The C Family

Alfie Walliss

March 26, 2022

1 Design

For the design of the simulation the program was split into the four main sub-categories as followed:

1.1 The Queue

The queue was implemented as a linked list; each node was represented as a C structure with two attributes. A 'wait' attribute with the type integer and a recursive 'next' attribute pointing to the next node in the list. The 'wait' attribute is used to hold the value associated with the number of iterations a car has currently been waiting in the queue for. The newNode() function creates and returns a new node with the next attribute set to NULL and the wait attribute set to whatever number is passed to the function. I decided to implement the queue with this parameter so that the queue can be used in more scenarios and built upon rather than being program specific. The queue also uses an enqueue() function which first checks if the queue is empty and if so returns a new node. If the queue is not empty, the function will loop through the queue and append the node to the end of the queue. There is also a dequeue() function which removes the node from the front of the queue and returns it. Furthermore, there is an is isEmpty() function which checks whether a queue is empty returning a Boolean value dependant on the result. Finally, I also produced a addOne() function which adds one to all of the wait times of all nodes within a queue. This allows the runOneSimulation() function to increment the wait times of all cars in a queue per iteration. I decided to implement the queues as a linked list as there is no way of knowing how many cars are going to be in the queue. I could have implemented the queue using arrays however this would have been very inefficient as the arrays would have had to been declared with size 500 to ensure they don't run out of space. With certain parameter values, this would be very inefficent and the queue would never be near full.

1.2 Result

As C does not let you return multiple values from a function, I was required to come up with a different way to return all the results from the runOneSimulation function. To do this I decided to create a result structure with the following attributes for both the left and right queues: passed, average, max_time and clear. These values were all required to be returned as per the specification.

1.3 runOneSimulation

The runOneSimulation() function takes in the four parameters as described in the specification. One assumption I made when creating this function was that both traffic lights have an equal chance of starting green. This concept is implemented in the beginning of the function. I then designed the program to start a while loop which will only terminate when both queues are empty. Per each iteration the program will check if 500 iterations has occurred; if they haven't, using random numbers, the program will potentially add a vehicle to the end of the queue either side dependant on the arrival rate. A car from the side with the green light will be allowed to pass through. If 500 iterations has been reached, the function will no longer allow cars to join the back of the queue and the program will continue till both queues are empty. Another option for an iteration is that the traffic lights change in which case the vehicles do not leave the queues from either side. These three options have been implemented through if-statements. At the end of each iteration the addOne() function previously mentioned is always called to add one to the wait times of all vehicles.

1.4 runSimulation

At the beginning of the runSimulation program I decided to initialise all the variables required for random number generation. I decided to use the GSL library to generate uniform random numbers; this is because these values are more 'random' than the random numbers generated by C's built in function. This will then allow for a more balanced and accurate experiment later on in the report. After this I decided to clear the screen as I thought this allowed for a better user experience. The next part of the runSimulation program is responsible for accepting user inputs and also validating them ensuring they are of correct types. This is done through the use of 4 while loops all responsible for one of the user inputs; the while loops will not pass until the user inputs a valid value. Following this is a loop which will run the runOneSimulation() function 100 times as outlined by the specification. The values returned by the runOneSimulation() function will be added to counters which will then be divided by 100 before being outputted by the program to show the average results over the execution.

These are all the main design choices I made whilst creating the project.

2 Experiment

2.1 The Experiment

For the experiment I have decided to analyse the effect of the the left traffic arrival rate on both the average left and right clearance times. The control variables will be: the right traffic arrival rate, the left traffic light period and the right traffic light period. I will set both the traffic light periods to 10 and the right traffic arrival rate to 0.5. I am then going to run the program 5 times with the traffic arrival rates of: 0.2, 0.4, 0.6, 0.8 and 1.0. As the runSimulation part of the program already repeats runOneSimulation() 100 times there is no requirement to repeat the experiment as it is clear that the results

produced will be accurate with minimal effect from anomalies.

2.2 The Results

Below are the results from the experiment:

Left Traffic Arrival Rate	0.2	0.4	0.6	0.8	1.0
Right Average Clearance Time	19.29	19.88	18.63	17.83	20.50
Left Average Clearance Time	2.11	4.20	101.56	301.57	498.95

Figure 1: Table showing the effect of changing the left traffic rate on both the left and right clearance times.

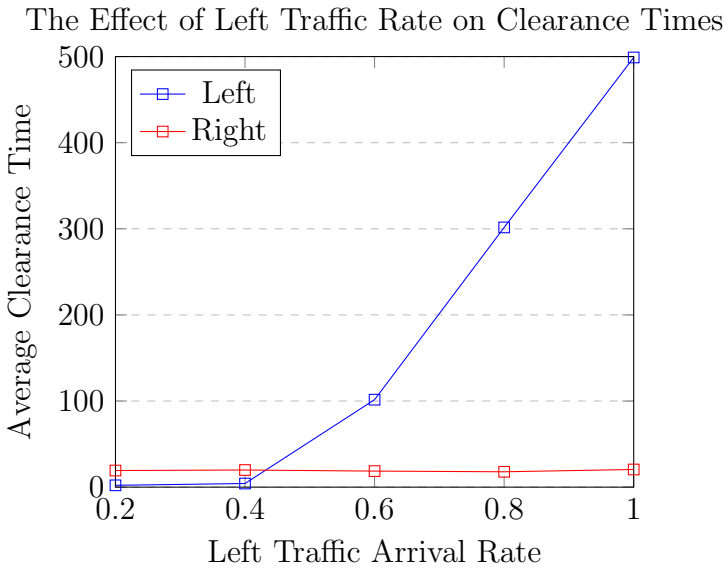

Figure 2: Graph showing the effect of changing the left traffic rate on both the left and right clearance times.

2.3 Analysis

From the results above it is clear that the left traffic arrival rate has no effect on the right average clearance time. The right average clearance time remains relatively consistent for all arrival rates. The reason for this is that the left traffic arrival rate only effects the number of cars that are arriving on the left; the traffic light period remains the same and therefore the number of cars that are on the right after the first 500 iterations of the simulation remains unaffected. As a result of this, the clearance time also remains similar throughout the experiment.

Figure 2 clearly highlights that the left traffic arrival rate has a monumental effect on the average clearance time for the left side. When the arrival rate is low at 0.2 and 0.4, the average clearance time for the left side remains low at 2.11 and 4.20. This is because the arrival rate is below 0.5 and, as each side of the simulation is allocated an equal time slot for the lights to be green, more cars are passing through the system than are arriving.

The values are not zero as cars will still be able to arrive on the 499^{th} iteration even if the green light is on the right side. Therefore, there may be a few cars on the left side after the original 500 iterations still needed to be cleared. When the arrival rate is increased further, the left average clearance time dramatically increases. This is because more cars will now be arriving than that of cars passing through the system on a given side. For example, over 22 iterations of the simulation the lights change twice and each side of the traffic lights allow 10 cars to pass through. When the traffic arrival rate is 0.6, on average, 13 cars will arrive however only 10 will be allowed to pass through. Out of these 13 cars, only 6 of them will arrive when the light for the current side is green and 7 will arrive when the light is not. These cars then has a cascading impact as when the light becomes green again there will already be 7 in the queue and therefore a backlog will be created when the 10 iterations are complete. It is clear from this that when the left traffic arrival rate is greater than 0.6, the backlog formed during each cycle of the simulation will be much greater resulting in a larger queue of cars after the 500 original iterations consequently increasing the average clearance time.

3 Demonstration

```
Enter the traffic light period for the left:
10
Enter the traffic light period for the right:
10
Enter the traffic arrival rate for the left:
0.4
Enter the traffic arrival rate for the right:
0.5

Parameter values:
   from left:
      traffic arrival rate : 0.40
      traffic light period :  10
   from right:
      traffic arrival rate : 0.50
      traffic light period :  10
Results (averaged over 100 runs):
   from left:
      number of vehicles: 182.47
      average waiting time: 5.55
      maximum waiting time: 16.86
      clearance time: 4.20
   from right:
      number of vehicles: 228.08
      average waiting time: 15.22
      maximum waiting time: 34.51
      clearance time: 19.88
```

Figure 3: Screenshot of sample input and output from the program used in the experiment.

4 Problems

There are no known errors or problems with the program.