A Physically-Realistic Simulation Of Vehicle Traffic Flow

Thomas L. Thorpe

January 11, 1998

Technical Report CS-97-104

Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873

Phone: (970) 491-5792 Fax: (970) 491-2466
WWW: http://www.cs.colostate.edu

A Physically-Realistic Simulation Of Vehicle Traffic Flow

Thomas L. Thorpe

January 11, 1998

1. INTRODUCTION

The elements of a vehicle traffic flow simulator are presented as a basis to model and improve traffic light control strategies. Traffic control simulators have been described (Haight 1963, Paal 1975, and Reitman 1971) for hardware using assembler and FORTRAN. This simulator is written in C, uses one-second, discrete time steps for the traffic light controller agent and the simulation of vehicle movement. Vehicle movement is simulated with realistic velocity and acceleration limits and is constrained by minimum following distances. The speed limits on each lane are user specified and vehicles do not exceed these limits. The traffic environment occurs in a user defined grid of east-west and north-south streets. The simulator is fairly realistic: vehicles are stopped at red lights, vehicles maintain safe following distances, and the physics of motion of the traveling vehicles are closely approximated.

Cars are inserted in the network with their starting and destination intersections chosen randomly before the simulation is started and more vehicles can be added during the simulation. The routes are chosen using an Iterative Deepening A* algorithm. The route for each vehicle may require more than one turn because the under estimate used in route selection is based on the travel time through a lane of traffic. By using the same random seed for all testing, vehicles follow the same routes. This ensures the testing results between the reference and experimental implementations are directly comparable. Different traffic light control strategies may be implemented and tested.

The simulator internal processing is described next followed by a description of the simulator inputs and which modules within the program must be changed to implement new traffic control strategies.

2. SIMULATION FLOW

The simulation occurs in three phases: *Environment Definition*; *Traffic Simulation*; and *Simulation Summary Report*. The *Environment Definition* phase initializes the simulation by defining the number of intersections, lanes that connect the intersections, lane speed limits and fixed light timing control at each intersection if fixed light timing is to be tested. The *Traffic Simulation* phase adds vehicles to the simulation, determines vehicle routing, manages the traffic light status as dictated by the control strategy being tested and moves vehicles through the simulation to their destination. The *Simulation Summary Report* gives an indication of the overall performance of the simulation including: world simulation time; vehicle travel time; vehicle wait time at a stop light; and number of vehicle stops.

2.1 ENVIRONMENT DEFINITION

Before the actual simulation can begin, the traffic environment is defined by loading: intersection data; associated traffic light control information; and lane to intersection connections. Other simulation parameters specifying the control strategy, random seed, number of vehicles in the simulation, and subsequent vehicle injection rate are also loaded from a file or command line parameter.

After the simulation parameters are loaded, the intersection data is loaded. Each intersection becomes an element of a linked list to facilitate the simulation. The data initially associated with each intersection header is shown below.

1. North-south and east-west street names and ordinal reference numbers.

2. Anticipated heavy traffic flow direction.
3. Minimum and maximum traffic light signal times for left, through and right turn signals.
4. Settings and timings for fixed timing light control.

The setting and timings for fixed light timing control consists of a linked list of light control settings indicating the color of the left, through and right turn signals for each of the directions of traffic and duration until the lights should be switched to the next phase. The pointers to inbound and outbound traffic lanes are NULL until lane connections are added.

Lane queues interconnecting intersections are loaded next as linked lists. The lane data consists of:

1. From and to intersection ordinal reference numbers.
2. Lane speed limit.
3. Compass direction of travel.
4. The turn directions allowed from the lane.
5. Lane length in feet.
6. The presence of a vehicle sensor in the lane.

The lane from and to intersections and direction of travel are used to find the intersections to interconnect and determine which inbound and outbound lane pointer to set for the intersections. The lane queue will contain a linked list of vehicles traveling through the lane.

2.2 TRAFFIC SIMULATION

Once the traffic environment has been loaded, the simulation begins by alternately adding vehicles to the simulation and managing the linked list of intersections. A large number of vehicles are added initially and a smaller number of vehicles are added for a period of time during the simulation. Each intersection in the linked list is processed before adding more vehicles to the simulation. After the linked list has been traversed, one second in world time has elapsed. When an intersection is visited, the traffic lights are switched, if necessary, and the vehicles within each inbound lane of the intersection are moved depending on the light color for the lane, the vehicle's route and relationship with other vehicles.

Vehicles are added to the simulation by randomly selecting start and destination intersections. The route is determined by using an IDA* search. The travel estimates through each lane are based on the average travel time through the lane. The initial estimate for a lane is the lane length divided by the lane's speed limit. As the simulation progresses, the estimate will change based on actual travel time through the lane. When the vehicles route has been chosen, it is placed at the end of the lane queue between the starting and second intersection.

When an intersection is initially processed, the selected traffic light control algorithm determines if the lights should change or remain the same. For fixed light timing the time since the last light change is compared with the duration specified for the current phase to determine when to switch lights.

Other traffic light strategies can be implemented to study their effectiveness and compared to fixed light timing. Strategies that could be tested include simple rule based strategies such as giving the right of way to the lane with the greatest number of vehicles and control using Reinforcement Learning (Thorpe 1997). A simple base line to judge all other strategies, would be to use fixed light timing with all light colors set green and not worry about avoiding oncoming traffic when turning or passing through an intersection.

The movement of vehicles is based on the physics of motion under constant acceleration (see Figure 1) constrained by limits of acceleration of an average automobile. When a lane is processed, starting with the first vehicle in the lane, each vehicle is moved based on the following algorithm:

1. *Determine leading reference position for current vehicle.* The leading reference position is used to determine the minimum following distance. If the car is following another car in the lane, the reference position is the rear of the lead car. If the car is leading the lane, the reference position is:

Figure 1: Physics of Motion under Constant Acceleration

$$a(t) = c \qquad a \text{ is acceleration, } c \text{ is a constant} \tag{1}$$
$$v(t) = \int a(t)\,dt = \int c\,dt = ct + v_0 \qquad v \text{ is velocity, } v_0 \text{ is initial velocity} \tag{2}$$
$$x(t) = \int v(t)\,dt = \int (ct + v_0)\,dt = \tfrac{1}{2}ct^2 + v_0 t + x_0 \qquad x \text{ is position, } x_0 \text{ is initial position} \tag{3}$$

the current stop light if the light is red or yellow or if the car is turning left and the intersection is not clear; the rear of traffic in the next lane if the light is green and the intersection is clear of traffic.

2. *Determine slow down distance if turning.* If the vehicle will be turning at the intersection, the distance from the intersection that the vehicle must begin to decelerate using the vehicles average deceleration is calculated as shown in Figure 2. Typical vehicle acceleration and deceleration values are shown in Table 1.

Figure 2: Distance to Initiate Deceleration to Achieve Ideal Turning Velocity

$$x_d = \tfrac{1}{2}ct_d^2 + v_0 t_d \qquad \text{deceleration distance} \tag{4}$$

ignore x_0 to yield relative distance

where

$$t_d = (v_f - v_0)/c \qquad \text{deceleration time} \tag{5}$$

v_0 = current vehicle velocity

v_f = turning velocity $\approx$ 15 mph

Table 1: Typical Acceleration Values for Average Vehicles

Description (Thorpe 1993b)	**Start Speed** (mph)	**End Speed** (mph)	**Time** (secs)	**Accel** (mph/sec)	**Accel** (ft/sec/sec)
Leisure Accel	0	50	38	1.32	1.91
Normal Accel	0	50	25	2.00	2.9
Fast Accel	0	50	20	2.50	3.63
Leisure Brake	50	0	30	-1.66	-2.42
Normal Brake	50	0	20	-2.50	-3.63
Emergency Brake	50	0	5	-10.00	-14.52

Note: 1 mile/hour = 1.452 feet/sec.

3. *Determine acceleration needed for ideal car spacing.* The acceleration required to achieve ideal car spacing is calculated as shown in Figure 3. Equation (10) simplifies to equation (11) since the elapsed time, t, is 1 second. The calculated acceleration is constrained by the vehicles maximum deceleration and average acceleration to produce a realistic simulation.

Figure 3: Acceleration Required to Achieve Ideal Vehicle Spacing

$$x_f = x_r - d_f \tag{6}$$
$$0 = -x_f + x_r - d_f \tag{7}$$
$$0 = -(\tfrac{1}{2}ct^2 + v_0t + x_0) + x_r - t_fv_f \tag{8}$$
$$0 = -(\tfrac{1}{2}ct^2 + v_0t + x_0) + x_r - t_f(ct + v_0) \tag{9}$$
$$a = (x_r - t_fv_0 - v_0t - x_0)/(t_ft + \tfrac{1}{2}t^2) \tag{10}$$
$$\Rightarrow a = (x_r - t_fv_0 - v_0 - x_0)/(t_f + \tfrac{1}{2}) \qquad \text{using 1 sec intervals} \tag{11}$$

where

a	= constant vehicle acceleration		
d_f	= vehicle following distance		
d_s	= stopped vehicle spacing	2 feet	
l_l	= lead vehicle length		
t_f	= inter-vehicle following time	ranges from 0.5 sec to 2 secs by driver	
v_f	= resulting vehicle velocity		
x_f	= resulting vehicle position		
x_r	= vehicle reference position		
	$= x_l - l_l - d_s$	if following a vehicle	(12)
	= intersection location	if lead vehicle and (light is red or vehicle is turning)	(13)
	= next intersection location	if lead vehicle and light is green and vehicle is not turning	(14)
x_l	= lead vehicle resulting position	vehicles front bumper	

4. *Check resulting speed with speed limit.* The acceleration calculated is used to determine the vehicles new preliminary speed and position. If this speed exceeds the speed limit for the lane, the acceleration is reduced to bring the vehicle within the speed limit. A new vehicle speed and position is recalculated if the acceleration was changed. The speed and position calculations are shown below and simplify because the elapsed time is 1 second.

$$v' = a + v_0 \tag{15}$$
$$x' = \frac{1}{2}a + v_0 + x_0 \tag{16}$$

5. *If the vehicle is turning, should it slow down?* If the vehicle is turning and the new vehicle position is within the distance to the intersection where it needs to slow down, the vehicles new acceleration is recalculated. If the vehicles speed is greater than the desired turning speed, average vehicle deceleration is used. Otherwise the acceleration is adjusted to reach the turning speed. The new speed and position are again recalculated.

6. *Should the car be moved to a new lane?* If the current car is the lead car in the current lane, the updated position is used to determine if the car should be moved to a new lane. In this simulation, the intersection is at the origin of a one dimensional axis and vehicles approach the intersection from left to right. If the cars position is greater than zero, the car must be moved into a lane controlled by another intersection and its velocity and position is recomputed. If the car

is not moving to a new lane, the vehicles speed and position are updated with the most recently computed values.

7. *Determine and report vehicle status.* The status of each vehicle can be recorded as it moves through the simulation. The recording is used to verify the simulator operation and to observe the effects of different control strategies. A sample recording is shown in Figure 4. The car id number is listed followed by the world time, the intersection the vehicle is approaching, the updated velocity and position, current direction of travel, direction of travel when passing through the intersection, current and next intersection traffic light colors, leading reference position, different accelerations calculated and vehicle status (coasting, accelerating or decelerating).

2.2.1 Graphical Display of Simulator

The simulation can be displayed graphically and used as a second verification of the simulator operation. Figure 5 shows a close-up of the simulation at a single intersection. The traffic light color is shown by a green, yellow or red line just in front of where the vehicle would stop for a red light. The id number of the vehicle is displayed on the top of the car. The id number is oriented for viewing from the rear of the car looking forward. The id number is the same id number used in the Vehicle Trace listing (see Figure 4). The car brake lights are red when the vehicle is braking, green when accelerating and blank when coasting. The turn the vehicle will make at the next intersection is shown by an arrow on the hood of the car. If the next intersection is the vehicles destination, the turn indicator is blank. Once a vehicle starts to enter an intersection by breaking the light indicator line, the turn direction indicator is updated to reflect the turn direction at the next intersection. As a car passes through an intersection, the lane pointer is indicating the turn direction at the next intersection, not the intersection it is within.

2.2.2 Graphical Display Manipulation Keys

When graphics is being displayed, the simulation can be paused by pressing any of the keys listed below, except, “G”. The window can be adjusted to zoom in or out to view the simulation at an appropriate scale.

“D”: Dump the display to a file, “carpic.tex”, in postscript format.

“S”: stop or single step the simulation

“G”: ’Go’: let simulation proceed as fast as possible

“+”: zoom in

“-”: zoom out

“up cursor”: move world window up

“down cursor”: move world window down

“left cursor”: move world window left

“right cursor”: move world window right

The scope of the single step control consists of moving all vehicles in the simulation one discrete time step. No attempt is made to pace the simulator display so that time is relatively constant during the simulation. Since the graphics interface is slow, the speed of the simulation slows as more vehicles are injected into the simulation and speeds as vehicles reach their destination. Despite this, it is easy to sense the vehicles slowing and accelerating thru the simulation helping to verify the operation of the simulator.

2.3 SIMULATION REPORTS

The overall performance of the simulator is given by the *Simulation Summary Report.* A sample *Simulation Summary Report* is shown in Figure 6. Times are in hours, minutes and seconds. The number of stops, travel time, wait time and world simulated time can be used to measure the relative effectiveness of different control strategies. To analyze different control strategies, a more detailed report of the simulation is required. A sample of the *Detailed Simulation Report* is shown in Figure 7. These

Figure 4: Sample Trace of Vehicle Movement in Simulator

```
Car:44 07:01:08 to 3rd & Birch:V=25.0, D=-239.99, N to N, R R 1 refpos= -0.1 a1=75.9 a2=0.0 coast
Car:31 07:01:08 to 4th & Birch:V=14.4, D=-10.10, N to N, G G 1 refpos=440.0 a1=171.5 a2=2.9 coast
Car:24 07:01:08 to 5th & Candy:V=24.0, D=5.40, endLan, G G accel
Car:24 07:01:08 reached destination.  Time required=00:01:08, Vehicle Stops= 1
Car:44 07:01:09 to 3rd & Birch:V=25.0, D=-214.95, N to N, R R 1 refpos= -0.1 a1=65.9 a2=2.9 accel
Car:31 07:01:09 to 4th & Birch:V=17.3, D=5.70, endLan, G G accel
Car:31 07:01:09 reached destination.  Time required=00:01:09, Vehicle Stops= 0
```

reports were used to compare the effectiveness of fixed timed light control strategies with an AI technique called Reinforcement Learning (Thorpe 1997).

The test suite used in the Reinforcement Learning experiments consisted of 90 runs for each control strategy. There were two main learning parameters used in the Reinforcement Learning experiments, the learning rate and the eligibility trace decay rate (Thorpe 1997). Each run had a fixed learning rate, α, and eligibility trace decay rate, λ. The values for α ranged from 0.1 to 0.9 and λ ranged from 0.0 to 0.9. There is one *Detailed Simulation Report* for each run. The number of training trials required per run is determined by the experimenter. A trial consists of inserting cars into the network and running the simulation until all cars reach their destination or a maximum time has been exceeded. After a predetermined number of training trials, a testing trial is performed and the results of each testing trial is recorded in the detailed report in a separate record.

The *Detailed Simulation Report* can be used as input into GNUPLOT without modification to produce some basic graphs. The first part of the report (see Figure 7) lists all the inputs that were used for the simulation. The name of the program on the first line of the report is followed by the program version and date and time it was compiled. The date and time the first trial was started is listed next. The remaining inputs are identified by a label and value pair:

seed: The random seed used for the testing trials. The random seeds used for training trials is determined by the system clock.

disp: Graphics usage indicator. A value of zero means graphics was not enabled and a value of 1 indicates that graphics were displayed showing the progress of the simulation. A value of 2 indicates graphics was used and the simulator is initially paused.

xtrl: The control method being used (Thorpe 1997): 0=fixed duration light timing 1=SARSA Learning and 2=SARSA Run.

mode: The controller mode (Thorpe 1997). This is used with SARSA Learning and SARSA Run options: 0=SARSA based on CAR Counts, 1=SARSA based on partitions, 2=Greatest Volume strategy (Used with the SARSA Run option only. It was easier to implement this way).

left: Left turn collision avoidance: 0= No avoidance, 1=Avoidance.

dLght: Direct light control: 0=indirect light control. The controller only knows about red and green lights. Using an intermediate/indirect controller, the lights are forced to cycle through yellow before a green light turn red. 1=direct light control (lights can go from green to red without an intermediate yellow or all-red phase).

injPt: How far down the intersection new cars are injected into a lane of traffic.

cong: Congestion handling indicator: 0=no handling, 1=try to manage congestion. Normally a left turning vehicle must yield to oncoming traffic. During congestion, the traffic may be in grid lock and it may be possible for a vehicle to make a left turn when oncoming traffic stops short of the intersection to avoid blocking an intersection.

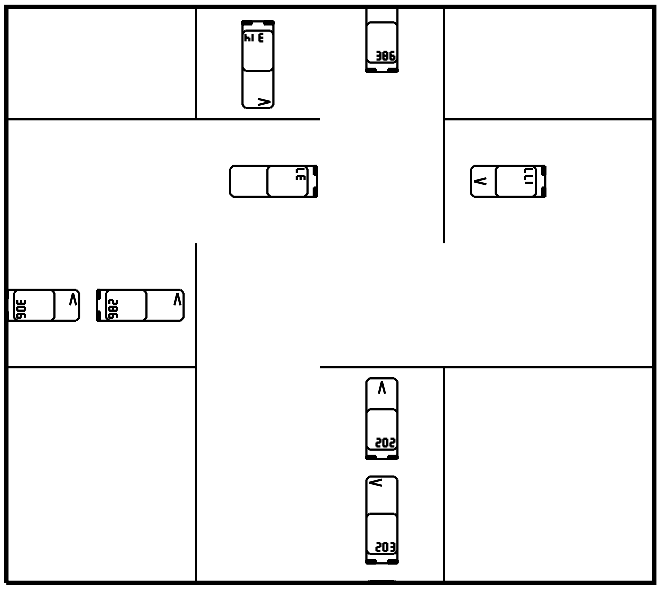

Figure 5: Example of simulator graphics output which is used to verify the operation of the simulator.

4: Treat all intersections as 4-Way stops: 0-no, 1=yes.

r: The reward method used (Thorpe 1997): A value of zero represents a traditional reward of -1 for each time step and a value of 1 represents a reward that varies from -5 to -1 depending on road sensor activations as follows. There are four sensors near an intersection, one in each lane, that are able to detect moving or stopped traffic. A sensor that is activated in a lane with a green light will detect moving traffic and a sensor that is activated in a lane with a red light will detect stopped traffic. The reinforcement is initially -3. If one of the lane sensors corresponding to a green light (travelling traffic) is activated one or more times during the previous second, the reinforcement is incremented by 1 to reward the controller for moving traffic through the intersection. If both green light sensors are activated, the reinforcement is incremented by 2. If a lane sensor corresponding to a red light (stopped traffic) has been activated during the previous second, the reinforcement is decremented by 1 to punish the controller for stopping traffic. If both red light sensors are activated, the reinforcement is decremented by 2. If there is an equal amount of traffic moving and waiting at an intersection, a neutral reward of -3 is assigned.

lrnx: The intersection learning will occur at. 0=all intersections. Otherwise the intersection indicated will be used for learning. Intersections are loaded and counted with the first being zero. It is not possible to train solely on intersection zero.

endIX: End intersection for learning. 0=any intersection otherwise the intersection specified.

sprd: Spread cars out in the intersection during learning: 0=no and implies learning under normal test scenarios. 1=yes and is used for learning on a single intersection.

trace: Does each intersection have its own eligibility traces (Thorpe 1997): no: common traces are used for all intersections, 1=yes: each intersection has its own set of eligibility traces.

Figure 6: Sample *Simulation Summary Report*

```
Simulation Statistics Report
----------------------------
A) Grand Totals
1.  Total Travel Distance (feet) .....  60500
2.  Total number of stops ...........  12
3.  Total Travel Time ...............  00:40:18
4.  Total Wait Time .................  00:01:00
5.  Total Cars ......................  50
B) Averages
1.  Average Travel Distance (feet) ...  1210.0
2.  Average number of stops .........  0.240
3.  Average Travel time .............  00:00:48
4.  Average Wait time ...............  00:00:01
C) Simulation Time vs Processor Time
1.  World Time simulated ............  00:01:29
2.  Processor Wall Clock Time ........  00:00:01
3.  Vehicle Route Finding Time .......  00:00:00
4.  Wall Clock - Simulation Time .....  00:00:01
```

vals: Does each intersection have its own state-action values (Thorpe 1997): 0=no: common state-action values are used for all intersections, 1=yes: each intersection h:s its own set of state-action values.

svSt: Should the traces be reset to zero if the simulator goes tarough an intermediate goal state: 0=no, 1=yes.

any: Is any state updated? 0=no, 1=yes.

decr: Should learning occur only if the simulation passes through states that are closer to the goal? 0=no, 1=yes.

minUpVal: The minimum state value required for state-action value updates to occur. Using a value of 1 prevents goal states from being updated.

yelLrn: Is learning allowed during yellow or all-red light phases? 0=no, 1=yes.

actB4Mn: Can new actions be chosen before the minimum light duration has elapsed? 0=no, 1=yes.

lrnB4Mn: Can learning occur before the minimum light duration has elapsed? 0=no, 1=yes.

lnkLst: Are eligibility traces implemented using a linked list rather than an array? 0=no, 1=yes.

forceMinDur: Is the minimum light duration enforced? 0=no, 1=yes.

forceMaxDur: Is the maximum light duration enforced? 0=no, 1=yes.

minTrc: The minimum eligibility trace value required for updates to occur.

Lane Partitions boundaries: Indicates the number of lane partitions and their locations in feet from an intersection.

Car Count Partition Boundaries: Lists the partition number followed by the maximum number of cars in a lane that will activate that partition.

Lane Duration Boundaries: Lists the Lane State Duration boundaries in seconds. The Lane State Duration indicates the number of seconds that the state value for an intersection has not changed.

lightStates: The number of light states used in determining the current state of a state action pair.

LightDuration Boundaries: Lists the boundaries in seconds to set the Light Duration state based on the number of seconds since the last time the lights changed at an intersection.

Wait Duration Boundaries: Lists the boundaries in seconds to set the Wait Duration state based on the number of seconds cars have been waiting at a red light.

Light Action Duration Boundaries: Lists the boundaries for choosing minimum light timings in seconds when changing light colors.

lightActStates: The number of light states used in determining the action part of a state action pair.

The second part of the *Detailed Simulation Report* (see Figure 7) lists the testing results for a run. There is one line for each testing trial in the run. Only a portion of the items reported are shown in the sample report. The items listed in the report for each testing trial are: testing trial number, learning rate: alpha, eligibility trace decay rate: lambda, the number of steps to complete the simulation, the number of cars in the simulation, how many cars were still traveling at the end of the simulation, the total number of stops made by all cars, the total travel time (including the time traffic is stopped waiting to move) in seconds of all cars, the total time of all cars spent waiting for traffic to move, the average number of stops made by each car, the average travel time per car in seconds, the average wait time per car in seconds, the average number of light changes per intersection, the average time in seconds per sensor that road sensors are depressed in a discrete time step, the average number of hits per sensor that road sensors experience per discrete time step, the average time per vehicle spent braking and accelerating in seconds, the minimum, average and maximum green light durations in seconds, the state-action value file name, the random seed used during the testing trial, the total number of nonzero state-action values, and the minimum state-action value.

Figure 7: Sample *Detailed Simulation Report*

```
#
# Lightc (03.18.05 19970119 113721) Started: Tue Feb 11 11:30:10 1997
# seed:103 disp:0 xtrl:2 mode:1 left:1 dLght:0 injPt:100 cong:1 4:0
# r:0 lrnX:0 endIX:0 sprd:0 trace:0 vals:0 svSt:0 any:0 decr:0 minUpVal:1
# yelLrn:0 actB4Mn:0 lrnB4Mn:1 lnkLst:0 forceMinDur:1 forceMaxDur:0 minTrc:1e-09
# The 0 Lane Partition bounds are:
# The 8 Car Count bounds(3 Parts): 0:0 1:1 2:10 3:20 4:30 5:35 6:40 7:45
# The 0 Lane Duration bounds (0 Parts):
# lightStates:2
# The 0 Light Duration bounds (0 Parts):
# The 0 Wait Duration bounds (0 Parts):
# The 8 Light Action Duration bounds (3 Parts): 0:0 1:5 2:10 3:15 4:20 5:25 6:30 7:40
# lightActStates:2
#
# Description  numStates  Partitions  Offsets  Multiplier
# -----------  ---------  ----------  -------  ----------
# NS CountCar      8          3          0        256
# EW CountCar      8          3          3         32
# Light State      2          1          6         16
# Duration Ac      8          3          7          2
# Light Act        2          1         10          1
# Total States    2048
#
#   1      2      3      4     5    6      7      8        9       10        11     12        13          14     15        16        17        18        19
#                      Total        Still          (secs)   (secs)           (secs)  (secs)
# TRIAL  alpha  lambda Steps  #cars TRAV  #stops TravTime WaitTime AvgStops AvgTrav AvgWait AvgChanges AvgSense AvgHits AvgBrake AvgAccel minGreen avgGreen
     1  0.1000  0.4000  666   500    0    1518    90673    43037     3.04     181   86.07      25.38   0.2076  0.2365       39        48     3.81    23.33
    10  0.1000  0.4000  933   500    0    1709   119981    67003     3.42     239  134.01      45.19   0.1914  0.2112       45        52     3.81    17.70
    20  0.1000  0.4000  606   500    0    1957   101732    42624     3.91     203   85.25      44.19   0.2407  0.2706       55        54     4.00    10.76
    30  0.1000  0.4000  706   500    0    1662    94950    39868     3.32     189   79.74      44.38   0.1780  0.2047       50        51     3.00    12.96
    40  0.1000  0.4000  665   500    0    2001   103070    40535     4.00     206   81.07      63.88   0.2189  0.2453       59        56     3.00     7.44
    50  0.1000  0.4000  568   500    0    1378    87214    36809     2.76     174   73.62      30.56   0.2203  0.2527       43        49     4.94    15.65
```

3. Simulator Controls and Inputs

The MS-DOS batch and OS/2 command files used to run the simulation during reinforcement learning (Thorpe 1997) training and testing are shown in Section 3.1 through Section 3.9. The inputs that control the learning parameters, traffic strategy being tested, and define the traffic environment are shown in Section 3.10.

3.1 Main Batch File "sim.bat"

Training and testing for a single control strategy is initiated and restarted using the "sim.bat" file. The "sim.bat" file executes a program, "bldskl", to examine how much testing and training has already occurred. It builds a new batch file so that testing and training resumes where it left off previously. Since a run of 4000 or more trials may require several hours to complete and there are 90 runs for each control strategy being tested (one for each alpha and lambda combination), training and testing will inevitably be interrupted. Having an automated restart mechanism has proven very helpful. The "sim.bat" file is shown below:

```
bldskl bat
simr
```

The inputs to the "bldskl" program are:

1. A skeleton of 90 runs that will be tested for each control strategy (see Section 3.2),
2. A skeleton of the trials that are executed for each run (see Section 3.3), and
3. State-action value files that are produced before each testing trial.

The outputs of the "bldskl" program are:

1. A batch file, "simr.bat", that will invoke the remaining runs needed to finish training and testing the current traffic control strategy (see Section 3.4), and
2. A batch file, "trialx.bat", used to finish the learning and testing trials of the current run if it is incomplete (see Section 3.5).

If a state-action value file exists for a testing trial, that testing trial is considered complete. If a state-action value file is missing, it marks the start where training and testing must begin. After the "bldskl" program has completed, the "simr.bat" file is invoked to finish up the training and testing of the current run.

3.2 Main Batch Skeleton File "simr.skl"

A portion of the "simr.skl" file which is used as input to the "bldskl" program (see Section 3.1) is shown below. It is used to determine where to restart training and testing when training and testing has been interrupted.

```
;inputs to this bat file - none
;
;inputs to next batch file:
;rem
;%1 = alpha=learning rate
;%2 = lam=eligibility trace decay rate
;%3 = root file name that will be used for report and state/action value files
; if the root file name is 010020 (alpha=.1 and lambda=.2)
; the detailed reports will have file names of:
; R0010020.RP1 for testing with 100 cars
; R0010020.RP2 for testing with 500 cars
; R0010020.RP3 for testing with 1000 cars
; V010020.000 for the state/action value file after the first testing trial which occurs
;  after the first 20 training trials
; V010020.001 for the state action value file after the second testing trial
; V010020.239 for the state/action value file after the last testing trial
;  (assuming there are 4000 learning trials per run
;
;           alp lam   root
call trials 0.1 0.0 010000
call trials 0.1 0.1 010010
call trials 0.1 0.2 010020
call trials 0.1 0.3 010030
call trials 0.1 0.4 010040
call trials 0.1 0.5 010050
call trials 0.1 0.6 010060
call trials 0.1 0.7 010070
call trials 0.1 0.8 010080
call trials 0.1 0.9 010090
;
call trials 0.2 0.0 020000
call trials 0.2 0.1 020010
call trials 0.2 0.2 020020
...
call trials 0.8 0.7 080070
call trials 0.8 0.8 080080
call trials 0.8 0.9 080090
;
call trials 0.9 0.0 090000
call trials 0.9 0.1 090010
call trials 0.9 0.2 090020
call trials 0.9 0.3 090030
call trials 0.9 0.4 090040
call trials 0.9 0.5 090050
call trials 0.9 0.6 090060
call trials 0.9 0.7 090070
call trials 0.9 0.8 090080
call trials 0.9 0.9 090090
```

3.3 Trials Skeleton Batch File “trials.skl”

A portion of the “trials.skl” file which is used as input to the “bldskl” program (see Section 3.1) is shown below. It is used to determine where to restart training and testing when training and testing within a run has been interrupted. If no training has occurred for a run, the simulation starts at the beginning by calling the “triali.bat” file. The “triali.bat” is called at the beginning of each run. It lists the input parameters used, the start time of the first testing trial and performs a testing trial using random actions by using state-action values of all zeros. The first training trial begins with all state-action values initialized to zero. Subsequent training and testing values are initialized with the state-action values from the most recent training trial.

The “trial.bat” and “trialt.bat” files are used to perform the remaining training and testing trials for a run. The “trialt.bat” file lists the input parameters at the end of a run along with the time the final testing trial was started. The input parameters to the “triali.bat”, “trial.bat” and “trialt.bat” files are described in the listing below.

The “chkerase” programs after the “trialt.bat” call are used to search the reports for the testing trials that produced the best results for minimizing the number of time steps for all vehicles to exit the summary. The state-action values for those testing trials are saved and the remaining state-action value files are erased.

```
;rem inputs to this bat file
;rem
;rem %1 = alpha
;rem %2 = lam
;rem %3 = root file name used for detailed reports and state/action value files
;
;
;rem inputs to triali and trial.bat :
;rem run a single trial of training and testing
;rem
;rem %1 = alpha= learning rate
```

```
;rem %2 = lam= eligibility trace decay rate
;rem %3 = ns cars (used only for training on a single intersection)
;rem %4 = ew cars (used only for training on a single intersection)
;rem %5 = report and state/action value file root names
;rem %6 = state/action value file 3 byte extension
;rem %7 = number of training trials to run
;         (only used for training on a single intersection)
;rem %8 = the testing trial number
;
erase d%3.*
erase dump
;
;rem        alp lam  ns  ew root.ext   --runs--
call triali  %1  %2   0   1   %3 000   20    20
call trial   %1  %2  10   1   %3 001   20    40
call trial   %1  %2   1  20   %3 002   20    60
call trial   %1  %2  20   1   %3 003   20    80
call trial   %1  %2   1  30   %3 004   20   100
call trial   %1  %2  30   1   %3 005   20   120
call trial   %1  %2   1  40   %3 006   20   140
call trial   %1  %2  40   1   %3 007   20   160
call trial   %1  %2   1  50   %3 008   20   180
call trial   %1  %2  50   1   %3 009   20   200
;
call trial   %1  %2  10   1   %3 010   20   220
call trial   %1  %2  10  10   %3 011   20   240
...
call trial  %1  %2  40  50   %3 228   10  3890
call trial  %1  %2  50  40   %3 229   10  3900
;
call trial  %1  %2  50   1   %3 230   10  3910
call trial  %1  %2  10  50   %3 231   10  3920
call trial  %1  %2  50  20   %3 232   10  3930
call trial  %1  %2  20  50   %3 233   10  3940
call trial  %1  %2  50  30   %3 234   10  3950
call trial  %1  %2  30  50   %3 235   10  3960
call trial  %1  %2  50  40   %3 236   10  3970
call trial  %1  %2  40  50   %3 237   10  3980
call trial  %1  %2  50  50   %3 238   10  3990
call trialt %1  %2   1  50   %3 239   10  4000
;
chkerase R0%3.RP1 x
chkerase R0%3.RP2 x
chkerase R0%3.RP2 x
;
erase V%3.*
```

3.4 Resulting Batch File "simr.bat"

A sample "simr.bat" file which is produced by the "bldskl" program (see Section 3.1) is shown below. It is used to restart training and testing at the correct point when training and testing has been interrupted.

```
call trialx 0.8 0.3 080030
call trials 0.8 0.4 080040
call trials 0.8 0.5 080050
call trials 0.8 0.6 080060
call trials 0.8 0.7 080070
call trials 0.8 0.8 080080
call trials 0.8 0.9 080090
call trials 0.9 0.0 090000
call trials 0.9 0.1 090010
call trials 0.9 0.2 090020
call trials 0.9 0.3 090030
call trials 0.9 0.4 090040
call trials 0.9 0.5 090050
call trials 0.9 0.6 090060
call trials 0.9 0.7 090070
call trials 0.9 0.8 090080
call trials 0.9 0.9 090090
```

3.5 Finishing an Incomplete Run, Trials Batch File "trialx.bat"

A sample "trialx.bat" file which is produced by the "bldskl" program (see Section 3.1) is shown below. It is used to restart training and testing within the last run when a run has been interrupted.

```
call trial  %1  %2  40  40   %3 227   10  3880
call trial  %1  %2  40  50   %3 228   10  3890
call trial  %1  %2  50  40   %3 229   10  3900
call trial  %1  %2  50   1   %3 230   10  3910
call trial  %1  %2  10  50   %3 231   10  3920
call trial  %1  %2  50  20   %3 232   10  3930
call trial  %1  %2  20  50   %3 233   10  3940
call trial  %1  %2  50  30   %3 234   10  3950
call trial  %1  %2  30  50   %3 235   10  3960
call trial  %1  %2  50  40   %3 236   10  3970
call trial  %1  %2  40  50   %3 237   10  3980
call trial  %1  %2  50  50   %3 238   10  3990
call trialt %1  %2   1  50   %3 239   10  4000
chkerase R0%3.RP1 x
chkerase R0%3.RP2 x
chkerase R0%3.RP2 x
rem
erase V%3.*
```

3.6 Trials Batch File "trials.bat"

A portion of a "trials.bat" file is shown below. This file is invoked to execute the trials for an entire run. The first batch file invoked, "triali.bat", is an initialization batch file. The last batch file invoked, "trialt.bat", is a termination batch file. The remaining batch files invoked, "trial.bat" are used for the intermediate training and testing.

```
rem inputs to this bat file
rem
rem %1 = alpha
rem %2 = lam
rem %3 = root file name used for detailed reports and state/action value files
rem
rem inputs to triali and trial.bat :
rem run a single trial of training and testing
rem
rem %1 = alpha= learning rate
rem %2 = lam= eligibility trace decay rate
rem %3 = ns cars (used only for training on a single intersection)
rem %4 = ew cars (used only for training on a single intersection)
rem %5 = report and state/action value file root names
rem %6 = state/action value file 3 byte extension
rem %7 = number of training trials to run
rem      (only used for training on a single intersection)
rem %8 = the testing trial number
rem %9 = epsilon exploration value
rem
rem             alp lam  ns  ew root.ext runs
call triali.bat  %1  %2   5   1   %3 000   20    20 .1
call trial.bat   %1  %2  10   0   %3 001   20    40 .1
call trial.bat   %1  %2   5  20   %3 002   20    60 .1
call trial.bat   %1  %2  20   0   %3 003   20    80 .1
call trial.bat   %1  %2   5  30   %3 004   20   100 .1
call trial.bat   %1  %2  30   0   %3 005   20   120 .1
call trial.bat   %1  %2   5  40   %3 006   20   140 .1
call trial.bat   %1  %2  40   0   %3 007   20   160 .1
call trial.bat   %1  %2   5  50   %3 008   20   180 .1
call trial.bat   %1  %2  50   0   %3 009   20   200 .1

call trial.bat   %1  %2  10   1   %3 010   20   220 .1
call trial.bat   %1  %2  10  10   %3 011   20   240 .1
call trial.bat   %1  %2  10  20   %3 012   20   260 .1
call trial.bat   %1  %2  20  10   %3 013   20   280 .1
call trial.bat   %1  %2  10  30   %3 014   20   300 .1
call trial.bat   %1  %2  30  10   %3 015   20   320 .1
call trial.bat   %1  %2  10  40   %3 016   20   340 .1
call trial.bat   %1  %2  40  10   %3 017   20   360 .1
call trial.bat   %1  %2  10  50   %3 018   20   380 .1
call trial.bat   %1  %2  50  10   %3 019   20   400 .1
...
call trial.bat   %1  %2  50   1   %3 230   10  3910 .0
call trial.bat   %1  %2  10  50   %3 231   10  3920 .0
call trial.bat   %1  %2  50  20   %3 232   10  3930 .0
call trial.bat   %1  %2  20  50   %3 233   10  3940 .0
call trial.bat   %1  %2  50  30   %3 234   10  3950 .0
call trial.bat   %1  %2  30  50   %3 235   10  3960 .0
call trial.bat   %1  %2  50  40   %3 236   10  3970 .0
call trial.bat   %1  %2  40  50   %3 237   10  3980 .0
call trial.bat   %1  %2  50  50   %3 238   10  3990 .0
call trialt.bat  %1  %2   1  50   %3 239   10  4000 .0
```

3.7 Initialization Trial Batch File "triali.bat"

The initialization trial batch file, "triali.bat", is shown below. Testing and training is performed with 100, 500 and 1000 cars corresponding to light, medium and heavy traffic loads. The first testing trial uses state-action values of all zeros. This results in a random action control strategy since all state-action values are equal. The first training trial also uses state-action values that are initialized to zero. At the end of each training trial, the updated state-action values are written to the "vals" file. The "vals" file is used to initialize the state-action values for future training and testing trials. After ten training trials at different traffic loads, three training trials are executed for 100, 500 and 1000 car loads.

The "-title" parameter causes the simulator inputs to be listed in the *Detailed Simulation Report* (see Figure 7). The "-init vals" parameter initializes the state-action values from the "vals" file for testing and training. The "-trials" parmameter indicates the next parameter specifies how many trials will be executed for one program invocation. The "-cars" parameter indicates the next parameter specifies how many vehicles are injected into the simulation during training or testing. The "-seed" parameter indicates the next parameter specifies the random seed to use for training and testing. A value of zero means the random seed is taken from the system clock time. If the "-seed" parameter is missing, the random seed will be taken from the *Traffic Environment Definition File* (see Section 3.10). The "-tr" parameter indicates a testing trial is being executed and the next parameter specifies the testing trial number which is listed in the *Detailed Simulation Report* (see Figure 7). The "-mxstep" parameter indicates the next parameter overrides the maximum number of discrete time steps to be executed as set in the *Traffic Environment Definition File* (see Section 3.10).

```
rem run a series of training trials followed by a testing trial for each traffic load
rem
rem %1 = alpha
rem %2 = lam
rem %3 = ns cars
rem %4 = ew cars
rem %5 = value savefile 8 digit name
rem %6 = value savefile 3 digit name
rem %7 = trials
rem %8 = testing trial number
rem
erase v%5.*
rem
rem run testing trials with q(s,a) all equal zero => random action test
rem The -title option reports the input parameters and date and time the run was begun
rem
lightc traffic2.d01  -cars 100 -title -tr 1 -root %5 -ext RP1 -alp %1 -lam %2
lightc traffic2.d01  -cars 500 -title -tr 1 -root %5 -ext RP2 -alp %1 -lam %2
lightc traffic2.d01  -cars 1000 -title -tr 1 -root %5 -ext RP3 -alp %1 -lam %2 -mxstp 2399
rem
rem perform a series of training trials with different traffic loads
rem NOTE: The first training trial does not specify an init value file => q(s,a) are initially zero
rem
lightc traffic.d01            -alp %1 -lam %2 -ns %3 -ew %4 -trials 2 -cars 100 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 1 -cars 500 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 1 -cars 500 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 1 -cars 1000 -mxstp 2399 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 2 -cars 100 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 1 -cars 500 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 1 -cars 500 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 1 -cars 1000 -mxstp 2399 -seed 0
rem
copy vals v%5.%6
copy dump d%5.%6
erase dump
rem
rem perform a testing trial for each traffic load with the current state/action values
rem
lightc traffic2.d01 -init v%5.%6 -cars 100  -tr %8 -root %5 -ext RP1 -alp %1 -lam %2
lightc traffic2.d01 -init v%5.%6 -cars 500  -tr %8 -root %5 -ext RP2 -alp %1 -lam %2
lightc traffic2.d01 -init v%5.%6 -cars 1000 -tr %8 -root %5 -ext RP3 -alp %1 -lam %2 -mxstp 2399
```

3.8 Intermediate Trials Batch File "trial.bat"

The intermediate trial batch file, "trial.bat", is shown below. It performs a series of training trials starting with the latest state-action values followed by testing trials.

```
rem perform a series of training trials with different traffic loads
rem
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 2 -cars 100 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 1 -cars 500 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 1 -cars 500 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 1 -cars 1000 -mxstp 2399 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 2 -cars 100 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 1 -cars 500 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 1 -cars 500 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 1 -cars 1000 -mxstp 2399 -seed 0
copy vals v%5.%6
copy dump d%5.%6
erase dump
rem
rem perform a testing trial for each traffic load with the current state/action values
rem
lightc traffic2.d01 -init v%5.%6 -cars 100 -root %5 -ext RP1 -tr %8 -alp %1 -lam %2
lightc traffic2.d01 -init v%5.%6 -cars 500 -root %5 -ext RP2 -tr %8 -alp %1 -lam %2
lightc traffic2.d01 -init v%5.%6 -cars 1000 -root %5 -ext RP3 -tr %8 -alp %1 -lam %2 -mxstp 2399
```

3.9 Termination Trial Batch File "trialt.bat"

The termination trial batch file, "trialt.bat", is shown below. It is the same as the "trial.bat" file except the simulator inputs are listed in the *Detailed Simulation Report* (see Figure 7).

```
rem perform a series of training trials with different traffic loads
rem
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 2 -cars 100 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 1 -cars 500 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 1 -cars 500 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 1 -cars 1000 -mxstp 2399 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 2 -cars 100 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 1 -cars 500 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 1 -cars 500 -seed 0
lightc traffic.d01 -init vals -alp %1 -lam %2 -ns %3 -ew %4 -trials 1 -cars 1000 -mxstp 2399 -seed 0
copy vals v%5.%6
copy dump d%5.%6
erase dump
rem
rem perform a testing trial for each traffic load with the current state/action values
rem The -title option reports the input parameters and date and time the final run was executed
rem
lightc traffic2.d01 -init v%5.%6 -cars  100 -title -tr %8 -root %5 -ext RP1 -alp %1 -lam %2
lightc traffic2.d01 -init v%5.%6 -cars  500 -title -tr %8 -root %5 -ext RP2 -alp %1 -lam %2
lightc traffic2.d01 -init v%5.%6 -cars 1000 -title -tr %8 -root %5 -ext RP3 -alp %1 -lam %2 -mxstp 2399
```

3.10 Traffic Environment Definition File "traffic.d01"

A sample *Traffic Environment Definition File* is listed next. It is used to set some simulation parameters and to define the traffic environment. The first character in each record defines the type of data being loaded. A semicolon in column one is a comment record. Comments follow and document a record the first time a new type of data record is encountered. Simulation parameter records occur first followed by intersection definition records and lane records which connect intersections. Intersections must be defined before they can be connected by lanes.

```
;
; traffic.d01 - traffic light simulation control and environment data file.
;
; The format of the input data is described as it is first encountered.
;
; (semicolon in first position is a comment)
;
; simulation control record is first
; ---------------------------------------------------------------------------------------
S 07:00 103    0     2     1   1       0    100    1200       1       0      2       2
; start seed dsp  Xtrl xMode avoid  Direct inject  max    congested  all  #curAct #nxtAct
;                                  Control Point   steps  left turn  4way  values  values
; ---------------------------------------------------------------------------------------
;   col 1 = S (simulation control record)
;   nxt field = start time of simulation (hh:mm)
;   nxt field = random number generator seed (if non zero, it overrides
;               the random seed using the system time or command line parm.)
;   nxt field = 0=no graphics, 1=display graphical status, 2=pause
;   nxt field = Xtrl=controller type: 0=fixed duration light timing,
;                                     1=SARSA Learning,
;                                     2=SARSA Run,
;                                     3=Neural Net eval and action network (not implemented).
;   nxt field = Controller Mode. This is used with SARSA Learning and
;               SARSA Run options:  0=SARSA based on CAR Counts,
;                                   1=SARSA based on partitions,
;                                   2=Greatest Volume strategy (Use with the
;                                     SARSA Run option only).
;   nxt field = collision avoidance method
;               0= no avoidance
;               1= avoid oncoming traffic
;   nxt field = Direct Light Control
;               1 = direct
;               0 = indirect
;   nxt field = point at which to inject vehicles into simulation in feet
;   nxt field = max steps per trial
;   nxt field = allow congested left turn (0=no, 1=yes)
;   nxt field = treat all intersections as 4 way stops?
;   nxt field = number of current light action states (not partitions)
;   nxt field = number of next light action states (not partitions)
;!!! NOTE !!!!!! need to re implement using lane information
;
;
; Reward and Learning Parameters
; ---------------------------------------------------------------------------
R  0   0   0    0   1   0  0   0   0    1    0   1   0   0   0  .000000001   1
;rew Lrn End Sprd own own sv yel Act  Lrn  lnk frc for upd decr min        min
;typ Int Int cars trc val st lrn B4Mn B4Mn Lst min max any only trace      upd
```

```
;                                         Trc                              val
; ---------------------------------------------------------------------------
;   col 1 = R (Reward and Learning Parameters)
;   nxt field = reward scheme
;                    0 = -1 for each time step
;                    1 = -5 to -1 depending on sensors
;   nxt field = intersection Learning occurs (0=all else indicated)
;               intersections are loaded and counted with the first being zero
;   nxt field = End intersection for learning
;               0=any else the intersection specified
;   nxt field = spread cars during learning
;               0=no and implies learning under test scenario
;                 (cars injected at start and inserted when safe)
;               1=yes and implies special set up/learning on one intersection
;   nxt field = does each intersection have its own eligibility traces
;               0=no=common, 1=yes
;   nxt field = does each intersection have its own state-action values
;               0=no=>common vals, 1=own values
;   nxt field = reset eligibility traces to zero if we go through an intermediate goal state
;               0=no, 1=yes
;   nxt field = learn during yellow or all red phases? (except 4way stop)
;   nxt field = new actions allowed while cur duration < minDuration
;   nxt field = learning allowed while curDuration < minDuration
;   nxt field = update old state if curDuration < minDuration (non-Markov)
;   nxt field = use linked list for eligibility trace updates
;               0=use an array, 1=use linked list
;   nxt field = force min duration during duration learning: 0=no, 1=yes
;   nxt field = force max duration during duration learning: 0=no, 1=yes
;   nxt field = update any state: 0=no, 1=yes
;   nxt field = update decreasing states only: 0=no, 1=yes
;   nxt field = minimum eligibility trace values for updates
;   nxt field = minimum state update value. If the state value is less than the
;               indicated value, no state update occurs. This prevents goal states
;               from being updated when using a value of 1.
;
;
; Debug Control Parameters
; -----------------------------------------------------
B  0   0   0   0     0      0     0      0       0
; bug tag ibg fTrace fOpen Integ nonNull abPause DispMem
; Col 1 = B = debug indicator
; nxt field = debug level
; nxt field = count of vehicles to initially tag (display progress)
; nxt field = idas debug level
; nxt field = file trace switch
; nxt field = file trace open/close switch
; nxt field = Link List integrity check switch
; nxt field = non Null pointer check switch
; nxt field = pause at abend indicator (0=no, 1=yes)
; nxt field = display memory report or not 0=1, 1=yes
;
;
; Network specification defaults (NOT IMPLEMENTED)
; -----------------------------------------------
N    0   0  0  0  0  0  0  0  0  0   0   0 0 0 0
; xArch p1 p2 p3 p4 p5 p6 p7 p8 p9 p10
; ---------------------------------
;   col 1 = N (Network defaults
;   nxt field = architecture scheme: (used with Neural Net only)
;                                    0= current intersection sensors only
;                                    1= 0 and use sensors 1 block away
;                                    2= 1 and also sensors 2 blocks away
;   nxt fields= control parm 1: # first hidden layer nodes
;                            2: # second hidden layer nodes
;                            3: number of delays
;                            4: number of seconds each delay holds
;                            5: output node type 0=sigmoid, 1=linear
;                         6-10: reserved (alpha, beta, momentum)
```

```
;
; vehicle injection rate
; ---------------------
V 0 0   2 2   07:00  10 07:05  30 07:10  30 07:15   0
; XDist Lanes time/rate time/rate time/rate time/rate
; ---------------------------------
;   col 1 = V (vehicle injection rate record)
;   nxt 2 fields = initial XDist for initial inject and subsequent
;       -1=lane length
;       0-999 = specified length
;   nxt 2 fields = inBound or outbound lanes for initial and subsequent
;       0=outbound lanes
;       1=in inbound lanes
;       2=safe entry into outBound Lanes
;   nxt 4 field pairs = time, car injection rate (cars / minute)
;
;
; Colors to use for graphics display
; ---------------------------------
C   1     12  10  14     7    2    3   13   15  0     0   2200
; BkGrnd Red Grn Yel Block car1 car2 car3 car4 Tx    Ty   Scale
;   col 1 = C (graphics color scheme control record)
;   colors represented are:
;          display background color
;          red, green and yellow lights on traffic lights and cars
;          Block outline color
;          4 car body outline colors to use
; Next fields are x any Translate values and scale adjust initial settings
; for graphics window display
;
;
; default min max light timing is loaded next
; ------------------------------------------
D M   2 20    10 45     2 20    25   10
;   min max  min max  min max  INC% DEC%
;   -------  -------  -------
;     LEFT    RIGHT    THRU
; ------------------------------------------
;   col 1 = D (default type record)
;   col 3 = M (min max light control indicator)
;   nxt 6 fields=min max green times for left, right and thru in seconds
;   nxt 2 fields=adj percent for increase and decrease time
;
;
; F traffic2.d02 x
; End this file and go to the next file
;
;
P 0 50 110 220
; Lane partition sizes are loaded next. Lane partitions are optional
; and only used with SARSA based on partitions, otherwise they
; are ignored. Up to four partitions can be specified.
; There are 4 partitions in the example below. The first partition
; is from 0 to 50 feet, the second from 50 feet to 110, the third
; is from 110 to 220 feet and the fourth is anything over 220 feet.
; The state value is determined by the presence or absence of
; cars within the partitions. The first partition has a value of one
; and each subsequent partition has a value that is double the previous
; partition. The total state value is calculated by adding up partition
; values for the partitions that have cars within them.
; P 0 50 110 220
;
;
;Z 0 1 10 20 30 35 40 45
; Car Count Partition boundaries are optionally Loaded next. They are
; only used with SARSA based on partitions, otherwise they are ignored.
; Up to 16 boundaries can be specified. There are 4 boundaries in the
; example below. If there are no cars in the lane, the first state
; boundary is active. If there are 1 to 9 cars in the lane, the second
; boundary is active. The third boundary is active if there are 10 to
; 19 cars in the lane and the fourth boundary is active if there are
; 20 or more cars in the lane. The state value is set to the boundary
; that is active and subtracting 1.
; Z 0 1 10 20
;
;
;A 0 50 10 20 30 40 50
; Lane State Duration boundaries are optionally Loaded next. They are
; only used with SARSA based on partitions, otherwise they are ignored.
; Up to 16 boundaries can be specified. The Lane State Duration is based
; on the amount of time in seconds that the state of the lanes leading
; into an intersection have not changed. If car counting is used and the
; car count state value does not change from one discrete step to
; another, the Lane State Duration Counter is incremented,
; otherwise the Lane State Duration Counter is reset to 1. The Lane
; State Duration value is calculated based on the boundary that the Lane
; State Duration Counter falls in. There are 4 boundaries in the example
; below. If the Lane State has not changed for 0 to 4 seconds, the Lane
; State Duration value is 0. If the Lane State has not changed for 5 to 9
; seconds, the Lane State Duration value is 1. If the Lane State has not
; changed for 10 to 19 seconds, the Lane State Duration value is 2. If
; the Lane State has not changed for 20 seconds or more, the Lane State
; Duration value is 3.
; A 0 5 10 20
;
;
;T 0 5 10 15 20 30 45 60
; Time Since Light Change boundaries are optionally Loaded next. They are
; only used with SARSA based on partitions, otherwise they are ignored.
; Up to 16 boundaries can be specified. The Time Since Light Change state
; is based on the amount of time in seconds since the last light change.
; For each discrete time step that the light remains the same, the Light
; Duration Counter is incremented, otherwise the Light Duration Counter
; is reset to 1. The Time Since Light Change state is
; set based on the boundary the Light Duration Counter falls in. There
; are 4 boundaries in the example below. If the Light Duration Counter
; ranges from 0 to 7 seconds, the Time Since Light Change state value is
; 0. If the Time Since Light Change Counter ranges from 8 to 15 seconds,
; the Time Since Light Change state value is 1. If the Time Since Light
; Change Counter ranges from 16 to 31 seconds, the Time Since Light Change
; state value is 2. If the Time Since Light Change Counter is 32 or
; greater, the Time Since Light Change state value is 3.
;T 0 8 16 32
;
;
;W 0 50 10 20 30 40 50
; Car Wait Time at red light boundaries are optionally Loaded next. They
; are only used with SARSA based on partitions, otherwise they are ignored.
; Up to 16 boundaries can be specified. The Car Wait Time state value
; is based on the amount of time in seconds since cars have been waiting
; at a red light. There are 4 boundaries in the example below. If cars
; have been waiting at a red light for 0 to 4 seconds, the Car Wait Time
; state value is 0. If cars have been waiting at a red light from 5 to 9
; seconds the Car Wait Time state value is 1. If cars have been waiting
; at a red light for 10 to 19 seconds, the Car Wait Time state value is
; 2. The Car Wait Time state value is 3 if cars have been waiting at a
; red light for 20 or more seconds.
; W 0 5 10 20
;
;
X 0 5 10 15 20 25 30 40
; Light Duration action boundaries are optionally Loaded next. They are
; only used with SARSA based on partitions, otherwise they are ignored.
; Up to 16 boundaries can be specified. The Light Duration action specifies
; a range of time that lights are set to a given color when choosing
; light control actions. There are 4
; boundaries in the example below. If the first boundary is chosen, the
; time that lights are set to a given color will range from 0 to 9 seconds.
; If the second boundary is chosen, the time that lights are set to a given
; color will range from 10 to 19 seconds. If the third boundary is chosen,
; the time that lights are set to a given color will range from 20 to 29
; seconds. If the fourth boundary is chosen, the time that lights are set
; to a given color will be at least 30 seconds.
; X 0 10 20 30
;
;
; intersection name, light control, min max data is loaded next
;---------------------------------------------------------------
I N 2nd             Birch                2      :  2       E         460 1780
;   NS Name         EW Name         NS Num    EW Num      HEAVY DIR    X    Y
I M   2 20   10 45   2 20   25  10
;     LEFT   RIGHT   THRU   INC DEC
I X   1    N R N    N R N    N R N    N R N  X        1    1    All=Red
I X  25    N G N    N G N    N R N    N R N  G        3  180    N/S=Green
I X   2    N Y N    N Y N    N R N    N R N  X        2    2    N/S=Yellow
I X   1    N R N    N R N    N R N    N R N  X        1    1    All=Red
I X  25    N R N    N R N    N G N    N G N  R        3  180    E/W=Green
I X   2    N R N    N R N    N Y N    N Y N  X        2    2    E/W=Yellow
;   secs   NORTH    SOUTH    EAST     WEST   NSVal  min  max
;          R T L    R T L    R T L    R T L         secs secs
;--------------------------------------------------------------------------
;   col 1 = I (intersection record)
;   col 3 = N (intersection name record) (one per intersection)
;   nxt fld = North South street name
;   nxt fld = East West street name
;   nxt fld = North South street reference number
;   nxt fld = East West street reference number
;   NXT FLD = assigned heavy traffic direction
;   nxt flds= x,y coords of center of intersection
;
;   col 1 = I (intersection record)
;   col 3 = M (absolute min max light controls) (zero or one per intersection)
;   (same format as default min max above)
;
;   col 1 = I (intersection record)
;   col 3 = X (intersection light controls) (zero or more per intersection)
;   nxt fld = initial duration of light setting in seconds
;   nxt 3 flds = North right, thru and left colors
;   nxt 3 flds = South right, thru and left colors
;   nxt 3 flds = East right, thru and left colors
;   nxt 3 flds = West right, thru and left colors
;   nxt fld = north/south light color from a light controller perspective
;   nxt fld = minimum duration of specified light phase
;   nxt fld = maximum duration of specified light phase
;
....
  more intersections
....
;
;
; lane data is loaded next
;---------------------------------------------
L C      0  0     2  2     20   S    A      440     Y        L
;   FromNS,EW  ToNS,EW  Speed  Dir Turn  Length  Sensor  Control
;        (y,x)   (y,x)
;---------------------------------------------
;   col 1 = L (lane record)
;   col 3 = C (lane connector)
;   nxt fld = NS from street reference number
;   nxt fld = EW from street reference number
;   nxt fld = NS to street reference number
;   nxt fld = EW to street reference number
;   nxt fld = lane speed limit in miles per hour
;   nxt fld = travel direction (N S E W)
;   nxt fld = turn direction allowed from lane (left, right, thru, any)
;   nxt fld = lane length
;   nxt fld = sensor indicator
;   nxt fld = control type (L=light, S=Stop sign, R=right of way)
;
L C      0  0     3  2     20   S    A      440    Y        L
L C      0  0     4  2     30   S    A      440    Y        L
L C      0  0     5  2     40   S    A      440    Y        L
;
L C      2  2     2  3     20   S    A      440    Y        L
L C      3  2     3  3     20   S    A      440    Y        L
L C      4  2     4  3     30   S    A      440    Y        L
L C      5  2     5  3     40   S    A      440    Y        L
;
L C      2  3     2  2     20   N    A      440    Y        L
L C      3  3     3  2     20   N    A      440    Y        L
L C      4  3     4  2     30   N    A      440    Y        L
L C      5  3     5  2     40   N    A      440    Y        L
;
;
;
;
L C      0  0     2  2     20   E    A      440    Y        L
L C      0  0     2  3     20   E    A      440    Y        L
L C      0  0     2  4     30   E    A      440    Y        L
L C      0  0     2  5     40   E    A      440    Y        L
;
L C      2  2     3  2     20   E    A      440    Y        L
L C      2  3     3  3     20   E    A      440    Y        L
L C      2  4     3  4     30   E    A      440    Y        L
L C      2  5     3  5     40   E    A      440    Y        L
;
L C      3  2     2  2     20   W    A      440    Y        L
L C      3  3     2  3     20   W    A      440    Y        L
L C      3  4     2  4     30   W    A      440    Y        L
L C      3  5     2  5     40   W    A      440    Y        L
;
```

4. Simulator Module Hierarchy and Strategy Modifications

The module hierachy of the traffic simulator is shown in the following sections. To create new traffic control strategies see Section 4.3.

4.1 Main Module Hierarchy - LIGHTC.c

The *main()* function of the simulator is in file LIGHTC.c. It starts by initializing control parameters to default values, and overriding some control parameters based on the command line options. Then the traffic environment including the traffic network topology and other control parameters are loaded by the *loadenv()* function. After the traffic network has been loaded, the simulation is controlled by the *traffic()* function. The *traffic()* function injects vehicles into the simulation, selects the traffic control strategy being tested, moves the cars through the lanes and intersections into new lanes and manipulates the traffic control lights. The *traffic()* function selects routes using the IDA* algorithm within the *routeSearch()* function. When the simulation has terminated, either because all vehicles have reached their destinations or the maximum number of time steps has elapsed, the final traffic environment is saved by the *saveenv()* function. The output of the final traffic environment was to be used to resume testing where the simulation ended. So far there has been no need to resume testing where the simulation ended and this function could be removed (it is not up to date). Finally the results of the simulation are optionally written to report files for later analysis. The simulation results are usually produced for debugging and testing trials.

```
LIGHTC.c Main program

    init() (in LIGHTINI.c) Set parameters specified on command line
    loadenv() (in LIGHTLOD.c) Load traffic topology
    traffic() (in LIGHTSIM.c) Simultates Traffic Environment (Move car and change lights)
        routeSearch() (in IDAS.c) IDA Star Route Search routines
        utils (in LIGHTUTL.c) Utility functions
    saveenv() (in LIGHTSAV.c) Save final traffic environment
    report() (in LIGHTRPT.c) Report Simulation results
```

4.2 Traffic Simulator Load Environment Module Structure - LIGHTLOD.c

The traffic environment including the traffic network topology and other control parameters are loaded by the *loadenv()* function by reading the *Traffic Environment Definition File* (see Section 3.10). Each of the major subroutines below are invoked by the *loadenv()* function based on the data type record as determined by the first character of the data record being loaded. The data type indication character is listed before the subroutine that is invoked using a case statement like syntax. A

```
loadenv() Load Traffic Environment

    initial() Initialize vars, get current clock time
    S: ldSimParms() Load simulation parms (mode, start time, random # seed)
    D: ldDefault() Load simulation defaults (min max times for light timings)
        M: ldMinMaxLght() Load light traffic default switch timings
            newMnMxLight() Create new min max control element
    I: ldXSection() Load Intersections
        X: ldXLght() Load Light Controls
            NewLXtrl() Create new light control structure
        N: ldXName() Load Intersection Name and NULL the pointers
        M: ldXMinMaxLght() Load Intersection Light Min Max Times
            newMnMxLight() Create new min max control element
    L: ldLane() Load traffic lanes
        NewLaneQ() Create New Lane Q
        FindXHead() Find Intersection to link lane to
    B: ldDebug() Load deBug control parms
```

```
N: ldNetParms() Load network default parms (not implemented)
V: ldVIR() Load vehicle injection rates
C: ldColors() Load graphics display colors
R: ldReward() Load reward and learning parms
P: ldParts() Load lane state partition sizes
Z: ldCarCnt() Load car count state partition vals
A: ldLaneDur() Load lane Duration state partition vals
T: ldLightDur() Load light state duration partitions
W: ldWaitDur() Load wait state duration partitions
X: ldActDur() Load action light duration partitions vals
F: ldFileName() Continue loading environment with a new file name
checkEnv() Check environment load time
    getCurTime() (in LIGHTUTL.c)
    timeDif() (in LIGHTUTL.c)
    EqTime() (in LIGHTUTL.c)
    printElapsed() (in LIGHTUTL.c)
    memRpt() (in LIGHTUTL.c)
Utility Functions in LightLod.c
    FindXHead() find a specific intersection element
    fps2mph() convert feet per second to miles per hour
    getnexttoken() get next token in string
    loadTime() convert string to wall clock time
    mph2fps() convert miles per hour to feet per second
    NewLaneData() create lane data element
    NewLaneQ() create new lane queue element
    NewLXtrl() create new light control element
    NewMnMxLight() create new light min max element
    NewTimeHist() create new time history tracking element
    NewTimeStamp() create new time stamp element
    NewVeh() create new vehicle element
    NewXHead() create new intersection element
    string2int() convert string to integer
    wait() wait for 9999 loop iterations
```

4.3 Traffic Light Simulator Simulation Module Structure - traffic() LIGHTSIM.c

The *traffic()* function determines which control strategy will be used. To implement a new control strategy, add a new function to the xControl switch statement and model the new function after one of the functions in Section 4.4 through Section 4.6.

The *traffic()* function calls routines to inject vehicles into the simulation, selects the traffic control strategy being tested, moves the cars through the lanes and intersections into new lanes and manipulates the traffic control lights. The *traffic()* functions selects routes using the IDA* algorithm within the *routeSearch()* function.

```
traffic() Initialize graphics (if needed) and select control strategy.
    Xez_Initialize() Initialize graphics when enabled.
    Xez_SetBkColor() Set Background color.
    switch (xControl)
        case 0:  TimedLights(); break;
        case 1:  SARSA_Learn(); break;
        case 2:  SARSA_Run(); break;
    Xez_Close() Close graphics windows
```

4.4 Traffic Light Simulator Simulation Module Structure - Timed Lights LIGHTSIM.c

The *Timed Lights* traffic control strategy illustrates the basic flow of the simulator. Initially vehicles are injected into the simulation by selecting a starting location and destination, determining the route from the start to the destination and then placing the car in the correct starting lane. At the beginning of each discrete time step, the simulation optionally injects more vehicles into the simulation and checks for graphics control commands. The linked list of intersections is then traversed. At each intersection the lights are changed if needed based on predetermined timings. Each lane queue leading into the intersection is then processed, one at a time. Each car in the lane is moved through the lane as discussed in Section 2.2 and moved to a new lane if needed. If graphics is enabled, the cars old position is erased and the car is drawn in the new position.

```
TimedLights() Run timed lights simulation

     InjectVehicle() Load a vehicle into the simulator randomly
         NewVeh() Create a new car.
         getRandXHead() Select a start or destination intersection at random
         RouteSearch() (in IDAS.c) Find route for vehicle to destination
         printRoute() Print the route the vehicle will take.
         Veh2Lane() Inject vehicle into the correct lane.
         prtVehInfo() Print critical vehicle information.

while cars in simulation and maximum steps not exceeded:

     ProcessFirstX() Process first intersection and inject more cars.
         scrnAdj() Adjust graphics window or single step simulation
         setNewInjRate() Set New vehicle injection rate
         InjectVehicle() Inject a car into the simulation if directed.

CheckLights() Check Lights at each intersection - change if needed.

     CkLightLevel1() - Fixed Light Timings

     CkLightLevel2() - look at local sensors only to change lights

     CkLightLevel3() - look at local sensors and sensors 1 block away

     CkLightLevel4() - Total knowledge of car positions.

ProcessLanes() Move cars through lanes at each intersection.

     drawBlk() Draw outline of block when graphics is enabled.

     ProcessLn() Move cars in a specific lane.
         findRefPos() Find safe distance following reference point
         MoveCar() Move car through lane, adjust acceleration for stop/turning
             drawCar() Erase old car location when graphics is enabled
             getAcc() (in LIGHTUTL.c) Calc acceleration needed to reach reference point
             MoveCarThruX() Move car through intersection
               Veh2Lane() Move car to new lane
             drawCar() Draw car in new position when graphics is enabled
```

4.5 Traffic Light Simulator Simulation Structure - SARSA Learn LIGHTSIM.c

The *SARSA Learn* control strategy uses Reinforcement Learning (Thorpe 1997) to train a controller to minimize the number of time steps to move vehicles through the traffic environment. The *SARSA Learn* strategy is similar to the *Fixed Timing* control strategy except the lights are changed based on state-action values set during previous and current *SARSA Learn* control strategy training trials. See (Thorpe 1997) for a description of the training algorithm used for the SARSA style of Reinforcement Learning.

```
SARSA_Learn()

     init

         set eligibility traces to zero

while trials to run:
        determine I Cars (for i=0 to 40)
        determine J Cars (for j=0 to 40)
        determine NS and EW Light Color
        inject I Cars in NS and J cars in EW, set NS lights Red, Learn
        inject I Cars in NS and J cars in EW, set NS lights Grn, Learn
        inject J Cars in NS and I cars in EW, set NS lights Red, Learn
        inject J Cars in NS and I cars in EW, set NS lights Grn, Learn
    save_state_values()
    inject car
        calc average spacing left in lane
        inject car randomly from 2 feet to average feet in lane
        determine safe following speed
        set speed randomly from 0 to safe speed
    Learn
        set light color based on previously selected or initial action
        determine state
        set eligibility trace to one for initial state
        move cars thru lanes = ProcessLanes() (take action a)
        observe reward r = -1 and new state s'
        choose next action a'
        improve value
            calc Temporal Difference Error (TDerr), e
            set current eligiblity trace (accum or replace)
            for all s,a:
                increment state-action value
                update eligiblity trace:
                  if s' non-terminal = gam*lam*elig(s,a)
                  if s' terminal = 0
    ProcessLanes() Move cars through lanes at each intersection.
        ProcessLn() Move cars in a specific lane.
            findRefPos() Find safe distance following reference point
            MoveCar() Move car through lane, adjust acceleration for stop/turning
                drawCar() Erase old car location when graphics is enabled
                getAcc() (in LIGHTUTL.c) Calc acceleration needed to reach reference point
                MoveCarThruX() Move car through intersection
                  Veh2Lane() Move car to new lane
                drawCar() Draw car in new position when graphics is enabled
```

4.6 Traffic Light Simulator Simulation Structure - SARSA Run LIGHTSIM.c

The *SARSA Run* strategy is similar to the *Fixed Timing* control strategy except the lights are changed based on state-action values set during the *SARSA Learn* control strategy training trials.

```
    SARSA_Run()
        init_state_values()
        InjectVehicle() Load a vehicle into the simulator randomly
            NewVeh() Create a new car.
            getRandXHead() Select a start or destination intersection at random
            RouteSearch() IDAS.c Find route for vehicle to destination
            printRoute() Print the route the vehicle will take.
            Veh2Lane() Inject vehicle into the correct lane.
            prtVehInfo() Print critical vehicle information.
```

```
while cars in simulation and maximum steps not exceeded:
    ProcessFirstX() Process first intersection and inject more cars.
        scrnAdj() Adjust graphics window or single step simulation
        setNewInjRate() Set New vehicle injection rate
        InjectVehicle() Inject a car into the simulation if directed.
    CheckLights() Check Lights at each intersection - change if needed.
        determine state
        Choose Epsilon Greedy action for current state
    ProcessLanes() Move cars through lanes at each intersection.
        ProcessLn() Move cars in a specific lane.
          findRefPos() Find safe distance following reference point
          MoveCar() Move car through lane, adjust acceleration for stop/turning
            drawCar() Erase old car location when graphics is enabled
            getAcc() (in LIGHTUTL.c) Calc acceleration needed to reach reference point
            MoveCarThruX() Move car through intersection
            Veh2Lane() Move car to new lane
            drawCar() Draw car in new position when graphics is enabled
```

4.7 Traffic Light Simulator Environment Save LIGHTSAV.c

When the simulation has terminated, either because all vehicles have reached their destinations or the maximum number of time steps has elapsed, the final traffic environment is saved by the *saveenv()* function. The output of the final traffic environment was to be used to resume testing where the simulation ended. So far there has been no need to resume testing where the simulation ended and this function could be removed (it is not up to date).

```
saveenv() Save Traffic Environment

    SimParmsOut() Output Simulation Parms used

    DefaultOut() Output Defaults used

    XHeadOut() Output Intersections used

        LMnMxOut() Output Lane min max times
        LXtrlOut() Output Lane controller list

    LaneOut() Output Lanes Used

    VehicleOut() Output Vehicle Information (not implemented)

    TimeHistOut() Output History Information for Analysis (not implemented)
```

4.8 Traffic Light Simulator Report Module Structure LIGHTRPT.c

The results of the simulation are optionally written to report files for later analysis. The simulation results are usually produced for debugging and testing trials. the outputs of the simulator reports are shown in Figure 6 and Figure 7.

```
report() Report Simulation Times

    memRpt() List memory usage

    SetTime() Output Simulation Parms used

    incTime() Increment Time

    prtTime() Format and print time

    compareTimes() get relative time difference

Utility Functions in Lightrpt.c

    prtfVehInfo() print vehicle information

    prtTime() format time and print it

    prtElapsed() print elapsed time

    compareTimes() get relative time difference

    printElapsed() print elapsed time for simulation
```

4.9 Traffic Light Simulator Utility Routines LIGHTUTL.c

The "lightutl.c" file contains utility routines for debugging: *invoke()* and *devoke()*, determining vehicle actions: *decelDist()* and *getAccel()* and time manipulation functions.

```
Utility Functions in Lightutl.c

     compareTimes() Determine relative difference between two times.
     decelDist() Determine distance to begin deceleration to begin safe turn
     devoke() Trace Program Execution
     EqTime() Equate one time to another
     getAcc() Determine acceleration needed to achieve safe vehicle spacing
     getCurTime() Get current wall clock time.
     incTime() Increment Time a specified amount
     invoke() Trace Program Execution
     SetTime() Set time variable
     timeDif() Calc Difference between two times
```

4.10 Traffic Light Simulator IDA Star Routines

Route selection is performed using the Iterative Deepening A Star (IDA*) method.

```
IDAS.c

     RouteSearch() Find a route for a vehicle
         calcNumBlocks() calc number of blocks from beginning to end
         IDAS() The routine that does the finding
             GetNewRouteNode()
               MoveRouteNode()
               printRoute()
             GetSuccessors()
               GetNewRouteNode()
               MoveRouteNode()
               printRoute()
             ChkSuccessors()
               getLane()
               calcNumBlocks()
               MoveRouteNode()
               printRoute()
     printRoute() Print route found if desired
     cleanUpRoutes()
         MoveRouteNode()
         printRoute() Print route found if desired
         ConcatList()
```

Utility Functions in IDAS.c

```
calcNumBlocks() calculate birds eye distance between blocks (an under estimation of distance required
by IDA*)
freeUnusedRouteNodes() move freed route nodes on free list
getLane() get lane information
```

5. SUMMARY

The basic physics and data structures for a physically-realistic vehicle traffic flow simulator using fixed light cycle timing have been presented. Using one-second time intervals for the traffic light controller agent and the simulation of vehicle movement simplifies the simulation while providing realistic vehicle movement. The simulator has been used to model and compare traffic control strategies based on reinforcement learning (Thorpe 1997), simple heuristics, and fixed light timing.

REFERENCES

Haight, F. A., (1963), *Mathematical Theories of Traffic Flow*, Academic Press

Paal, F. F., (1975), "Third Order Simulation of Coherent Traffic Flow", *Annual Simulation Symposium, 1975 Simulation Conference*, Gordon and Breach Science Publishers, pp. 135-140.

Reitman, J., (1971), *Computer Simulation Applications, Discrete-Event Simulation for Syntesis and Analysis of Complex Systems*, Wiley Interscience Division of John Wiley & Sons, pp. 297-331.

Thorpe, Thomas L., (1993a), Personal observations of traffic lights at 104th and Wadsworth and 100th and Wadsworth in Westminster, CO.

Thorpe, Thomas L., (1993b), Timings of acceleration and deceleration of personally owned Honda Civic and Toyota 4Runner.

Thorpe, Thomas L., (1997), "Traffic Light Control Using SARSA with 4 State Representations", Masters Degree Project Report, Colorado State University, Fort Collins, CO 80523