BOX module. This module needs to be modified based on the chosen parameters. In this module the user has to specify a method to extract the initial set of parameters (chosen for calibration), from the parameter file (**paralib.dat**).

CHANGEPARAMETERS: This module writes the new set of parameters into the parameter file (**paralib.dat**). This module will also change depending upon the parameters chosen for calibration. The user has to specify a method in this module to write the values of chosen parameters into the parameter file.

FUNC: This module returns the function values. It calls the simulator and the outputs of the simulator are then processed in this module to give the value of the objective function.

Convergence Criterion

There are four points where the convergence has to be verified:

- OD convergence

- Steady state travel time convergence
- Parameter convergence within Box algorithm
- Parameter convergence in the main iteration loop

The termination criterion used for checking the convergence of the OD matrix estimation module is based upon the maximum percent difference between the elements of the new estimated OD (x^k) with the old OD (x^{k-1}).

$$\max\left|\frac{(x_{ij}^k - x_{ij}^{k-1})}{x_{ij}^k}\right| < \varepsilon$$

$where:$

$\varepsilon : Tolerance$

The iterations end when this tolerance criterion is met or when maximum number of iterations has been performed.

The termination criterion for checking the converges of steady state travel time is exactly the same as in the case of OD-estimation, where the OD flows are replaced by the link travel times (tolerance ϕ).

Within the Box algorithm, the convergence is achieved if the difference between the maximum value of the objective function and the minimum value of the objective function is within β percent of the minimum function value, for γ (which is specified by user) consecutive iterations or when maximum number of iterations has been performed.

The convergence of parameters in the main iterative loop is again on the same lines as that of the OD convergence. The convergence criterion is based on the elements of the new estimated parameter value (α^k) with the old value of the parameter (α^{k-1}):

$$\max\left|\frac{(\alpha_i^k - \alpha_i^{k-1})}{R_i}\right| < \mu$$

$where:$

R_i : Difference between Upper and lower bound of the parameter

μ : Tolerance

The iterations end when this tolerance level is achieved, or when maximum number of iterations has been performed.

Input files

This section describes the input file required for the calibration program. The main files required are:

- counts.dat
- weights.dat
- INITIAL.m

counts.dat: In this file, the field counts are specified. There has to be one to one correspondence between sensor counts reported by MITSIMLab and order of sensors in this file. For e.g. if following is the order in '**sensor.out'** file:

```
27900 {
      0 1 300
      10 1 650
      5 1 2050
      3 1 200
```

```
}

      28800 {
            0 1 310
            10 1 750
            5 1 2210
            3 1 330
}

Then the format of counts.dat is:
```

```
27900  0  372
27900  10  600
27900  5  1980
27900  3  210
28800  0  350
28800  10  780
28800  5  2150
28800  3  240
```

Format of couts.dat

If more than one day data is used for calibration purpose then the data of day two is written below data for day 1 and so on.

weights.dat: In this file, the weights of the sensors at various locations (during different time intervals) are specified, followed by the weights of the OD pairs. The weights are specified as row vector in the file. The weights for sensors (for all intervals) are specified in the same order as the order of the sensors in the "**sensor.out**" file. Also, if more then one day is used then the weights for day two are written below day one and so on. In the end, the weights of the OD pairs (for all intervals) are specified. For e.g., Let there are 60 sensors reporting counts data, 500 OD pairs to be estimated, 10 intervals and 3 days of data available for calibration. Then the weights.dat file will have 600 (60 X 10) entries for counts for day 1, followed by 600 entries for days 2 counts, followed by 600 for day 3 and in the end 5000 (500 X 10) entries for OD pairs. So in total there would be 6800 weights.

INITIAL.m: This file is an input in the Box algorithm. The following parameters are specified in this file:

- Number of parameters to be calibrated (N).
- Initial number of sets to be generated (K).
- Maximum number of Iterations for Box algorithm (ITMAX).
- Alpha (reflection factor used in Box algorithm for getting new point).
- Beta (β) (Refer to Convergence criterion section).
- GAMMA (γ) (Refer to Convergence criterion section).
- DELTA (Explicit constraint violation correction).
- Vector of Lower Bound of parameter values (G).
- Vector of Upper Bound of parameter values (H).

Below is the format of the file:

```
N = 5;                          % No. of parameters to calibrate
K = 8;                          % No. of initial points (min N+1)
ITMAX = 10;                     % No. of iterations
ALPHA = 1.3;
BETA = 0.01;
GAMMA = 3;
DELTA = 0.0001;
G = [ -0.5; -0.5; 0; 0; -0.1];  % Lower bound of parameters
H = [ 0; 0; 1; 0.1; 0];         % upper bound of parameters
```

There are three modules that will change depending upon the parameters to be calibrated and the choice of objective function. They description is given below:

INITIALPARAMETERS.m : This function returns the initial values of the parameters to the BOX module. So, depending upon the choice of parameters this module will change and user will have to specify a method to extract the initial values. The default function calibrates 5 parameters (3 Route-Choice Parameters and 2 Car-Following parameters)

Route-Choice parameters:

- Travel time coefficient Beta (2: Unguided and Guided)
- Freeway Bias

These parameters are in the **paralib.dat** file and the highlighted number shows the parameters that are chosen for calibration.

```
[Route Choice] = {
%     Fraction     Beta
        0.7        -5.0
        0.3        -5.0
}

[Freeway Bias]     = 1.0      # Travel time factor
```

Car-Following Parameters: The scale parameters of Kazi CF model are calibrated. These parameters are also specified in the paralib.dat file.

```
[CF Kazi Parameters] = {
#    alpha   beta gamma rho  lambda stddev
 0.0400 0.722 0.242 0.682 0.600 #0.825 # acc 0.0225 (original alpha)
-0.0418 0.000 0.151 0.804 0.682 #0.802 # dec
}
```

CHANGEPARAMETERS.m: This function takes the new set of chosen parameters and writes them in appropriate file. Default function writes back the new set of above-specified parameter values in **paralib.dat**.

FUNC.m: This function returns the value of the objective function to the BOX module. Depending upon the available data and the choice of functional form, this function will change. Default objective function is:

$$\sum_{j=1}^{n_days} \sum_{i=1}^{l} (y_i - y_i^j)^2$$

where:

y_i : MITSIMLab count at sensor i.

y_i^j : Field count at sensor i on day j.

l : Number of sensors

n_days: Number of days used for calibration

To evaluate this function a **SensorId.dat** file is required. This file specifies the sensors ID that are useful (reporting counts, speed or occupancy), in the same order as reported in the **sensor.out** file.

A function COUNTSPEED is called by this function and it reads the **sensor.out** file and separates it into four file:

- **counts.out**
- **speeds.out**
- **occupancy.out**
- **headway.out**

These file report the count, speed, occupancy and headway respectively.

Instruction for Running the Module

- Copy all the calibration module file in a directory (e.g. ~/utilities/calibration)
- Start matlab from the directory where the input files are present.
- Add in path, the calibration directory:

  path(path,'~/utilities/calibration')
- Start the calibration program:

  CALIBRATION

The user will now be prompted to input the tolerance and the maximum number of iterations. The following are the inputs desired from the user:

- Command to execute (e.g. smc – m master.mitsim).
- Number of days of data used for calibration.
- Maximum number of iterations for OD-estimation.
- Tolerance (ε) for OD-estimation (Refer to convergence section).
- Maximum number of iterations for Steady State Travel Times.
- Tolerance (ϕ) for Steady State Travel Times (Refer to convergence section).
- Maximum number of iterations for parameter calibration.
- Tolerance (μ) for parameter calibration (Refer to convergence section).
- Simultaneous OD-estimation (1) or Sequential OD-estimation (0) (Enter 0 or 1 depending on the problem size).

The output is logged in a file called **OutputLog.out**. It reports the function values at each iteration and the corresponding parameter set. It also reports the tolerances after each iteration.