Computer Science Large Practical report

Isabella Chan (s1330027)

December 21, 2016

1 Introduction

This report describes the Computer Science Large Practical 2016 project, a command-line application developed to execute stochastic simulations of the bin collection process in a "smart" city. The simulation allow user to input parameters to model a setting and simulate bin disposal and collection events. This section will give a brief outline of discrete event simulation and some descriptions for the bin collection problem domain. Section 2 will discuss the code architecture, how the events are modelled in the code and some functionality of the program including experimentation and statistics collection. Section 3 will discuss the experiments carried out to study the input parameters on the performance of the simulator, with attempts to optimise results and obtain the 'best' parameter for various topology. Section 4 will describe the design choices made in implementing the system including various route planning algorithms and the reason behind choosing Java. Section 5 summarises testing efforts and explains which aspects have been tested and the expected results.

1.1 Discrete stochastic simulations

In a discrete-event simulation, a system is modelled in terms of its state at each point in time. This is appropriate for systems where changes occur only at discrete points in time. This is thus appropriate for our bin collection problem.

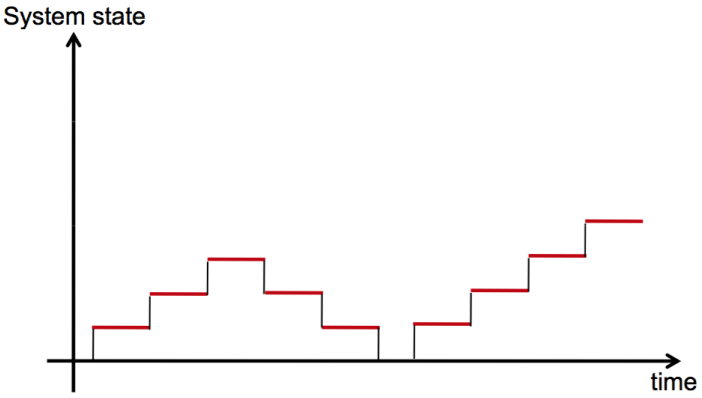


Figure 1: Discrete-event simulation: the system state remains the same for a certain period of time until the next state occurs.[3]

A stochastic simulation is a simulation that traces the evolution of variables that can change stochastically with certain probabilities. The underlying simulation algorithm is as follows:

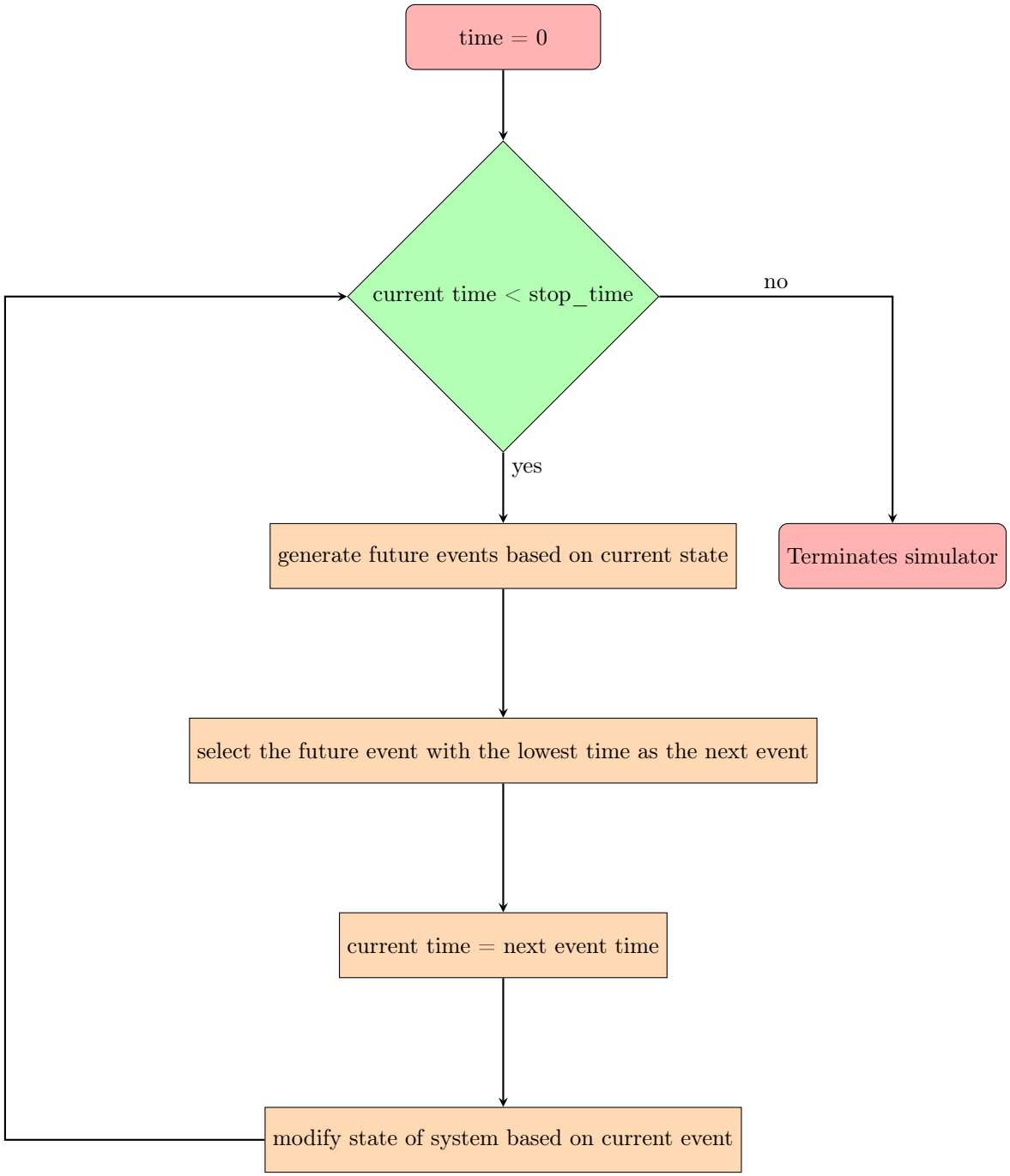


Figure 2: Discrete-event simulation algorithm

1.2 Problem Domain

The problem domain is to simulate bin collection in a city. A city is divided into multiple service areas, each with their own lorry, bins and bin collection schedules. Users can dispose trash bags into the bins, and the bins have sensors to detect their current content volume and weight. Each service area has a certain threshold value for the occupancy of bin. Trash content exceeding that threshold value will mean that the bin requires servicing. At the start of each bin collection schedule, an

optimal route will be computed for the lorry to travel around the service area and emptying bins. This simulation collects statistics from the trips made by the lorry as well as the trash it collected. These statistics will provide insights into how efficiency how collection service is with a certain parameters, and the relationship between certain settings of the system (i.e. number of bins) with those parameters.

2 Code Architecture

2.1 Entities

The parsing of input file is done in the `Parser` class. All input parameters are inputted in a text file by the user, whom should follow a straight specification, although the program checks for unrealistic input parameters. There are three variables that can be adjusted to optimise bin collection efficiency and to provide insight into area of improvements of bin service scheduling. They are

- Disposal distribution rate
- Disposal distribution shape
- Service frequency

In our *stochastic* simulation, bin bags are disposed in each bin at time intervals sampled from an Erlang distribution, determined by the disposal distribution rate and shape. Each bin bag has the same volume, and a weight bounded by a lower and upper limit. Service frequency, either unique for each service area or the same for all service areas, determines the number of bin collection schedules per hour in each service area. These parameters determine the system state by changing the values of various variables over the simulation. Entities and their variables that could change the system state are described in the following table

Entities and their variables	
Entity	Description
Bag	Models a bin bag being disposed according to an erlang-k distribution. Its weight is random and is bounded by some upper and lower limits, where as its volume remains constant throughout the simulation.
Bin	Trash weight and volume can change over the simulation, determines states such as lorry's actions or route planning.
Lorry	Lorry's location, weight and volume of collected trash can change. This class collects statistics for bin collection efficiency.
ServiceArea and ServiceAreaInfo	ServiceArea extends ServiceAreaInfo which contains information about a specific service area. This class models the service area of different service frequency, threshold values for bin collection and the number of bins it contains. A ServiceArea instance can compute paths for a lorry to collect bins and collect statistical information from the lorry. It also provides methods to rescheduling.

2.2 Events

Following the algorithm of a discrete stochastic simulation, the list of possible events include

1. a rubbish bag was disposed of a bin
2. the occupancy threshold of a bin was exceeded
3. a bin overflowed
4. a bin was emptied
5. a lorry was emptied
6. a lorry arrived at a location (bin or depot)
7. a lorry departed from a location (bin or depot)

A list of scheduled events in kept sorted in a queue. An event with a lower (nearest future) scheduled time has a higher priority. The event queue in implemented in the `Simulator` class in java PriorityQueue. An outline of the event queue and its methods is shown below.

```
public class Simulator {
  private PriorityQueue<AbstractEvent> events;
  public void insert(AbstractEvent e);
  public void doAllEvents();
```

The `insert` method allows new AbstractEvent instances to be inserted into the `events` queue and still keep the queue sorted with the nearest future event as the event with highest priority. The `doAllEvent` method extract (and remove) the next event from the event queue and the system state, including current simulation time, changes according to that event.

Events are implemented in an abstract class `AbstractEvent`. An outline of this class is shown below.

```
public abstract class AbstractEvent implements Comparable {
  private static float stopTime;     // maximum simulation time
  private int eventTime;       // simulation of this AbstractEvent instance
  public void schedule(int eventTime);  // sets eventTime
  public abstract void execute(Simulator simulator);
  public int compareTo(Object anotherEvent);
```

If a event is scheduled at a time greater than `stopTime`, it will not be inserted into the event queue. The attribute `eventTime` is the time during which this event should occur in the simulation. The method `execute` is called in `Simulator.doAllEvents()` each time an event is extracted from the event queue. Six classes extends this `AbstractEvent` class and they are described in the following table.

Events	
Class	Event it corresponds to
BinEmptiedEvent	Scheduled at a time during which a bin has been emptied. Updates bin and lorry content. Sets a LorryDepartureEvent to go to next destination according to current system state.
BinServiceEvent	Scheduled initially at fixed time interval. Computes path to travel around the service area to collect specific bins. This event finishes when all required bins have been serviced. If this event takes longer than the time interval, the next BinServiceEvent for that service area will be delayed until this event finishes.

DisposalEvent	Each bin has a reoccurring disposal event with time intervals generated from an Erlang-k distribution. When executed, a new bin bag will be disposed of in the bin. DisposalEvent changes the content of a bin until it is overflowed or being serviced during a schedule.
LorryArrivalEvent	Scheduled at a time during which a lorry arrives at the desired destination. Determines the lorry's action according to its location: if at depot, empty lorry and reschedule event if require; if at a bin, empty bin.
LorryDepartureEvent	Scheduled at a time during which a lorry departs from the current location. Generates a subsequent LorryArrivalEvent with a desired destination according to current system state. The lorry will depart to the depot if either the lorry has reached it's capacity or if there is no bins left to be serviced. Otherwise, it will go to the next bin that requires servicing.
LorryEmptiedEvent	Scheduled at a time during which a lorry has fully empty its content at the depot. The lorry's next action will be to reschedule if there are still bins left to be serviced or wait at depot otherwise.

Each `BinServiceEvent` instance has its own service queue to store the route for visiting bins. A BinServiceEvent only inserts the next BinServiceEvent into queue if it has been fully carried out by the lorry (serviced all bins scheduled at the beginning of the event). This occurs when a lorry returns to the depot, with no bins left in the service queue. The `LorryEmptiedEvent` checks this and if there is no bin left, it will call the BinServiceEvent instance's `update()` method to pass on statistics collected from the trip. The next BinServiceEvent will be inserted into the event queue if the scheduled time is less than `stopTime`. If there are still bins left, the `LorryEmptiedEvent` will call the BinServiceEvent instance's `reschedule()` method to carry out route planning for the remaining bins.

2.3 Algorithms and Mathematics

The following table describes the class which involves calculation and algorithms.

Algorithms and mathematics	
Class	Description
Random	Includes static method for calculating erlang-k.
Nearest-Neighbour	Contains method to produce a service queue for a particular BinServiceEvent instance using the nearest neighbour algorithm to compute a path to visit all bins that require servicing.
Brute-Force	Implement an algorithm to compute the optimal graph traversal path in route planning and provide a service queue for a specific BinServiceEvent instance.
ServiceArea.java and ServiceAreaInfo.java	ServiceArea extends ServiceAreaInfo which contains information about a specific service area. This class models the service area of different service frequency, threshold values for bin collection and the number of bins it contains. A ServiceArea instance can compute paths for a lorry to collect bins and collect statistical information from the lorry. It also provides methods to rescheduling.

2.4 Experimentation

An `experiment` keyword could be used with the three parameter: `DisposalDistrRate`, `DisposalDistrShape` and `serviceFreq`. Multiple parameters can be inputted for these variables and the simulator will run all simulations for all combinations and output summary statistics for each experiment.

2.5 Summary Statistics

Summary statistics are printed at the end of the simulation. This includes: average trip duration, number of trips per schedule, trip efficiency, average volume collected, and percentage of overflowed bins. An example of experiment summary statistics generated is shown below.

```
Experiment #1: disposalDistrRate 2.0 disposalDistrShape 3 serviceFreq 0.03
---
area 0: average trip duration 00:04:35:45
overall average trip duration 00:04:35:45
area 0: average no. trips 1.000
overall average no. trips 1.000
area 0: trip efficiency 13.029
```

```
overall trip efficiency 13.029
area 0: average volume collected 12.852
overall average volume collected 12.852
area 0: percentage of bins overflowed 0.000
overall percentage of bins overflowed 0.000
---
```

3 Experiments

The simulator supports experimentation of different combinations of the three variables at once. Output will be disabled and only summary statistics will be printed to the console and saved as a text file (in the folder `/cslp/bin/output_files/`). The aim of this section is to study the effects of parameters on the practicability of the system, and thus, in real-life bin collection scheduling.

While performing experiment, the simulator is initialised with reasonable variables - variables such that it could model real-life situation. Here are the input parameters I decided to remain constant throughout the experiments:

```
stopTime 300
warmUpTime 16.0

lorryVolume 24
lorryMaxLoad 8500
binServiceTime 300
binVolume 3.0
bagVolume 0.04
bagWeightMin 3
bagWeightMax 10
```

It would be reasonable to assume each service area to have no more than 20-30 bins during to the lorry's capacity. Although this could be affected by other parameters such as threshold values and service frequencies. The number of bins experimented with are 10, 25 and 50. Various service area specifications were experimented with:

3.1 roadsLayout matrix

The algorithms implemented in the code will always work if and only the input `roadsLayout` matrix does not contain any dead ends. with the maximum number of -1's in the matrix and still be able to perform a cycle which starts and ends at depot, and traverse all other locations, the `roadsLayout` matrix should have the elements next to the diagonal elements equal to -1.

Three different service area sizes are used with the number of bins = 10, 25 and 50. The

distances between locations, if a path exists, is between 10 and 20 minutes.This is a reasonable assumption since stopping too often is not cost efficient for other factors such as fuel cost in real life. To give an idea of distances between bins, see the following `roadsLayout` matrix. The proportion of -1's to positive values are preserved for different matrix sizes.

0	7	6	-1	5	5	5	-1	8	-1	7
-1	0	9	9	5	6	8	8	5	5	8
9	7	0	6	6	6	5	7	6	6	5
6	-1	-1	0	7	6	-1	-1	5	5	7
8	9	6	7	0	5	9	-1	7	7	7
-1	6	-1	-1	-1	0	7	9	7	6	7
8	-1	8	7	9	9	0	-1	-1	-1	-1
5	5	6	5	5	5	5	0	6	5	7
-1	7	5	5	8	9	6	-1	0	5	7
-1	8	9	-1	8	5	9	6	6	0	8
8	5	6	6	8	6	6	7	-1	6	0

Figure 3: An example roadsLayout matrix with 10 elements

3.2 Erlang-k distribution

The disposal distribution rate and shape parameters are experimented to study the effects on the bin collection performance. The busier the service area is the more frequent bin collection should be. In this experiment, a service area with 30 bins has been used. Several rate and shape parameters have been used. These Erlang-k parameters are also compared with threshold value.

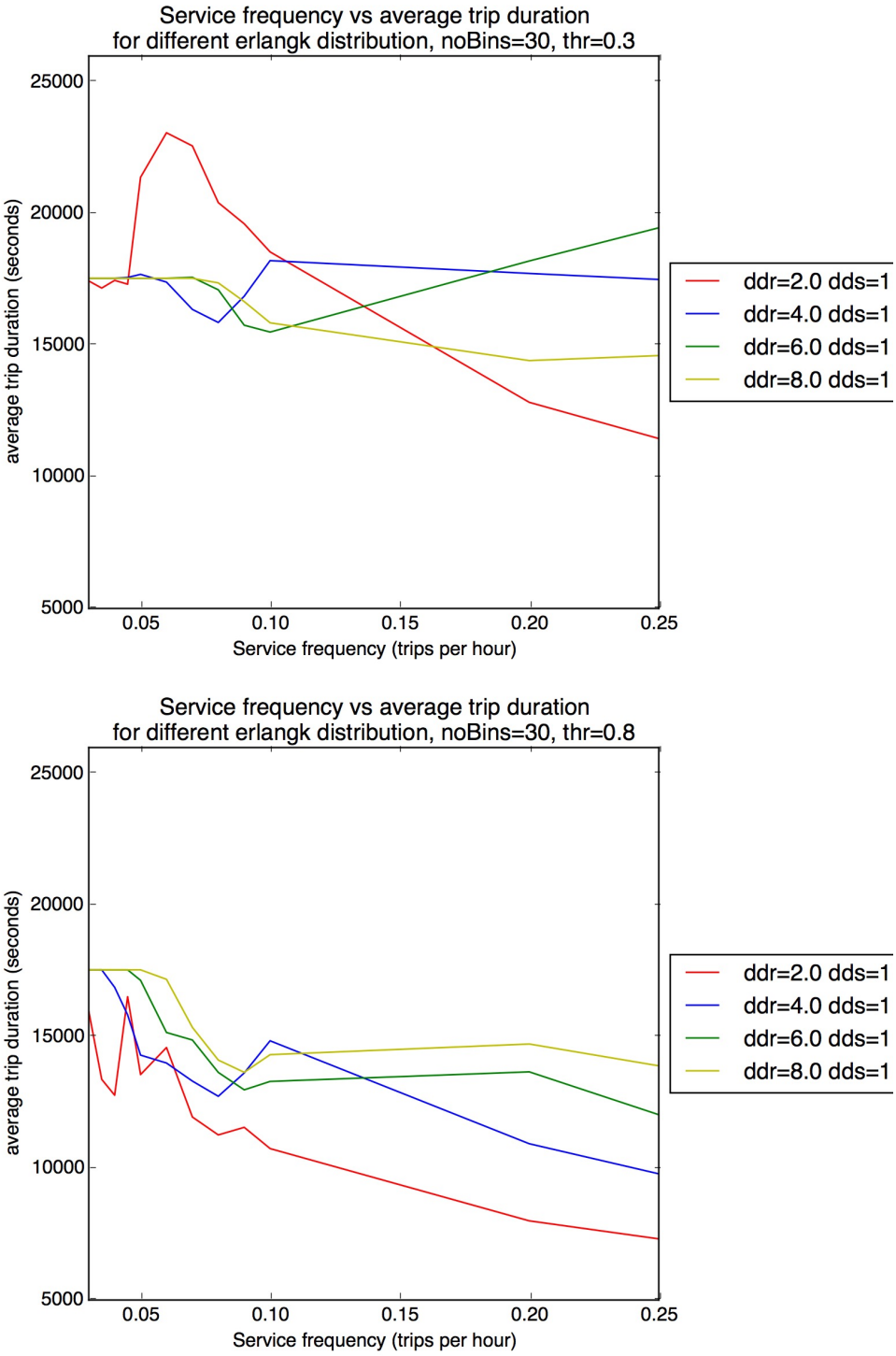


Figure 4: Service frequency vs average trip duration for a service area with 30 bins and different threshold values: 0.3 (above) and 0.8 (below). The trips are lengthier the lower the threshold value, as the lorry would have to service more bins. The lower the rate (ddr), the less lengthy a trip is, since bins get filled up slower.

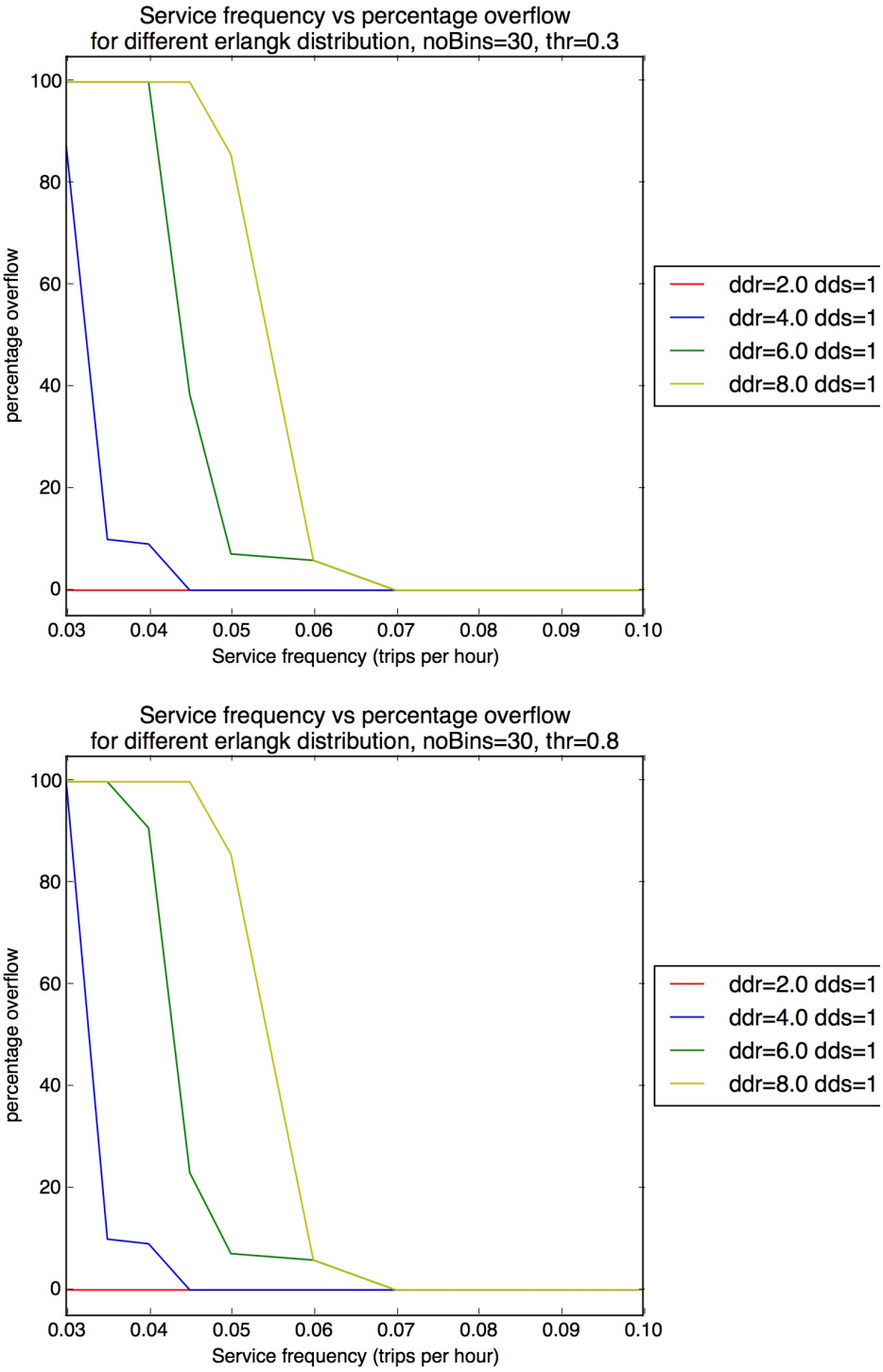


Figure 5: Service frequency vs overflow percentage for a service area with 30 bins and different threshold values: 0.3 (above) and 0.8 (below). A higher threshold value would require a higher service frequency in order to keep the overflow percentage low.

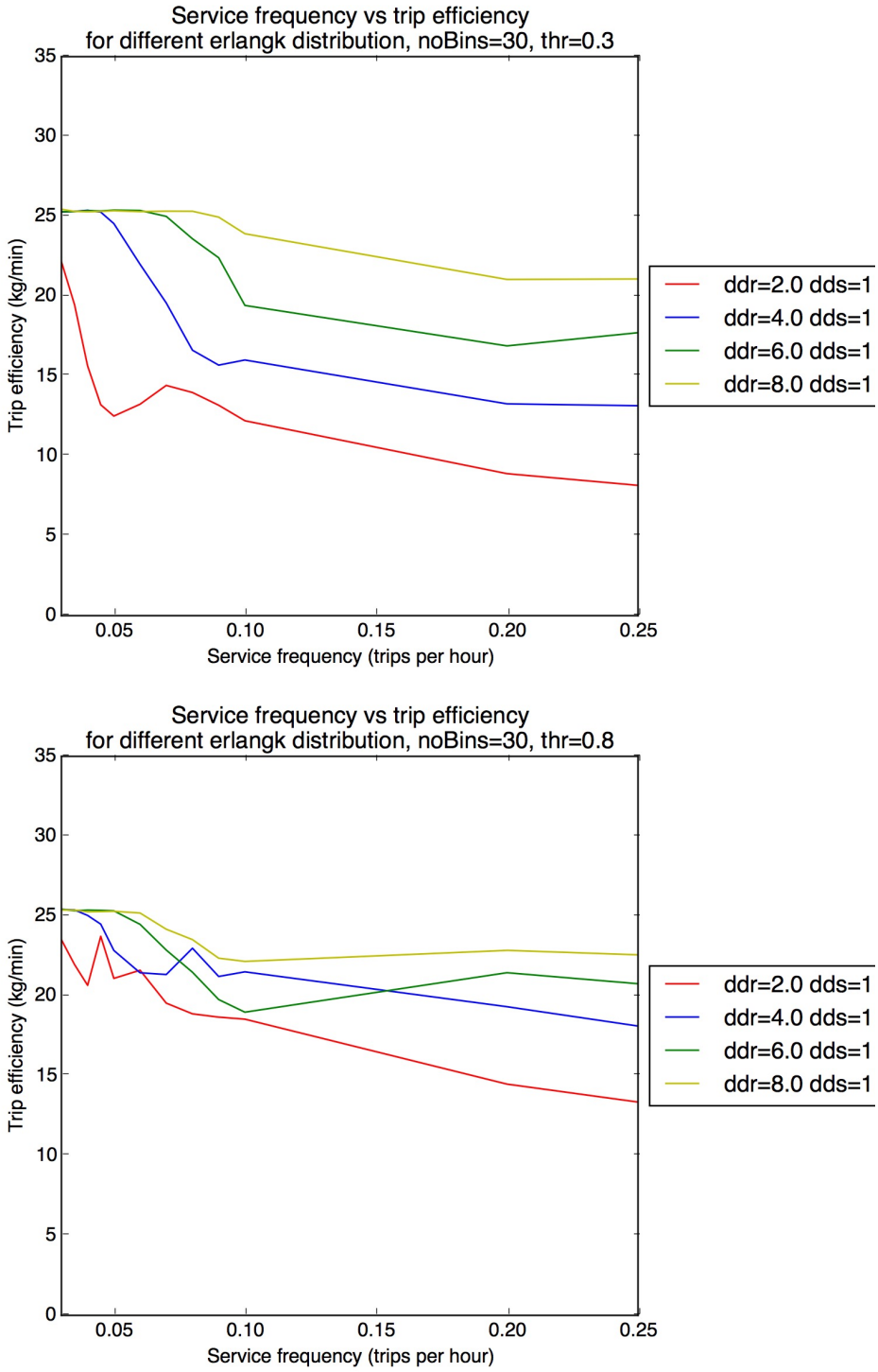


Figure 6: Service frequency vs trip efficiency for a service area with 30 bins and different threshold values: 0.3 (above) and 0.8 (below). This suggests that for a low distribution rate (when compared with the same shape) would benefit from a higher threshold.

The Erlang-k distribution would have an impact on the threshold values and what the optimal size (distance between bins) of a service area should be. It would be more efficient to assign multiple small service areas in an area with high disposal rate.

3.3 Number of bins and threshold values

Experiments were performed with increasing matrix size to test efficiency, and as expected, the greater the matrices are, the longer it takes. A surprising finding is that the percentage overflow does not increase at the size of matrix increases. See the following figures for percentage overflow for different matrix sizes

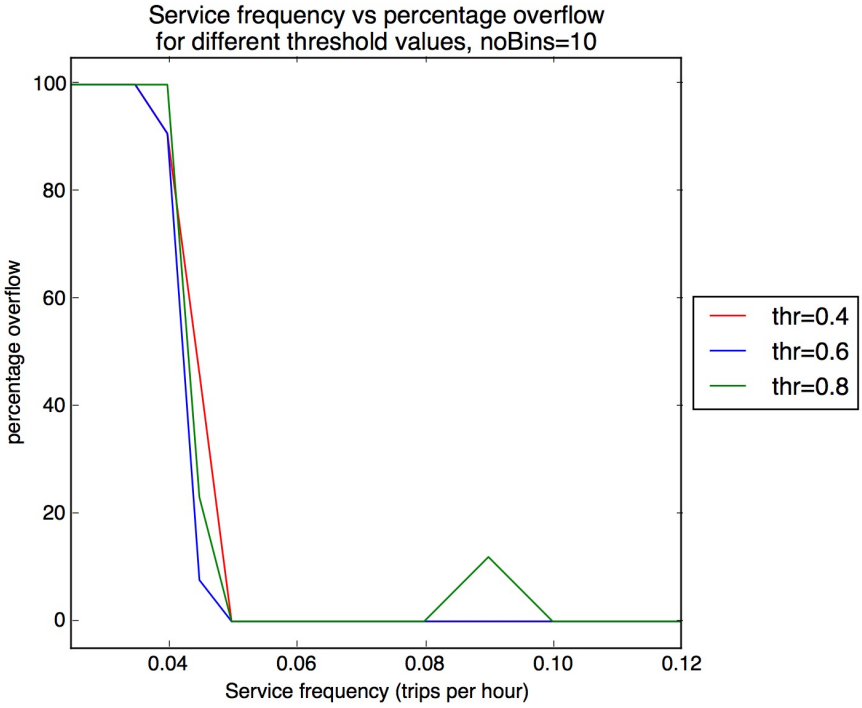


Figure 7: Service frequency vs overflow percentage for different threshold values for a service area with 10 bins. For noBins = 10, a threshold value of 0.6 shows the best performance and would go to 0 for a service frequency between 0.04 and 0.05, which could be practical.

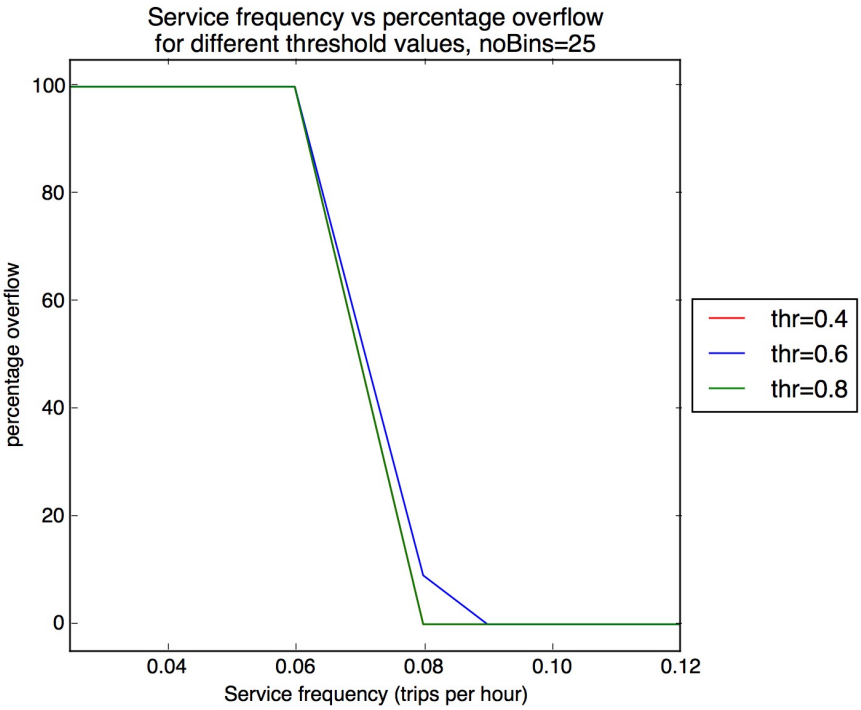


Figure 8: Service frequency vs overflow percentage for different threshold values for a service area with 25 bins. For noBins = 25, a higher threshold value than 0.6 shows a better performance with percentage becomes close to 0 at just before service frequency = 0.8. This is higher than the service frequency for noBins = 10. Also, notice that the decrease is more gradual compare to the previous plot. This suggest a easier control on adjusting the parameters, which is desirable for experimenting and optimising with other inputs.

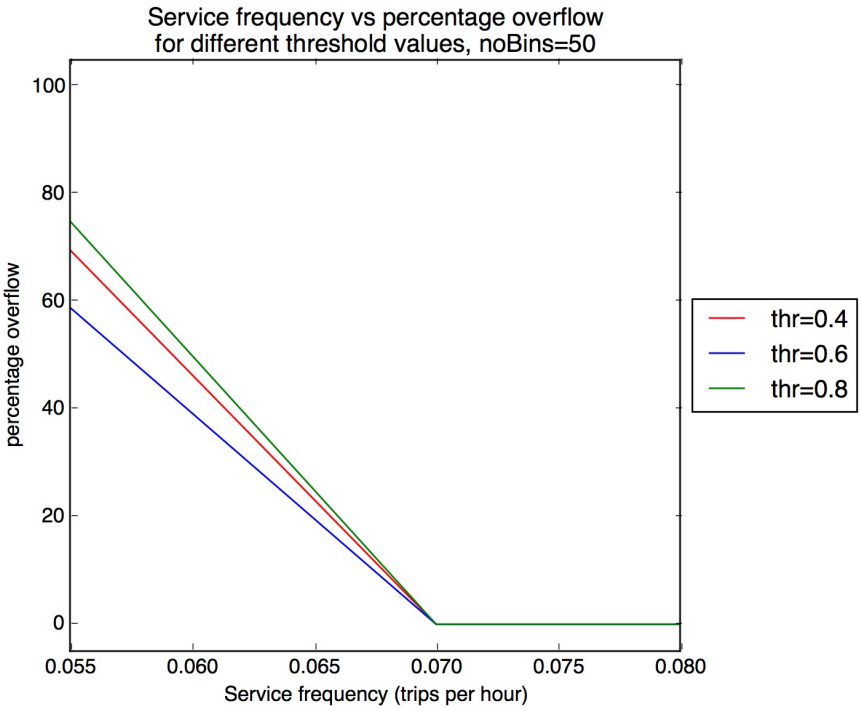


Figure 9: Service frequency vs overflow percentage for different threshold values for a service area with 50 bins. This plot has an even more gradual decrease in overflow percentage, The threshold at 0.6 also seems to give the best performance, overall. Notice that, the service frequency required for a small overflow percentage is lower than that of noBins = 25. One hypothesis is that the lorry will be carrying out lengthy servicing events and thus, after a transient period, the lorry would seemingly be carrying out bin collection events constantly. This could perhaps mean starting one scheduling straight after one finishes due to delay.

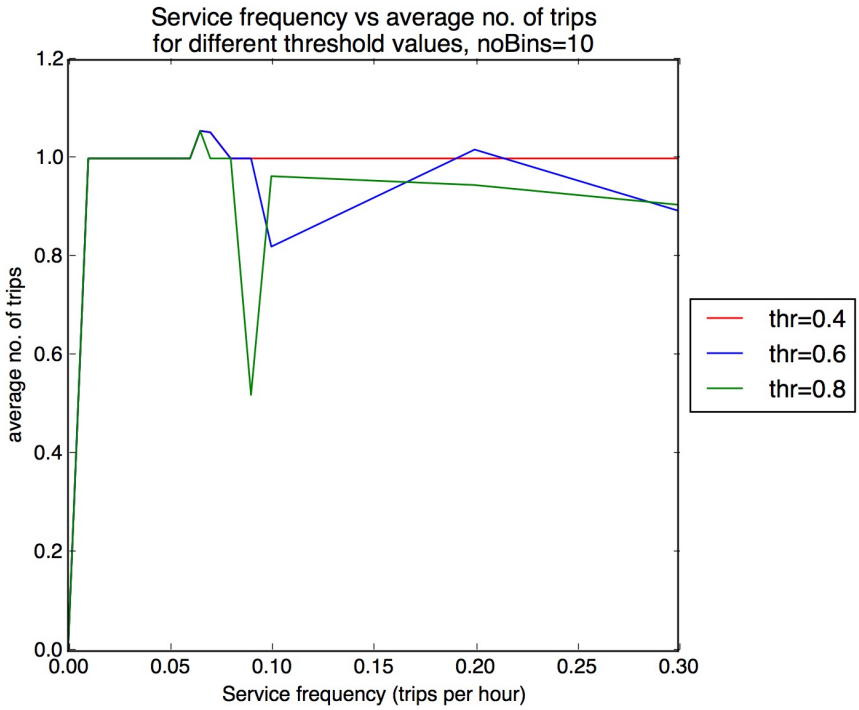
In conclusion for the study in the relationship between service frequency and threshold value, with different service area sizes, is that for a lower overflow percentage, we require more frequent bin collection events such that the figures fall to 0. However, this relationship does not continue as the service area's size increase. This could mean we would have to adjust other parameters such as disposal distribution rate and shape parameters.

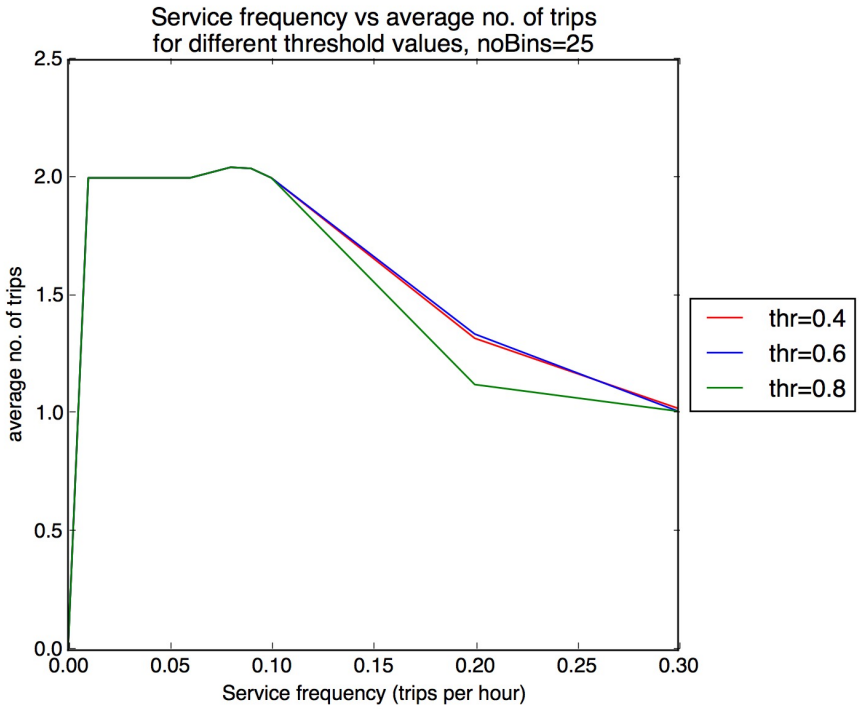
3.4 Service frequency and threshold value

In this experiment, the aim is to find a relationship between service frequency and threshold in order to optimise trip efficiency. Compared to a large service areas, smaller service areas can have a higher threshold values since the lorry is now required to travel less distance.

The range of service frequencies are between 0.05 trips per hour and 0.30 trips per hour, this is equivalent to bin collection scheduled every 20 hours and 3 hours, respectively. It is reasonable for bin collection to be carry out once per day or every two days, depending on the bin

capacity. It is not ideal for a city to have bin collection scheduled every 3 hours, however, this is not a concern in this project. Thus, the smaller the service frequency, the more practical the system.





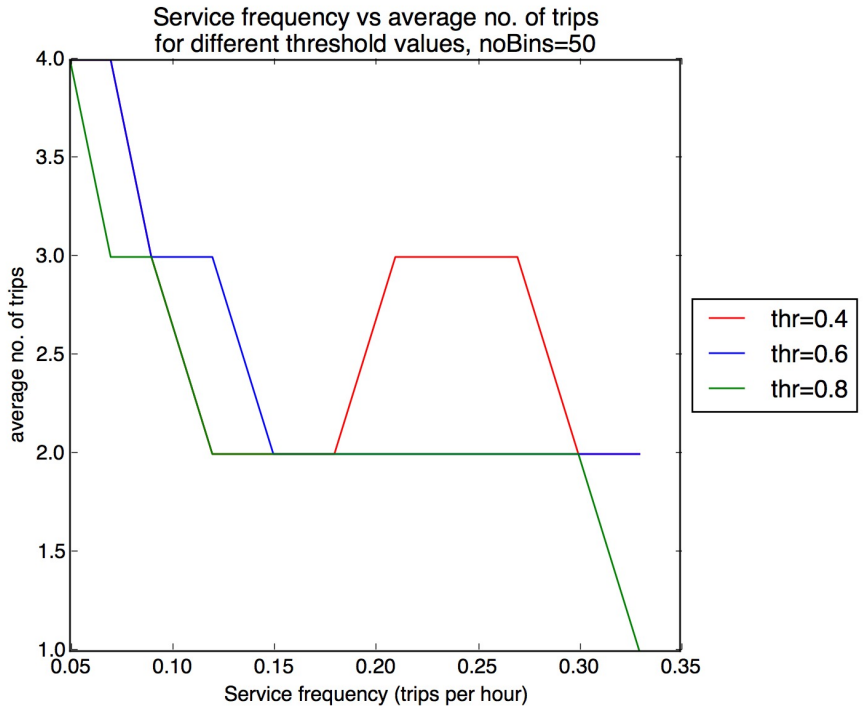
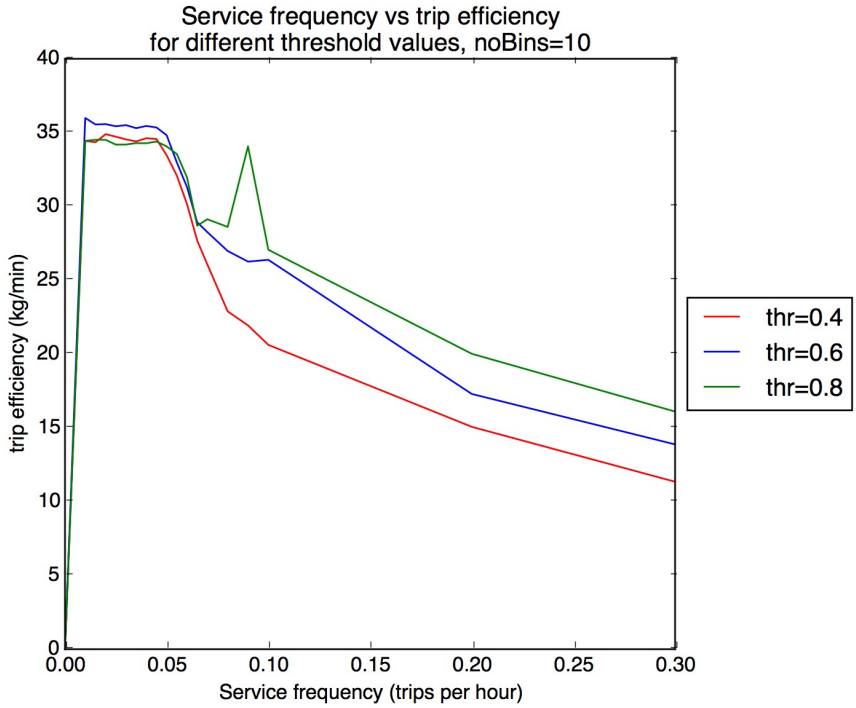
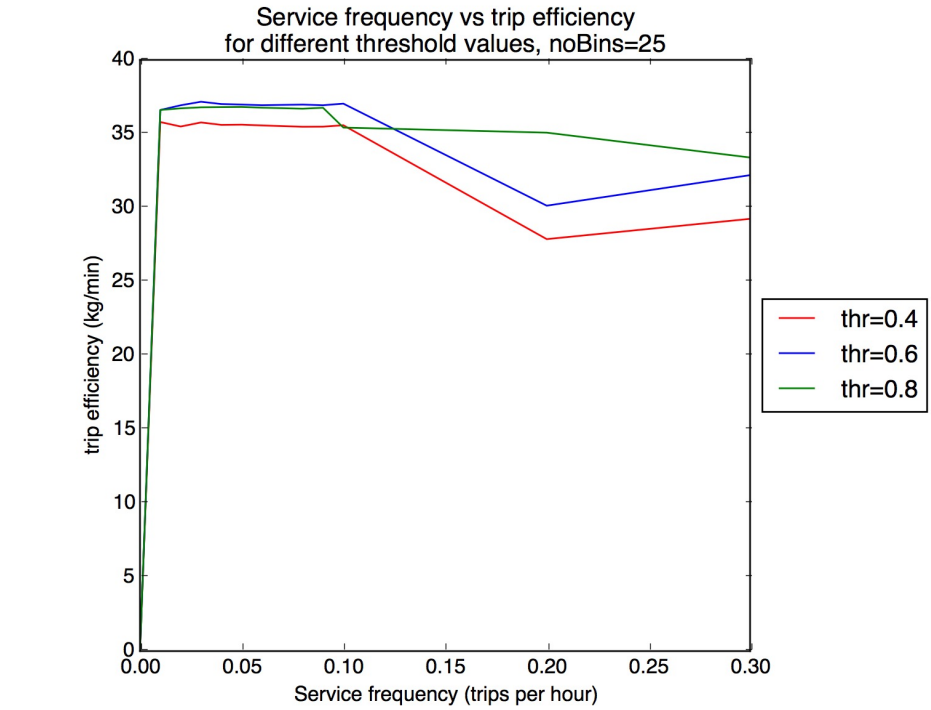


Figure 10: The above graphs shows that for a service area with a small service frequency, the number of trips has to be made is affected by the service area sizes. The smaller the service area, the less the number of trips. This relationship, however, does not hold if the service frequency is high. For a large area, if it wants to perform less trips per schedule, it just has to increase its threshold value.





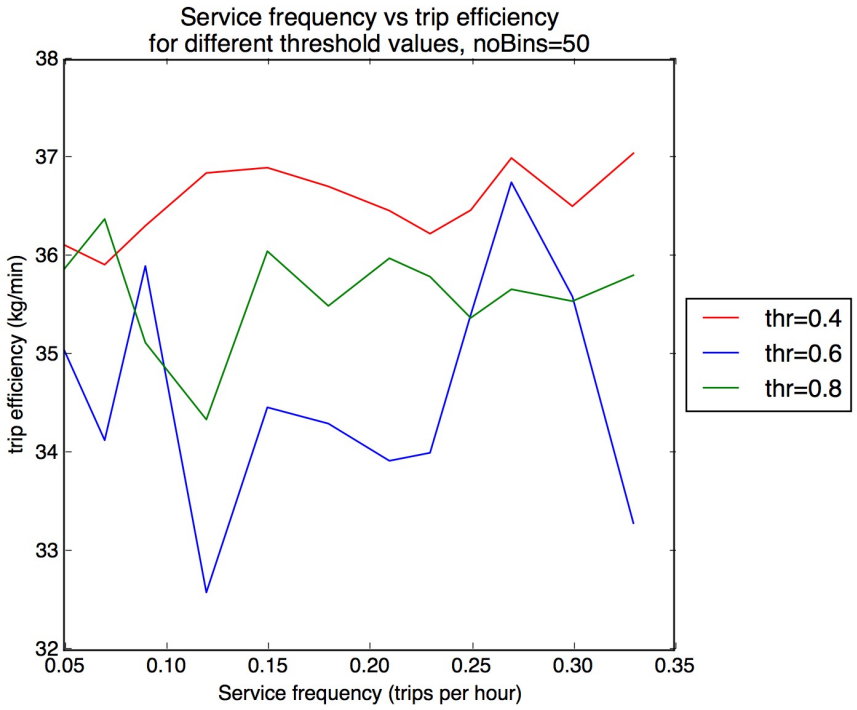


Figure 11: The different in trip efficiency is obvious for service area with 10 bins and 50 bins. For low service frequencies, noBins = 10 shows a much better performance than that of noBin = 50's, which is not the case at the service frequency increases. This plots also show that the lower the threshold value, the better the performance for a large service area.

4 Design choices

In this simulation, various design choices have to be made which will be discussed in this section. Essentially, all input parameters will affect the performance of the bin collection schedules, and may affect the efficiency of this program. Input parameters such as `lorryVolume, lorryMaxLoad, binServiceTime, binVolume, bagVolume, bagWeightMin` and `bagWeightMax` should be set to resemble a realistic scenario.

Parameters we would like to study and design to optimise bin collection performance are `disposalDistRate, disposalDistrShape,serviceFreq, thresholdVal, noBins` and `roadsLayout`. Note that variables such as `serviceFreq` and texttttthresholdVal should be dependent on each other - increasing one should decrease the other in order to prevent overflowing of bins or the lack of bins to be serviced during each schedule. Such relationship is what this project aims to obtain.

4.1 Route Planning

At the start of every schedule, the simulator will determine, for each service area, the bins that schedule should service and its travelling route. Relevant code from the `ServiceArea` class is shown below.

```
1 public class ServiceArea {
2   private int[][] minDistMatrix;    // shortest path between all locations
3   private ArrayList<Integer> serviceQueue = new ArrayList<Integer>();
4   private BinServiceEvent binServiceEvent;
5
6   public void insertToQueue(int location);  // add location to queue
7   public int[] createRequiredVertices();
8   public int[][] createRouteLayout(int[] requiredVertices);
9   public void computePath();
```

4.1.1 Shortest path algorithm

Given a matrix of distances (`roadsLayout`) between locations, a matrix `minDistMatrix` can be computed to give the minimum distance between any pair, from one location to another. Various shortest path algorithms could be used. This simulation uses the *Floyd-Warhshall shortest path algorithm*. This algorithm is implemented in the `FloydWarshall` class[1].

This algorithm is picked because it is straightforward to implement and will always find the lengths of the shortest paths between all pairs of vertices, although its complexity is high - $\Theta(n^3)$. The resulting `minDistMatrix` will always be a complete graph given that the `roadsLayout` matrix is appropriate (This will be discussed in 5).Once this minimum distance matrix is obtained, a sub-matrix containing the relevant bins only will be computed such that we know the minimum

distances between all pairs of bins that required servicing, ignoring the details of route (such at the locations it will go pass).

4.1.2 Graph-traversal algorithm

In order to obtain a route that will pass all bins in the sub-matrix, graph traversal algorithms such as *nearest-neighbour* and *brute-force* [2] have been explored. The brute-force algorithm is too slow when the matrix is too big, in which case the nearest-neighbour is a quicker method. In order to reduce the cost of the simulator, the algorithm in route planning has been chosen to be brute-force if the size of the sub-matrix is less than 12. This threshold, i.e. the size of matrix, has been determined by running the brute-force algorithm with different matrices and compares the time it took to compute the route (See `/testing/CompareBruteForce.java` file for testing effort). The threshold has been chosen to be 10 for optimal performance. Thus, which a `minDistMatrix` of size 10 or under, a brute-force approach will be used to compute route and thus a optimal result will be guaranteed. Otherwise, nearest-neighbour approach will be used instead to reduce time complexity of the algorithm, giving us the upper bound of the shortest path.

4.2 Choice of programming language

Java has been chosen to implemented this discrete stochastic simulation

An object-oriented programming language would suit well to model the entities in our problem domain. **Java** is chosen because it contains classes such as `PriorityQueue` which is are effective in implementing our simulator. For future developments of more complex models, Java is feasible to develop compatible and reusable simulation components, thus suitable for that purpose. The program could also be further implemented as a web-application with its graphics facilities, such that people across the world could access it through the web. However, the main disadvantage is that it has a longer simulation run times compared with programming languages such as C. However, we prefer object-oriented programming instead of procedure oriented programming since this is more efficient for storing system state values for data collection and experimental analysis. Java is also easier for error handling and thus easier to debug.

5 Testing

JUnit testing	
Test file	Aspects tested
AbstractEventTest.java	Checks that time is converted into DD:HH:MM:SS correctly.
BagTest.java	Checks that the bag's content are within its limit.
BruteForceTest.java and NearestNeighbourTest.java	Checks that the correct route is returned.
CompareBF.java	Compares the performance of brute-force under different sizes of graphs.
ServiceAreaTest.java	Checks the `getTimeBetweenLocation()` to obtain distance between bins. Checks the `createRouteLayout()` to obtain the minimum distance matrix with the Floyd Warshall algorithm.

Table 2: JUnit test cases located in the */cslp/src/test/* folder.

Testing with input scripts	
Input scripts	Aspects tested
inputX.txt	Checks that the input parameters follow a specific format.
exptestX.txt	Checks that all experimentations are performed.
standard.txt and standardexp.txt	Accurate input scripts which should not raise warnings, errors or exceptions.
summaryStats.txt	Checks that the information collected are accurate.
test1.txt and test2.txt	Checks that bin service events are correctly generated; service queues are accurate; lorry location is accurate throughout the simulation; bin/lorry emptying events are carried out properly and update content information; handles delays and rescheduling events; and bins in queue remains the same after rescheduling.
testMatrix.txt	Matrices are also checked to determine whether it is suitable for the simulator. The matrix should not contain any islands in order to perform the shortest path and graph traversal algorithms.

Table 3: Input test scripts located in the */input_scripts/* folder.

The `Logger` class was used to output insight details about the simulation for checking purpose. This can be enabled to print onto console with a flag raised when running `simulate.sh`. See the following for an example of checking route planning and bin servicing events.

```
Dec 21, 2016 12:53:48 AM cslp.BruteForce getRoute
INFO: Route: 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 0 -> END    Total duration =
    4080

Dec 21, 2016 12:53:48 AM cslp.BruteForce getServiceQueue
INFO: serviceQueue: 1 -> 2 -> 7 -> 8 -> 10 -> 11 -> 15 -> END

Dec 21, 2016 12:53:48 AM cslp.BinEmptiedEvent execute
INFO:   Current time: 601798 (06:23:09:58)
  Current Location: 2
  Before emptying: bin weight = 326.64642 vol = 1.9599992 Lorry weight = 481.5198
    vol = 1.4399992
  After emptying:  bin weight = 0.0 vol = 0.0 Lorry weight = 808.16626  vol =
    2.419999
  Inserted LorryDepartureEvent for areaIdx = 0  eventTime = 601798 (06:23:09:58)
  Current location = 2  Current destination =  7
  Go to next bin.
```

Listing 1: Example of logger output

6 Discussion

6.1 Limitations

Limitations of the simulator include:

1. Limited memory which would not support service area with a huge number of bins.
2. Most of the parameters have little flexibility. It would be more realistic for parameters such as binVolume to vary, and for some service areas to have multiple lorries, etc.

6.2 Improvements

In this section, we shall discuss limitations of our simulation and possible further improvements.

1. Memory management could be improved.
2. More shortest path and graph traversal algorithms could be explored and tried.
3. Implement the random path algorithm and compare it with nearest neighbour and brute-force to explore its performance.

4. In some instances a `Java Stack` should be used instead of an `ArrayList`.

5. Implement route planning such that take into account the lorry's capacity constraint.

6. Test more aspects of the parameters and explore more about their relationship with each other.

7 Conclusion

This report describes the Computer Science Large Practical 2016 project, a command line program for discrete stochastic event simulation. This report explored different algorithms to solve the bin collection problem and evaluate the performances. Results from different input parameters have been shown to determine the relationship between variables in order to obtain the optimal result to our bin collection problem.

References

[1] Tushar Roy. *Floyd-Warshall All Pair Shortest Path* [Computer Code]. Available at `http://www.akira.ruc.dk/ keld/research/JAVASIMULATION/JAVASIMULATION-1.1/docs/Report.pdf`

[2] Sanfoundry Global Education & Learning Series ? 1000 Java Programs. *Java Program to Generate All Possible Combinations of a Given List of Numbers* [Computer Code]. Available at `http://www.sanfoundry.com/java-program-generate-all-possible-combinations-given-list-numbers/`

[3] Dr. Mesut Gunes. *Modeling and Performance Analysis with Discrete-Event Simulation.* Rwthacchen University.