Report – Post Office Queuing Simulation

Design Decisions

Assumptions

The unit of time will be minutes, as this is the most appropriate time to measure how long customers would typically spend in a Post Office branch. The alternatives of seconds and hours would result in numbers which are far too large and far too small, respectively.

When leaving the queue to join a service point, only the task time from the customer will be handed over to the service point. This is because the task time is the only relevant data; tolerance of the customer is irrelevant because the customer has left the queue, and the next customer in the queue is implicit.

For each minute, I have assumed that only a single customer can leave the queue to join a service point, even if there are multiple people in the queue and multiple service points are available. This is because in real life, it would take some time for the person at the front of the queue to realise that there is an available service point and move towards it.

GNU Standards

In my C programs, I have adopted the GNU Standards (https://www.gnu.org/prep/standards/html_node/Writing-C.html), as this will help standardise my code for future development.

For example, this includes the convention of keeping the lengths of source code to 79 characters or less, as this ensures maximum readability for the widest range of environments. It also includes the convention of commenting your code briefly to explain its purpose, which will help other programmers to understand parts of your code.

Input Parameters

In addition to the essential input parameters (maximum queue length, number of service points, and closing time), I have decided to allow the user to configure other parameters which will be used in tandem with the implementation of random distributions.

The user can configure the average number of customers per minute, mean and standard deviation of minutes for a customer's task, and mean and standard deviation for maximum queue time of a customer. This enables them to configure the simulation to reflect the demand for their Post Office branch more effectively; some branches may have less demand due to being in a more rural area, whilst others may be in the middle of a busy city, and so have more demand.

Choice of Random Distributions

I have included the GSL library to implement randomness for the number of new customers arriving at the branch for each time interval, the minutes required to complete the task of each customer, and the maximum time a customer is willing to queue for.

For the number of new customers arriving for each time interval, I decided to implement the Poisson distribution. This distribution is used to show how many times an event is likely to occur within a specified period. It takes the expected number of events in each interval as the parameter. For example, if the user expects two customers to arrive for every minute, then they would set the parameter *averageCustomersPerMinute* to have the value of 2.

On the other hand, I decided to implement the normal distribution (also known as the Gaussian distribution) for the time required to complete the task of each customer, and the maximum time a customer is willing to queue for. This is because the normal distribution approximates many natural phenomena, with everyday data often fitting the bell curve shape of this distribution. The *central limit theorem* states that for many situations, when independent random variables are added, their normalised sum tends towards a normal distribution as the sample size becomes larger, even if their original variables are not normally distributed.

Customers are likely to have tasks which take similar lengths of time, which would form the middle of the bell curve, and a small number of customers will have tasks which take an extremely long or short amount of time, which would be represented by the ends of the bell curve. The same would apply for the tolerance of a customer; most will have a similar tolerance for queue time, but there will be a small minority who are either easily frustrated and have a very low tolerance, or very patient and have a very high tolerance.

The normal distribution takes the mean and standard deviation as parameters. These can be configured in *inputParameters.txt* for the task completion time and tolerance of customers. For example, if the user expects customers to have an average task completion of five with a standard deviation of two, then they would set *meanMinsPerCustomerTask* to 5, and *standardDeviationMinsPerCustomerTask* to 2.

Project Structure

My program performed the queuing system simulation through the implementation of a queue as a data structure, using linked lists. A queue is represented by a linked list, which has attributes of *queue_length* and *max_queue_length*. In the linked list, there are nodes, which have front and rear pointers to other nodes, or *NULL* if they are at the front or back of the queue.

Customers join the queue as nodes, which have attributes of *mins* (for the time their task will take to complete), *time_waited* (for how long they have waited in the queue for a service point), and *tolerance* (for how long they are willing to wait in the queue before leaving early).

Service points are managed by an array of integers, *service_points*. This implementation is possible because the number of service points is specified in the input parameters file. It has an advantage over other data structures, as the time complexity for accessing it in the worst case is O(1), compared to O(n) for many other data structures, such as linked lists, stacks, and queues. Meanwhile, it remains level in space complexity in the worst case, at O(n), which is the same as the data structures as aforementioned.

At the start of the program, the array of service points is created. Then, the simulations start to run. At the start of each simulation, the queue is created. Time intervals are represented by time slices, each representing a minute passing.

During each time slice, regardless of whether the branch is past closing time, if the branch is not empty (meaning there are either customers in the queue, or in a service point), a series of processes are performed. Customers already in the service points get served, meaning their task time gets reduced by one. This may result in a customer who has finished being served, and leave the system. If there is a customer at the front of the queue and a service point is free, it will fulfil them, and other customers in the queue will move up by one. All customers in the queue have their time waited incremented, and customers who have passed their tolerance for waiting will be timed out.

Then, the program checks if there are new customers who want to join the queue if the branch is not past closing time. Their task time and tolerance are randomly generated using the normal distribution, while the number of new customers is generated by the Poisson distribution. Each new customer either joins the queue if there is space, or leaves unfulfilled otherwise.

Problems Encountered

I have attempted to free pointers where necessary to prevent memory leaks. However, using the *valgrind* utility has identified there are cases where my program will encounter memory leaks. These will not occur all the time; a leak occurs approximately 50% of the time. Memory leaks are more likely to occur for longer simulations. The origin can be traced back to the *customer* variable in the create*_new_customer* function of *customer.c*, which is called from the *enqueue* function in *queue.c*, which is called from the *main* function in *simQ.c*. I have attempted to fix this with *free(customer)* at the end of the *dequeue* function (when the customer goes out of scope), but the issue persists.

Simulator Experiment

For my experiment, I investigated how to optimise the fulfilment rate. This involved maximising how many customers, on average, can be served in one day, whilst at the same time minimising how many customers are either unfulfilled or timed out. To do this, I ran a variety of simulations with different parameters for maximum queue length and number of service points, with the goal of determining the point of diminishing returns, and hence the optimal parameters.

I assumed that a typical Post Office branch opens at 8am and closes at 5pm. This means that it remains open for nine hours, which gave us the fixed value of 9 * 60 = 540 minutes for the closing time parameter. I also assumed that this would be a busy branch, located in the middle of London, so it would average 0.5 customers per minute. Together with these assumptions, I made a rough estimate of realistic mean values and standard deviations for task length and customer tolerance for queuing, resulting in the following controlled variables:

Closing Time: 540
Average Customers Per Interval: 0.5
Mean of Task Length: 5.0
Standard Deviation of Task Length: 2.0
Mean of Customer Tolerance: 5.0
Standard Deviation of Customer Tolerance: 2.0

Example Output from the Simulation

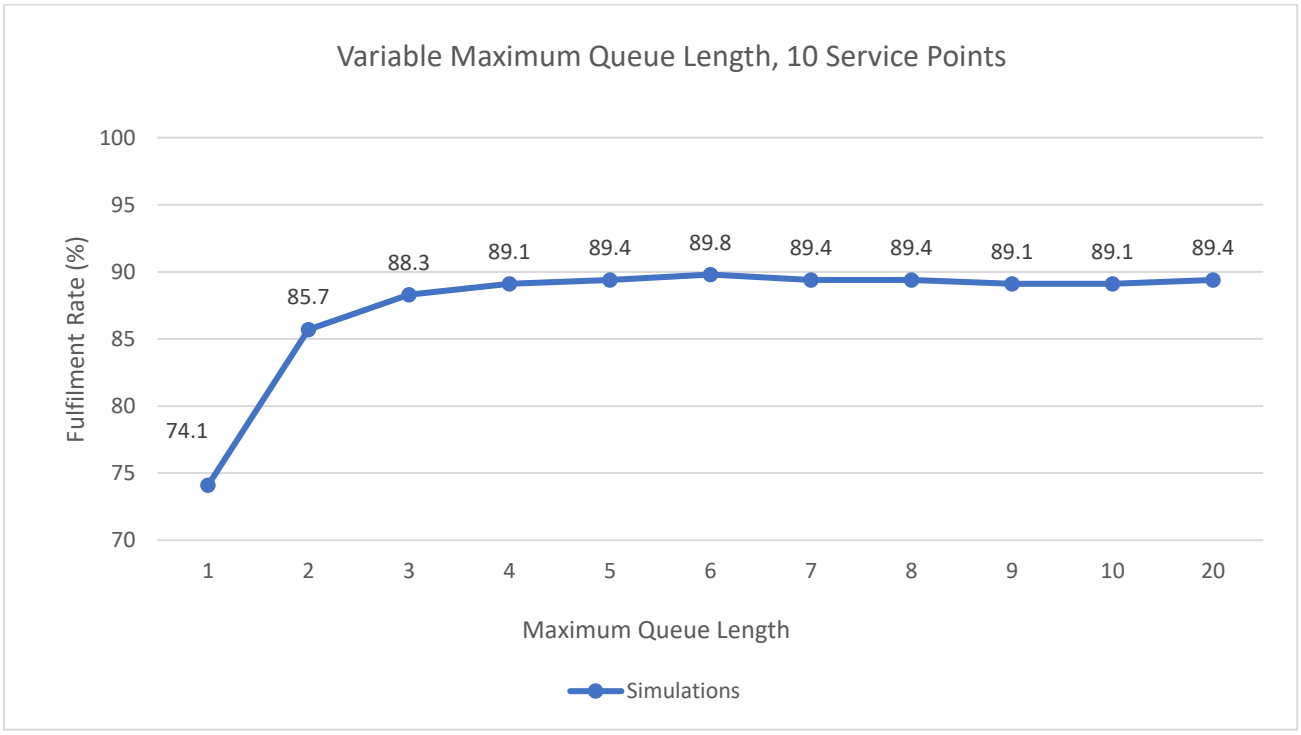
```
Parameters Read From Input File:
   Max Queue Length: 2
   Number of Service Points: 2
   Closing Time: 540
   Average Customers Per Interval: 0.500000
   Mean of Task Length: 5.000000
   Standard Deviation of Task Length: 2.000000
   Mean of Customer Tolerance: 5.000000
   Standard Deviation of Customer Tolerance: 2.000000

Average Number of Customers Fulfilled: 186.087997
Average Number of Customers Unfulfilled: 49.313999
Average Number of Customers Timed Out: 30.879000
Average Waiting Time of Fulfilled Customers: 1.130992
Average Time After Closing to Finish Serving Remaining Customers: 5.594000
```

The example output shows that around 186 customers would be fulfilled per day on average, with around 49 unfulfilled per day, and 31 timed out per day. This results in a calculated fulfilment percentage of approximately 70.0%, using the formula of *(fulfilled customers / total number of customers) * 100*.

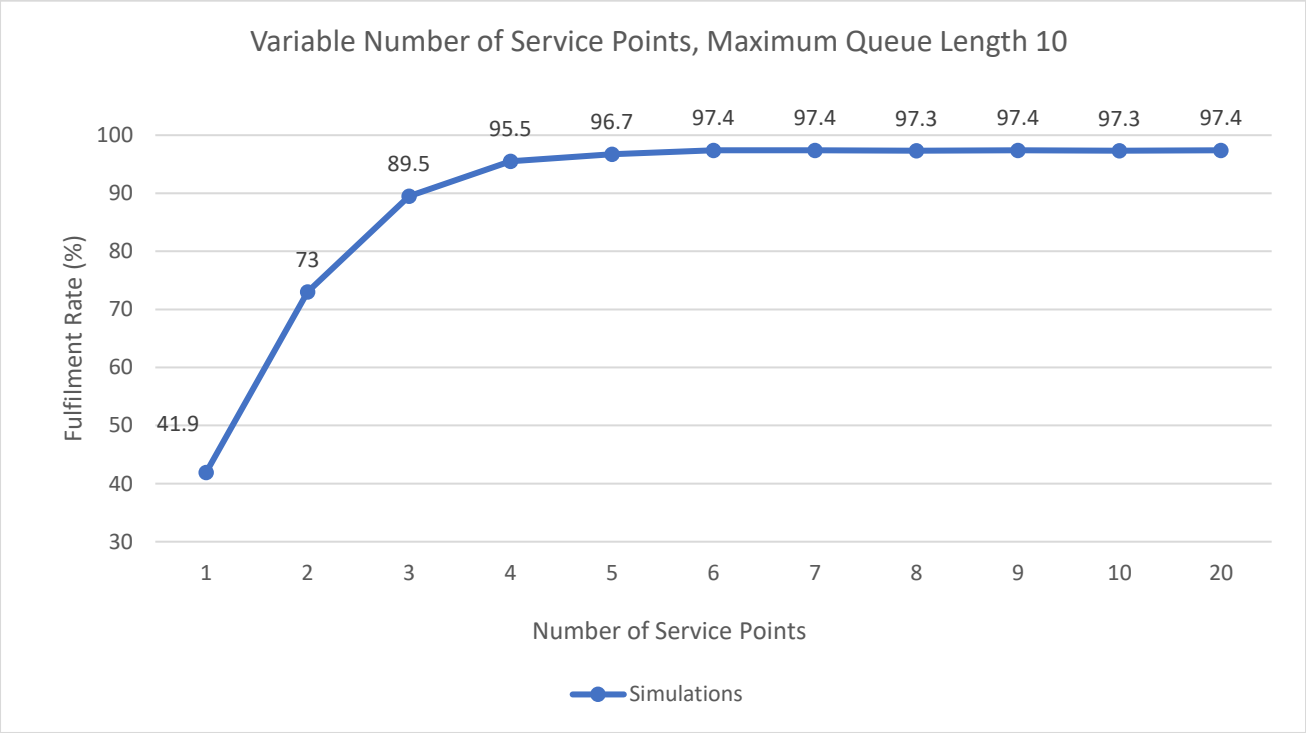
To optimise this number, I tested different configurations of maximum queue length and number of service points to identify a point of diminishing returns. This point would be the optimum values for these two parameters. Simulations were performed with a large sample size of 1000 simulations per configuration, with each simulation representing a working day.

Results

I started by varying the maximum queue length, and keeping the number of service points constant at 3. The results for this part of the experiment, with 1000 simulations per configuration, are shown in the graph below:



Then, I varied the number of service points, and kept the maximum queue length constant at 10. This part of the experiment was also performed with 1000 simulations per configuration, with the results shown in the graph below:



Evaluation

Both parameters demonstrated similar behaviour, as evident in the graphs. With low values, they become the bottleneck for the system, resulting in low fulfilment rates, which increase at a very high rate initially. This rate of increase quickly decreases. With the varying maximum queue length, this drop-off occurred much more quickly, after only a value of 2, whereas for the varying number of service points, this started to occur at a value of 4.

The mechanism for this is clear. A very low maximum queue length makes it more likely that the queue becomes full at any given time, and hence new customers are unfulfilled. Meanwhile, a very low number of service points makes it more likely for a customer to spend extended amounts of time in the queue, and hence more customers become bored of waiting in the queue, and leave early as timed-out customers.

However, this mechanism also explains the difference between these two parameters. Customers in a branch with a low number of service points have more of a chance of being fulfilled, as the branch may get lucky and receive a customer who has a high tolerance for waiting in the queue. This luck factor does not exist for branches with a low maximum queue length, as customers are ruled out of being fulfilled at an earlier stage; they do not even have the chance to join the queue in the first place.

Analysing the graphs reveals that the optimal value for the maximum queue length is 5. This is because the fulfilment rate ceases to show any improvement by increasing the maximum queue length further; all subsequent rates were within the margin of error, at around 89.4%. A similar case is found for the number of service points, where the optimal value is 6, as the fulfilment rate remained roughly the same at around 97.4%, despite increases in the number of service points.

Conclusion

Both the maximum queue length and number of service points act as significant bottlenecks to the queuing system for a Post Office branch. Despite this, these parameters diminish in significance after they are increased only slightly, with fulfilment rates quickly converging to a point of equilibrium, also known as the point of diminishing returns. Ergo, to maximise the fulfilment rates, and therefore profit, smaller Post Office branches should focus on expanding their capacity, so that queue length and number of service points do not limit their potential sales.