ECM2433 – The C Family

Traffic Simulator Report

March 2022

1 Design Decisions and Assumptions

1.1 Deque

The core part of the traffic simulator implementation is to manage a list of waiting vehicles on both sides (right and left) of the road. Such a list would have an unknown size, as this would depend on the random vehicle arrival rate and traffic light period.

Hence; I decided to implement the deque as a data structure to store the waiting vehicle list. The deque would be implemented as a doubly-linked list. I chose using a deque over a regular queue, as cars would be both arriving (back of the deque) and passing the traffic light (front of the deque).

An individual element in the deque would also have an integer value of the iteration the vehicle joined. This would later be used to calculate the average and maximum wait times of the vehicle.

Deque is a pointer structure and hence requires dynamic memory allocation. To ensure there are no memory leaks, memory is freed both upon a node pop (vehicle passes the light) and when emptying the whole deque (at the end of each simulation). Deque is implemented as a header file (**deque.h)** and a source file (**deque.c)**.

1.2 Simulator Base

To carry out the simulation the below information should be available; current traffic light color, current vehicle queues, fixed arrival rates and fixed light periods. Similarly, after the simulation the following data should be available to print the result; average wait times, maximum wait times, clearance times, a total number of vehicles.

To simplify the storing (and passing through functions) process of the above data, I decided to create two structures; **FULL_SIM** and **FULL_RESULT**. As both would also be storing data for both sides of the road, I created substructures; **SIM** and **RESULT**, with full structures having pointers to them.

Additionally, I also decided to store the iteration when traffic lights changed as a variable (lightChangeTime) in **FULL_SIM**. This will be useful to calculate when set light periods (on either side of the road) have reached timeout.

$$Current\ Iteration - Light\ Change\ Time = Light\ Period$$

The above equality suggests the light period has been reached and traffic lights should change.

I also stored total wait times instead of average. They are much easier to store, and the average wait can be calculated on print:

$$Average\ Wait = \frac{Total\ Wait}{Vehicle\ Count}$$

Finally, similar to the deque implementation, I also created functions to free allocated memory. Here, **freeSim** would free the simulation structure and the vehicle deque it contains. While the function **freeResult** frees the result structure.

These structures are located in the **simulatorBase.h** header file. The corresponding source file (**simulatorBase.c**) contains functions for set up, initialization (memory allocation), and print.

1.3 Running Simulations

The simulation can be seen as a 3 level process, I decided to define by 3 functions; **runIteration**, **runSimulation**, **runSimulations**. An individual iteration of the simulation (that is a single time frame) is run by **runIteration.** The function **runSimulation** runs an individual simulation in a number of iterations. Likewise, **runSimulations** runs a number of simulations and outputs the average result.

For clarity, I defined the fixed iteration and run count (amount of runs to average based on) as macros in the header file.

The above 3 functions make up the core of **runSimulations.c**. However, the source file also contains the main function (with user input handling) and several side functions utilized by **runIteration**. The source file also makes use result and simulation structures of the simulator base (**simulatorBase.h**).

1.4 Time in Simulation

To clarify, I considered an iteration as a time unit in this simulator. For example, an average wait time of 13 suggests the average vehicle waited for 13 iterations before passing through lights. Measuring the real-time (e.g. milliseconds) would not be useful, as we would be measuring the program's performance, not the optimality of traffic control at given parameters.

1.5 Randomness

Randomness plays an important role in the simulator, determining the vehicle arrival (based on a predetermined rate). I accommodate this I implemented a function returning a biased random boolean in reference to the given arrival rate. For randomness itself, I used the default library function **rand()**.

While more advanced libraries such as GSL (GNU Scientific Library) would have provided less predictable (and more random) results. I believe in the case of this simulator the **rand()** function is adequate. Given that the role of randomness is to simply manage a biased flow of arriving vehicles, I assumed more advanced random generators won't make a valuable difference.

2 Traffic Simulator Experiment

2.1 Defining the Experiment

As an experiment, I decided to compare the effects of the traffic light period on traffic management. To simplify the experiment, I used 3 traffic scenarios:

1. *Light Traffic:* with a traffic arrival rate of 0.1 (that is a 10% chance a vehicle arrives on any iteration)
2. *Medium Traffic:* with a traffic arrival rate of 0.5
3. *Congested Traffic:* defined by a traffic arrival rate of 0.9

For all scenarios, I will run simulations (of 100 run average) in varying traffic light periods. The idea behind the experiment is to demonstrate the difference with varying traffic light periods (and not the difference on two sides of the road), so the results of both sides will be averaged as one (e.g. average wait = average waiting time on both sides).

Such an experiment will demonstrate what traffic light period is best in a given scenario. As a result, for the given road sketch, it should be possible to adjust the traffic lights at different levels of congestion to optimize the travel through the road.

2.2 Results

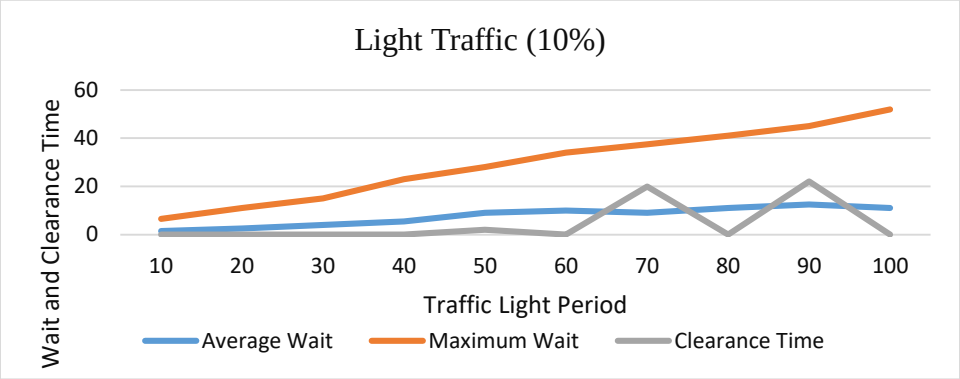

Figure 1 - Light Traffic (10%)

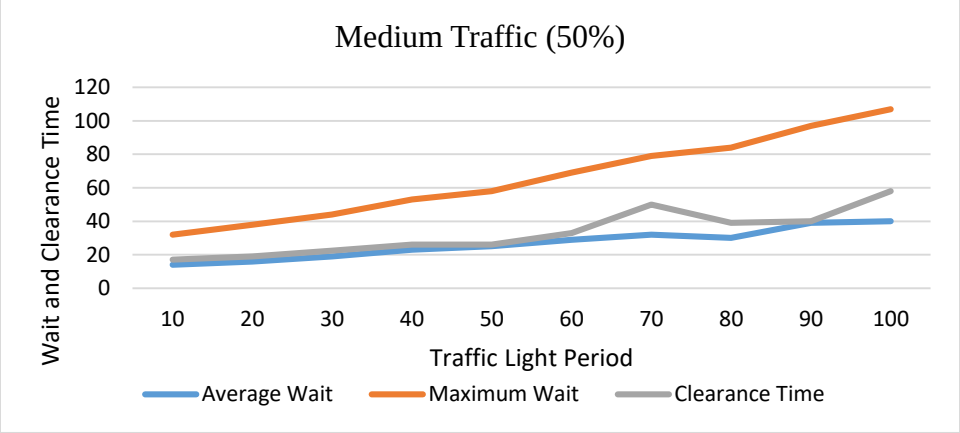

Figure 2 - Medium Traffic (50%)

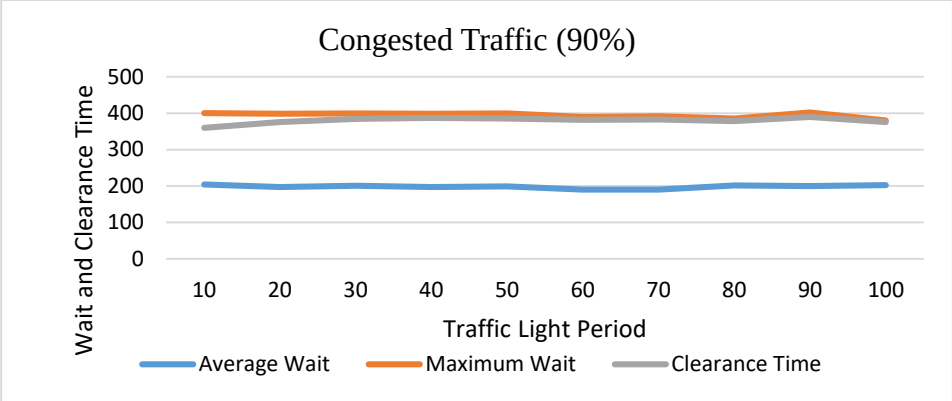

Figure 3 - Congested Traffic (90%)

2.3 Interpreting the Results

In light and medium traffic, we can see a tendency of the increased light period leading to an increase in wait and clearance time (especially the maximum waiting time). Whereas in highly congested traffic the light period does not show a significant difference. With the exception being the maximum wait having slightly lower values in longer traffic light periods.

Overall, it is clear that a short light period is to be preferred for light and medium traffic, while for highly congested traffic longer wait times can lead to marginally better traffic management.

3 Example Output

An example output can be obtained by running the runSimulations executable with 4 parameters (respectively; left arrival rate, left light period, right arrival rate, right light period):

```
./runSimulations 0.2 10 0.65 10
```

Which would print the following output:

```
Parameter values:
        from left:
                traffic arrival rate: 0.20
                traffic light period:   10
        from right:
                traffic arrival rate: 0.65
                traffic light period:   10
Results (averaged over 100 runs):
        from left:
                number of vehicles:     91
                average waiting time:    3
                maximum waiting time:   10
                clearance time:          0
        from right:
                number of vehicles:    293
                average waiting time:   77
                maximum waiting time:  156
                clearance time:        137
```