Computer Science Large Practical 2016–2017

Dr Paul Patras

School of Informatics

Issued on: Monday 19th September, 2016

The CSLP coursework handout is structured as follows:

1 Introduction

The requirement for the Computer Science Large Practical is to develop a command-line application, the purpose of which is to *execute stochastic simulations of the bin collection process in a "smart" city.* This may help city councils obtain insights into how waste management operations can be improved. To develop the simulator you may use any programming language you are most familiar with, *as long as your code will compile and run on DiCE.*

— ◇ —

Stochastic simulation is an important tool in many fields such as physics, medicine, computer networking, logistics, etc. and is particularly useful to understand complicated processes. Unlike deterministic simulations, stochastic simulations with the same input may produce different outputs. This is due to the fact that some events are intrinsically random.

— ◇ —

Different to previous software development assignments you may have had, this practical will expose you to the design and implementation of a more complex system that incorporates realistic constraints specific to large processes. Although a set of requirements is given below, some of these are intentionally incomplete, to allow you the choice of implementation methodology. As this is a relatively large task, your source code must be clean, commented and reusable. This is an individual assignment and thus code sharing is not permitted.

— ◇ —

At the end of this course, you are expected to produce a written report that explains the key building blocks of your design, discusses the results of the analyses you performed with different inputs, and summarises your most important findings. These can be accompanied by graphs plotted based on the numerical output of your simulations. There should be evidence that *your implementation has been thoroughly tested*, and you must also submit input files you have generated yourself. Take time to understand if the produced output is sensible, i.e. the numerical values obtained would make sense in a real setting.

It is essential that you read and implement carefully the requirements of the assignment. In particular, submissions that do not accept correctly formatted input or produce output with incorrect format will lose marks, since **your code will be put through automated testing**.

— ◇ —

In what follows the requirements of your application are detailed, including input formatting, the expected application behaviour, and the format of the output. There are three deadlines associated with this coursework. The first part is a *formative assessment, for feedback only.* As such it is not mandatory that you submit something, but you are *highly encouraged to do so.* This is an opportunity for you to submit a short proposal document that outlines early on the structure and key design choices you plan to make in developing the simulator. This will enable you to obtain feedback prior to submitting the part 2, which is a summative assessment, worth 50% the overall assessment. The report and code you submit for the third deadline weight additional 50% of the final mark. Further details are given in parts 1–3.

2 Requirements

In this section the requirements of the application you must submit are discussed.

2.1 Problem Domain

To be able to simulate a real system, first it is required to construct a model that captures the most important aspects of the system that will be studied. Not all the features of a real system may be relevant to a specific problem. The challenge is to identify the important elements to be modelled and omit others to reduce complexity, while still obtaining information that helps understanding the system's behaviour.

Different inputs can be given to a well designed simulator to obtain estimates of different parameters of interest. Such simulation-based studies are quicker and significantly cheaper than approaches based on extensive measurements of the real system. In addition, simulations allow to investigate exceptional scenarios that are foreseeable and likely problematic, but for which a real data set is not yet available.

Our focus is on the bin collection process in a city. We are interested in a smart city scenario where community bins are equipped with sensors that indicate their current occupancy. Such deployments are expected to be rolled out in densely populated cities in order to improve the scheduling and route planning for the lorries that dispose of community waste. The City of Edinburgh Council is currently trialling such a scheme.

Traditional collection strategies widely used today suffer from two major drawbacks. First, lorries are scheduled periodically at fixed intervals and always follow the same routes. As such, they often make unnecessary frequent trips and sometimes take lengthy routes, while many bins are far from exceeding their capacity. This increases the operational cost as well as the operation's carbon footprint. Second, user daily demand often varies significantly and bins can overflow much earlier than scheduled pick-ups. This triggers health hazards, degrades the city landscape, and incurs additional cleaning costs for city councils.

In this practical we will build a simulator of a proposed system in which the bins have occupancy sensors that indicate the volume and weight of the waste they store. Such measurements can be used to assist scheduling collections and tackle the aforementioned issues. More precisely, we will investigate the efficiency of the waste management process in terms of weight collected per time unit when different service intervals and bin occupancy thresholds are used. We will consider each lorry has a specified capacity in terms of maximum volume and weight it can carry, and is assigned a single service area. In each area road layouts and the distances between collection points are known. Upon each schedule, the decision of whether or not to visit and empty a bin will be made based on configurable occupancy thresholds. The route taken by each lorry is planned accordingly at the start of a service and is not updated while that lorry is in service. Routes begin and end at the depot. One can infer that small thresholds can potentially lead to longer journeys and lesser operation efficiency, but ensure cleaner streets. On the other hand using large thresholds can prove more cost efficient, but may lead to a larger average number of bin overflowing per day, which is another metric we wish to estimate. The simulator will further allow us to understand whether a system is under provisioned, e.g. an area may be too large for a single lorry to service effectively.

2.2 Simulation Outline

The underlying simulation algorithm is fairly straightforward and is outlined below.

> time ← 0
> **while** time ≤ max_time **do**
>   determine the set of events that may occur after the current state
>   delay ← choose a delay based on the nearest event
>   time ← time + delay
>   modify the state of the system based on the current event
> **end while**

A couple of clarifications with respect to the above pseudo-code are appropriate. The set of possible events include *(i)* a rubbish bag was disposed of in a bin, *(ii)* the occupancy threshold of a bin was exceeded, *(iii)* a bin overflowed, *(iv)* a bin was emptied, *(v)* a lorry was emptied, and *(vi)* a lorry arrived/departed form a location (bin or depot).

Users will dispose of rubbish bags at time intervals sampled from an Erlang distribution with known shape k and rate λ, which will be given as input parameters. An Erlang-k distribution is effectively the distribution of the sum of k independent exponential variables with mean $\mu = 1/\lambda$. You can draw from an exponential distribution with mean μ, by computing $-\mu \cdot \log(\text{rand}(0.0, 1.0))$, where $\text{rand}(0.0, 1.0)$ returns a random number between 0.0 and 1.0 (exclusive). It follows that the mean delay between two bag disposal events is k/λ. We will consider bag disposal events to be independent of each other and identically distributed (i.i.d.). That is a disposal event at a given bin does not depend on previous disposal events at the same or other bins. We will consider bin service events to be deterministic. That is lorries are scheduled at fixed time intervals.

When determining the set of events that may occur after the current state, each of these will have an associated delay. To advance the simulation, you will first need to find the closest one, i.e. the one with the smallest delay. Then you will update the simulation time with this delay, execute the corresponding event and update the system states accordingly.

This procedure is repeated until max_time expires.

2.3 Simulation Components

Your simulations must include the following elements:

- **Users.** At any bin, we consider users dispose of waste bags at time intervals that follow an Erlang-k distribution with the rate and shape given as an input parameters. We assume a bag is of fixed volume (in cubic meters), given as input. A bag's weight (in kilograms) is a random value, uniformly distributed between a lower and an upper bound, which are also input parameters.

- **Bins.** All bins have a fixed volume that is specified as an input parameter. We assume bins do not have maximum weight limitations, but we consider such limits on the waste bags disposed (as above). We consider bins are equipped with sensors that indicate their current occupancy as a fraction of their maximum volume. Sensors also track the weight of the current contents. In addition, you should account for the event that a bin has 'overflowed'. Since the system relies on bin sensors, such events can only be tracked at most once between two bin service instances. It is also acceptable to assume the occupancy

of a bin may exceed capacity, when it becomes full after the disposal of one bag. Again, for any bin this can happen at most once between two service instances.

- **Lorries**. Lorries are scheduled periodically at fixed intervals and their daily frequency is given as input. Lorries have fixed capacity both in terms of volume and weight and these values are input parameters. On the other hand, upon service a lorry compresses the contents of a bin to half its original volume. We will consider a bin is serviced (emptied) in constant time (expressed in seconds), irrespective of the load of a particular bin being emptied. When at depot we will consider the time required to empty a lorry is also fixed and this is five times as long as the bin service time.

- **Service Areas.** We will assume a city divided into multiple service areas, each comprising a specified number of bins. For this practical we will assume the number of bins in an area can be represented as a 16-bit integer value, i.e. the maximum number is 65,535. Each service area is assigned a single lorry. The locations of the bins, the road layout, and the time it takes to drive between neighbouring bins (expressed in minutes) are also given. We will use a directed graph representation to model this, as shown in the example illustrated in Figure 1. The total number of bins in each area and the graph representation of the distances between them (arc costs) will be provided as an input in matrix form. We thus account for both one-way and two-way streets.
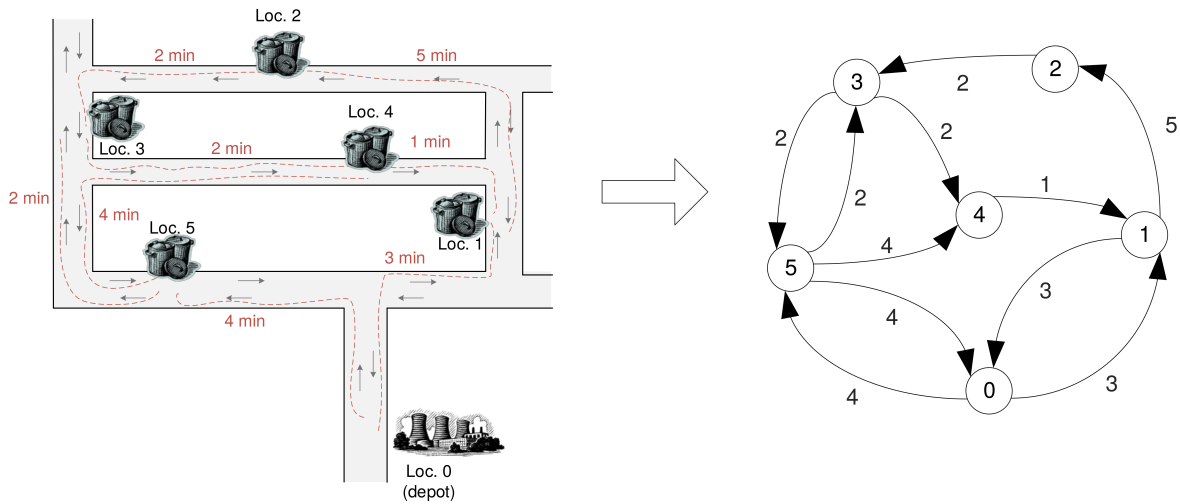

Figure 1: Example scenario with a single service area and the corresponding graph model. The vertices represent bin locations, the arcs represent the roads between two adjacent bins and the arc costs the duration (in minutes) required to travel between them.

- **Route Planning**. At every schedule, an occupancy threshold will be used to decide which bins need to be serviced. This threshold is a given input parameter based on which you must compute the appropriate route a lorry must take within an area. Note that all routes are circular, i.e. they must start and end at the origin (the depot).

There are several important aspects here. First, there may not be a direct link between all bins that must be visited. Thus, when computing a path, you may have to keep track of the intermediary arcs between vertices where the occupancy threshold was not exceeded and adjacent ones where this has happened. Second, you must compute optimal routes that visit all bins requiring service and minimum costs. How you implement this is entirely your choice. For instance you may choose to run shortest path algorithms such as Johnson–Diskstra, Floyd–Warshall, etc. and combine these with heuristics that ensure paths start at ends at the depot. The latter is generally known to be difficult as the

optimal solution cannot be found in polynomial time. There are however several heuristic algorithms that work well in finding a solution most of the time. Two such approaches that are easy to implement are the "Nearest Neighbour" and "Sorted Edge" algorithms. Exploring different solutions is appropriate.

Note that when e.g. a small number of vertices need to be visited, it is often more efficient to travel multiple times through the same location, even if the route previously serviced a bin there. Further, for small areas, even a brute-force approach that computes all the possible routes and selects the one with the small cost may be appropriate.

Thus when planning the collection route, you are free to choose any existing heuristic or implement your own algorithm, but you are expected to document your choice and discuss its implication on system's performance in your written report.

Finally, if a lorry cannot service in a single trip all the bins whose occupancy exceeds the predefined threshold, you may have to perform this task in multiple steps. A lorry cannot receive an updated route from the depot while in service, if e.g. a bin's occupancy threshold was reached after the lorry's departure. The occupancy of some bins may increase as a lorry traverses a planned route and as a consequence the lorry may not be able to service all assigned bins due to capacity constraints. Should this happen, you must compute the shortest path from the current bin to the depot, return on that path and immediately restart the service procedure (including new route planning). If a lorry misses a schedule due to a lengthy previous journey, you should schedule right away. Note that such a situation may perpetuate in a cascade fashion.

At the start of the simulation you must accommodate a "warm-up" period to eliminate transient effects and ensure the statistics you collect are representative for steady state operation. The warm-up duration will be given as an input parameter. At start time all bins are considered empty and lorries stationed at the depot (location 0). Waste disposal and collection processes can fire off at any time, as long as they do so according to the timing parameters specified in the input.

From the description of the components you should be able to identify the key requirements for the simulator you will develop. When implementing these, you are expected to include comments in your code that explain these requirements, as well as the design choices you made.

2.4   Command-line Arguments and Bash Script

The application you will develop is to be executed at the system console and should accept command-line arguments. It is mandatory that one argument is the name of the input script that describes your simulation. You can optionally add other parameters, e.g. for logging purposes or to allow disabling detailed output when repeating the simulations with different values of a certain input.

To allow you to develop the simulator in the programming language of your choice, while ensuring your code can be marked through automated testing, you will have to modify a Bash script that launches your application. A skeleton script named $\boxed{simulate.sh}$ will be given. You must ensure this script is updated such that it launches your application correctly with one or more command line arguments. As such, running your code at the console will be done as follows:

```
$ ./simulate.sh <input_file_name> [OPTIONS]
```

When your simulator is run without arguments, you must display usage information.

The output of your simulations should be printed to the system standard output (`stdout`). The input and output formatting is specified next.

2.5 Input Formatting

Your input file will contain a set of global parameters, including lorry volume, maximum lorry load, bin service duration, bin volume, rate and shape of the distribution of rubbish bag disposal events, rubbish bag volume, and minimum and maximum bag weight limits. Then we define the number of areas. Each area will have specific parameters such as the waste service frequency, the occupancy threshold that triggers collection, the number of bins, and the graph representation of the bin locations with the distances between them, where the first location is considered to be the depot. Except for the roads layout, each object will be described in one line of input. The graph corresponding to an area will be given in matrix form.

Input Specification

The input parameters will be given in the input script according to the following specification:

lorryVolume <uint8_t>
lorryMaxLoad <uint16_t>
binServiceTime <uint16_t>
binVolume <float>
disposalDistrRate <float>
disposalDistrShape <uint8_t>
bagVolume <float>
bagWeightMin <float>
bagWeightMax <float>
noAreas <uint8_t>
areaIdx <uint8_t> **serviceFreq** <float> **thresholdVal** <float> **noBins** <uint16_t>
roadsLayout
```
 0        <int8_t>  <int8_t>  ...   <int8_t>
 <int8_t>  0        <int8_t>  ...   <int8_t>
 ⋮         ⋮         ⋮         ⋱    ⋮
 <int8_t>  <int8_t>  <int8_t>  ...   0
```
areaIdx <uint8_t> ...
...

stopTime <float>
warmUpTime <float>

We will work with lorry volumes and maximum loads expressed in cubic metres and respectively kilograms. The time required to empty a bin is a constant independent of bin's load and is given in seconds. The volume of the bins is expressed in cubic meters. At each bin bags are disposed at time intervals that follow an Erlang-k distribution, for which the average rate is expressed in numbers of bags per hour, and the shape (k) is a positive integer. The individual bag volume is given in cubic metres and the weight will be a random number uniformly distributed in the `bagWeightMin -- bagWeightMax` range, expressed in kilograms (Note: use 3 decimal points). For each area, the input file must contain an area index, the service frequency (given as number of trips per hour) and the occupancy threshold that will trigger waste collection (a value between 0 and 1). Finally, the `roadsLayout` keyword introduces the matrix representation

of the duration in minutes between two locations in an area. The (0,0) element corresponds to the depot, while elements (i, j) are zero when $i = j$ and we use -1 to indicate there is no direct link between two locations. Remember we will be using directed graphs, i.e. there may only be a one way link between two vertices.

In addition to the above, it is essential that you specify when the simulation should terminate and how long is the warm-up time. These will be given in hours. Appropriate conversion to minutes/seconds may be required.

While the order of the parameters is not strict, area descriptions of the areas must follow the the noAreas parameter. Except for the road layout description, all parameters should be given each on a single line.

You may use the **#** character to comment out a line in the input file and you should ignore blank lines. An example is given below.

```
# Total waste volume a lorry can accommodate (cubic metres)
lorryVolume 20
# Maximum lorry load (kg)
lorryMaxLoad 7000
# Time required to empty a bin (in seconds)
binServiceTime 130
# Bin volume (cubic metres)
binVolume 2
# Rate of the Erlang distribution of the disposal events (avg.  no.  per hour)
disposalDistrRate 2.0
# Shape of the Erlang distribution
disposalDistrShape 2
# Bag volume (cubic metres)
bagVolume 0.05
# Minimum expected weight of a waste bag disposed
bagWeightMin 2
# Maximum expected weight
bagWeightMax 8
# Number of service areas
noAreas 1
# For area 0, service performed once every 16 hours (i.e.  0.0625 trips per hour)
# Bins serviced if occupancy is more or equal 75% and there are 5 bins in this area
areaIdx 0 serviceFreq 0.0625 thresholdVal 0.75 noBins 5
# Road layout and distances between bin locations (in minutes)
roadsLayout
  0    3   -1   -1   -1    4
  3    0    5   -1   -1   -1
 -1   -1    0    2   -1   -1
 -1   -1   -1    0    2    2
 -1    1   -1   -1    0   -1
  4   -1   -1    2    4    0
# Simulation duration (hours)
stopTime 36.0
# Warm-up time (hours) allowed before collecting statistics
warmUpTime 12.0
```

2.6 Output Formatting

Your simulator should output information about events in a human readable format. On the other hand the output must be easily parsable, such that specific information can be extracted e.g. for visualisation purposes. Each event will be output on a single line in the following format:
`<time> -> <event> <details>`

We will output time in the format **days:hours:minutes:seconds**. Since some of the input parameters (e.g. distances between locations, bin service time) are given in minutes or seconds, you should make the appropriate conversions and work with second granularity. You must account for the following events:

- bag disposed of
- bin load/contents volume changed
- bin occupancy threshold exceeded
- bin overflowed
- lorry arrived at/left location
- lorry load/contents volume changed

Output Specification

The output formatting for the expected events is given below. Note that parts of the **functionality you implement will be automatically tested**. Thus **strictly abiding to the output format is essential**.

`<time> -> bag weighing <float> kg disposed of at bin <uint8_t>.<uint16_t>`

`<time> -> load of bin <uint8_t>.<uint16_t> became <float> kg and contents volume <float> m^3`

`<time> -> occupancy threshold of bin <uint8_t>.<uint16_t> exceeded`

`<time> -> bin <uint8_t>.<uint16_t> overflowed`

`<time> -> lorry <uint8_t> arrived at location <uint8_t>.<uint16_t>`

`<time> -> load of lorry <uint8_t> became <float> kg and contents volume <float> m^3`

`<time> -> lorry <uint8_t> left location <uint8_t>.<uint16_t>`

Note that we index lorries according to their service area and `bin x.y` refers to bin y in area x.

Example

Example excerpt of valid output is shown next.

```
...
00:14:30:00 -> bag weighing 5.2 kg disposed of at bin 0.2
00:14:30:00 -> load of bin 0.2 became 140.1 kg and contents volume 1.4 m^3
00:14:52:20 -> bag weighing 4 kg disposed of at bin 0.4
00:14:52:20 -> load of bin 0.4 became 158.3 kg and contents volume 1.5 m^3
00:14:52:20 -> occupancy threshold of bin 0.4 exceeded
00:15:24:20 -> bag weighing 6.2 kg disposed of at bin 0.1
00:15:24:20 -> load of bin 0.1 became 180.1 kg and contents volume 2 m^3
00:15:24:20 -> bin 0.1 overflowed
00:16:00:00 -> lorry 0 left location 0.0
00:16:03:00 -> lorry 0 arrived at location 0.1
00:16:05:10 -> load of bin 0.1 became 0 kg and contents volume 0 m^3
```

```
00:16:05:10 -> load of lorry 0 became 161.1 kg and contents volume 0.9 m^3
00:16:05:10 -> lorry 0 left location 0.1
...
```

Statistics Analysis

In addition to reporting the events timeline, your application should return summary statistics at the end of the simulation time. The analyses you are expected to perform are detailed next.

Average Trip Duration

Your simulator should report the average duration of trips performed by a lorry over the duration of your simulation. This will provide an indication of how the bin occupancy threshold impacts route lengths and how well heuristics solving the route planning task work. You should report the average trip duration as a floating point number with two levels of granularity, i.e. per area and overall. The format for displaying these statistics is the following:

```
area <area number>:  average trip duration <average duration (minutes:seconds)>
overall average trip duration <average duration (minutes:seconds)>
```

Here `<area number>` indicates the area for which this statistic is computed.

Number of Trips per Schedule

We will also compute the average number of trips per schedule. This is particularly important to evaluate whether a lorry can service all bins whose occupancy exceeded the given threshold in a single trip. This will be reported as a per area and overall statistics, as follows:

```
area <area number>:  average no.  trips <average number of trips per schedule (float)>
overall average no.  trips <average number of trips per schedule (float)>
```

Trip Efficiency

To be able to assess how efficient the waste collection process is, it is appropriate to compute the average weight collected per unit of travel time. For this purpose you will need to trace for each route scheduled the total duration, as well as the weight of the waste the lorry has collect at the end of a trip. All lorries leave the origin with zero load. The efficiency will be displayed as a floating point number, again with two levels of granularity, i.e. per area and overall, in the following format:

```
area <area number>:  trip efficiency <average weight per time unit (kg/min)>
overall trip efficiency <average weight per time unit (kg/min)>
```

Such statistics may prove useful to adjust the service frequency, or the bin occupancy thresholds.

Average Volume Collected

Similar to the above, it may be useful to measure the average volume of (compressed) waste collected during over a service route. This would be useful to understand whether lorries may have total weight or volume limitations. You will need to trace again for each route the total duration, as well as the volume of waste the lorry has collect at the end of a trip. This metric will be displayed as a floating point number with two levels of granularity, i.e. per area and overall:

```
area <area number>:  average volume collected <average waste volume (m^3)>
overall average volume collected <average waste volume (m^3)>
```

Percentage of Overflowed Bins

Finally, we are interested in monitoring whether bins are serviced frequently enough and whether waste disposal rates cause bins to exceed their capacities in some areas. For this purpose, in each area between the start of two service instance, we will compute the ratio of bins that overflowed and their total number (multiplying by 100). It is straightforward to observe that there are several input parameters that impact this statistic. We will use again the same two level granularity and the following semantic:

```
area <area number>:  percentage of bins overflowed <overflowed bins percentage(%)>
overall percentage of bins overflowed <overflowed bins percentage(%)>
```

Note: You must delimit the start and end of the statistics with a '---' sequence.

Example

An example of valid summary statistics is shown below.

```
 ---
area 0:  average trip duration 32:20
overall average trip duration 32:20
area 0:  average no.  trips 1.00
overall average no.  trips 1.00
area 0:  trip efficiency 22.25
overall trip efficiency 22.25
area 0:  average volume collected 4.7
overall average volume collected 4.7
area 0:  percentage of bins overflowed 0.0
overall percentage of bins overflowed 0.0
---
```

Experimentation

To acquire a better understanding of the impact of certain parameters on the metrics of interest, your simulator should allow experimenting with different means/shapes of the distribution of waste bag disposal events and different bin collection frequencies. For that, in the input file you should be able to specify a set of values instead of a single one. This set will be introduced using the **experiment** keyword, followed by the values. For example, the line

`disposalDistrRate` `experiment 2.0 4.0 6.0`

instructs the simulator to run three simulations with the same parameters and topology, but varying the bag disposal rate. Similarly, we can experiment with the shape of the Erlang distribution of the disposal events, e.g.

`disposalDistrShape` `experiment 1 2 3`

We can also experiment with different service frequencies. To keep things simpler though, for experimentation purposes we will use the same frequency for all areas and here the `experiment` keyword should override the values defined for each area. For example, in the following input script

`serviceFreq` `experiment 0.042 0.083 0.125`
`noAreas` `1`
`areaIdx` `0` **`serviceFreq`** `0.0625` **`thresholdVal`** `0.7` **`noBins`** `5`

although a collection frequency of 0.0625 is explicitly defined for area 0, the line **`serviceFreq`** `experiment 0.042 0.083 0.125` instructs the application to ignore later definitions and instead run three simulations with the list of values given at the start (which in this example correspond to one, twice, and respectively three times per day). For a given topology, it would be appropriate to attempt to find the optimal value of one parameter (e.g. service frequency or occupancy threshold) that maximises trip efficiency or minimises the percentage of bins overflowed.

Lastly, you should also allow experiment with multiple parameters at the same time. When running a set of such experiments, you should *disable the output of detailed information* about the events and instead *only to display summary statistics*. To delimit the different experiments, precede the output of each with the following heading:

```
Experiment #<experiment no.>:  <param1> <param1-value> <param2> <param2-value> ...
---
```

For example, a fragment of valid output would look like this:

```
 ...
Experiment #2:  disposalDistrRate 2.0 disposalDistrShape 2
---
area 0:  average trip duration 32:20
overall average trip duration 32:20
area 0:  average no.  trips 1.00
overall average no.  trips 1.00
area 0:  trip efficiency 22.25
overall trip efficiency 22.25
area 0:  average volume collected 4.7
overall average volume collected 4.7
area 0:  percentage of bins overflowed 0.0
overall percentage of bins overflowed 0.0
---
Experiment #3:  disposalDistrRate 2.0 disposalDistrShape 3
---
average trip duration 33:12
...
```

Visualisation

As you have noticed, one requirement is to produce output in a format that is easy to parse. This is particularly important when experimenting with different values of a parameter, as you may wish to use this output to plot various metrics. For instance you could plot the trip efficiency as a function of the service frequency. Similarly, you may also examine the time evolution of bins' contents volume.

To plot your results you can load your output into spreadsheets (e.g. OpenOffice Calc, Google Spreadsheet, Microsoft Excel, etc.) or you can used specialist plotting programs such as GNUplot, R or matplotlib.

Compiling with Bash Script

To compile your application you will use another Bash script named $\boxed{compile.sh}$, which will launch the suitable compiler (e.g. `gcc`, `javac`, `ghc`, etc.) and produce a binary file based on your source code. This in turn will be later launched with the *simulate.sh* script to execute your application. A skeleton script for compilation purposes will be provided, but you must update it to suit your choice of programming language. As such, to compile your code you must run:

```
$ ./compile.sh
```

Testing

To demonstrate that your application has been thoroughly tested, you will be required to submit a number of test inputs of your own, both valid and invalid. You should add comments in the scripts to explain what you are testing and in your report you should explain the purpose of each test and the results expected.

Important: The functionality of your code will be subject to automated testing with different previously seen and unseen inputs. Thus, even if your output may be semantically correct, it will be regarded as invalid if not abiding to the specified format and you will lose marks. Also, refrain from prompting the user for interactive input; your programme should have enough information to run simulations after parsing an input script. Likewise, if you decide to add additional arguments to the command line, these should be strictly optional and documented.

Source Code Control

It is important that large software projects are well maintained, therefore a good source code control mechanism is used. For this project we will use the `git` source code control system. Your final submission will also be marked on how well you have managed your source code using git, mostly on how appropriate your commits are. Although this is subjective, the general approach should be having a single commit for each "unit of work". Two good rules of thumb are:

- If you cannot describe the work done in a single sentence without using the word "and" you probably have more than one commit's worth of work;
- Could someone conceivably wish to revert some changes in the commit without reverting all of them? If not, again you probably have more than one commit's worth of work.

Importantly these are just rules of thumb. For example you might break the first by stating a few pieces of fixing formatting and spelling errors. The following commit message may well

represent a reasonable commit "Corrected some spelling mistakes in the comments and removed some trailing white space".

It is also possible to commit too often, although this is pretty rare. Generally this will not be penalised, unless the student is clearly abusing the source code control mechanism simply to get around using it properly, for example automating a commit whenever a file is saved.

BitBucket

The code you develop will be automatically tested every week and you will be able to track your progress through an online scoreboard. For this purpose, you will be required to create a Bitbucket account on `https://bitbucket.org` with your university email address and fork the CSLP-16-17 repository.

IMPORTANT: Do not share your code and repository with anyone. Keep your source code secret. Students with identical code snippets will be reported for academic misconduct.

3   Frequently Asked Questions

- *What programming language must I use?*

  The choice of programming language is left largely up to the student. However, there are some restrictions. We must be able to evaluate your program. This means that it must compile and run, unmodified on a DICE machine, without any additional dependencies. Here is an obvious list of languages which should work on DiCE without any problems: C, C++, Python, Java, Haskell, C#, Objective-C, Ruby. However care should be taken with versions. You must also be able to justify your choice for a particular language.

  Note that Informatics gateways (e.g. `student.ssh.inf.ed.ac.uk`) are not configured as DiCE machines, so refrain from testing on these.

- *Can I develop my application on my Windows/MAC/Linux laptop/desktop?*

  Yes. Just make sure that it compiles and/or runs on DiCE as well.

- *Shouldn't we take into account different distributions of disposal events at different locations and changes in the rates triggered by some events (e.g. holidays)?*

  Arguably. In this practical we are analysing snapshots of limited durations and we assume people have on average a similar behaviour. We could potentially evaluate even more complex scenarios by using different inputs and further extending the functionality of the simulator. Nevertheless, the application build according to our requirements will still provide important insights into the system's performance.

- *What about fuel consumption, refilling times, and the waste disposed of at overflowed bins?*

  Although these are some of the many practical aspects one could consider, in your simulations you are not required to include such constraints.
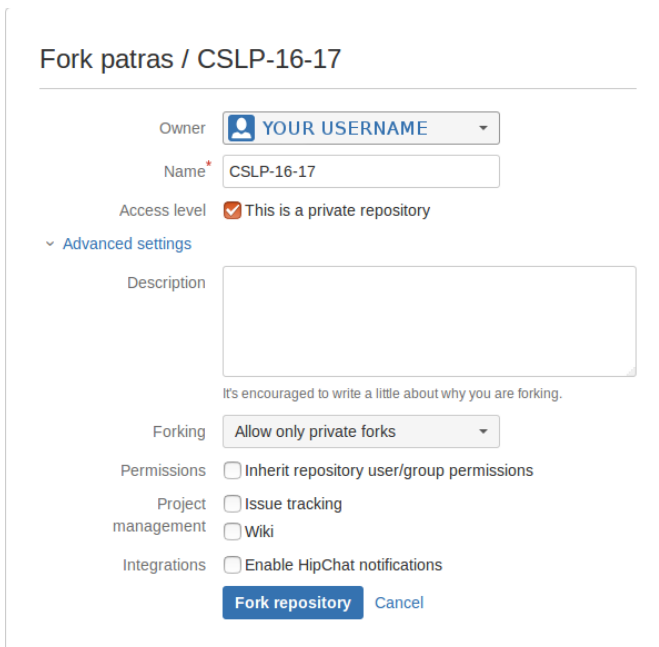
4   Getting Started

The first thing to do is to fork the CSLP-16-17 repository on BitBucket. You will need to have your own copy of this repository. If you use your own machine, make sure to install Git and verify that your code compiles and runs on DiCE.

To fork the repository go to `https://bitbucket.org/patras/cslp-16-17` and in the left hand-side panel, click on "Fork", as shown below

Subsequently you will see a form like the one below. You can name your repository as you wish and add a description. **Remember** to tick "This is a private repository". When ready, click "Fork repository".

Then you must grant the teaching staff read access to your repository. Click on Settings, then go to "Access management", and grant access to the following users:

- Paul Patras (username: patras)
- Philip Ginsbach (username: s1523501)

The web interface should look similar to the image below.

Settings

GENERAL

Repository details

Access management

Branch permissions

Username aliases

Deployment keys

Git LFS `BETA`

Transfer repository

Delete repository

Access management

Users

| Username or email address | Read ▾ | Add |

👤 YOUR USERNAME   owner

Groups

| Select a group ▾ | Read ▾ | Add |

Finally, you will have to clone the forked repository to your local machine. For this purpose, launch a terminal and type:

`$ git clone https://YOUR_USERNAME@bitbucket.org/YOUR_USERNAME/YOUR_REPOSITORY.git`

where `YOUR_USERNAME` must be your BitBucket account name `YOUR_REPOSITORY` must be the name you choose for your fork of the CSLP-16-17 repository.

Part 1

Computer Science Large Practical 2016–2017

Dr Paul Patras

School of Informatics

Issued on: Monday 19th September, 2016

Deadline: Friday 7th October, 2016 at 16:00

1   Introduction

This part of the CSLP is optional and it is primarily intended to allow you to receive useful feedback on your project plan and on the approaches you intend to take to implement the requirements of the practical. This will enable you to understand whether you are on track to complete the second part in time.

Remember that this is an individual project. As such, sharing your design proposal and source code is not permitted. At this stage you should be able to justify in your own words the implementation strategy you will pursue.

2   Description

Part one of the CSLP requires you to submit a *proposal document*, no longer than 3 pages, describing the key building blocks and design choices of the software you are expected to deliver for Parts 2 and 3 of the CSLP assignment. At this stage you are not required to submit any source code. Instead, you should be able to explain how you plan to implement:

1. Handling command line arguments;

2. Parsing and validation of input scripts;

3. Generation, scheduling, and execution of events;

4. Graph manipulation/route planning algorithms;

5. Statistics collection;

6. Experimentation support and results visualisation;

7. Code testing.

This report will prove that you have started the work on the practical and became familiar with the concept of discrete-event stochastic simulation.

In this instance you will not be provided with exhaustive qualitative feedback as this would arrive too late to be useful to you. You are expected to ask for specific feedback on the simulator's main building blocks that you plan to implement. Naturally, you cannot ask for solutions.

3   Submission & Deadline

You are to submit a **PDF** document summarising what you plan to implement, how, and why.

First, create a folder named $\boxed{doc}$ inside your BitBucket repository. Then copy the PDF proposal into that folder and push it to BitBucket. For example, if your proposal document is named $\boxed{proposal.pdf}$ and assuming you copied it to the appropriate folder, then you should submit it using the commands:

```
$  cd doc
$  git add proposal.pdf
$  git commit -m 'Added proposal document'
$  git push
```

The **deadline** for this practical exercise is

$$\boxed{\textbf{Friday 7}^{\textbf{th}} \textbf{ October, 2016 at 16:00}}$$

4   Frequently Asked Questions

- *How long should be my proposal document?*

  There is no minimum page limit, but it should not be longer than 3 pages. The proposal should demonstrate you understood what is required to complete the practical, you have a good plan of how to implement the simulator, and you can justify your choices.

- *How detailed should be my proposal document?*

  Your proposal need not be verbose. You can explain your choices using bullet point lists.

- *I have already begun coding and I want to explain which parts of my application are not working. Can I document this for Part 1?*

  Yes, if you wish to then you can do this. If doing so then please ask precise questions, without explicitly asking for solutions to very specific problems with your code.

- *If I don't complete Part 1 before the deadline, can I still submit the proposal document with Part 2 and receive feedback on it?*

  You may, although this is not advisable. When submitting Part 2 it would already be late to receive useful feedback that could help you develop good code for Part 2, which is marked. You will receive separate feedback on the code you submit for Part 2.

Part 2

Computer Science Large Practical 2016–2017

Dr Paul Patras

School of Informatics

Issued on: Monday 19th September, 2016

Deadline: Friday 11th November, 2016 at 16:00

1 Introduction

This part of the CSLP is an assessed submission, covering a substantial part of the work associated with the practical. This part is summative assessment, worth 50% of the assessment overall. Your mark will be expressed as a percentage given as an integer between 0 and 50. Thus it is not possible to score more than 50 and it is not possible to score less than zero.

Any work you choose to submit should be your own, although you may make use of any part of any publicly-available source code that you find useful. If you do re-use code available on online repositories, clearly state which parts of the code are not your own. Code sharing among colleagues is not allowed.

2 Assessment

Part two of the CSLP requires you to submit your simulator as a preliminary version of the software you are expected to deliver for Part 3 of the assignment. It is understood that at this stage the functionality of your application will be incomplete, but a number of features are expected to be fully working and to have been tested extensively, as detailed next. You will only be assessed on the execution of these features, on which you will receive qualitative feedback through a brief report.

Your application will be tested with the existing example input scripts available on the course, as well as with previously-unseen input scripts, and scripts you have generated yourself.

— ◇ —

Below is a list of the items/functionality expected:

1. A README file should be submitted, explaining your code structure;

2. Valid `compile.sh` and `simulate.sh` scripts must be submitted;

3. The submitted program compiles without errors and warnings;

4. The program displays usage information if no input files are given;

5. The program correctly performs parsing and validation of the input, including

   (a) checking for valid/invalid tokens;
   (b) verifying formats and magnitudes of parameters' values;
   (c) identifying missing parameters and wrong order;
   (d) identifying experimentation directives;[1]
   (e) distinguishing valid but unrealistic inputs and signalling these as warnings (e.g. disposal rate smaller than service rate, lorry capacity smaller than bin capacity, etc.).

6. Your code must generate, schedule, and execute wasted *disposal events* correctly;

7. Your program must produce valid and correctly formatted output;

8. A suite of test input scripts must be submitted, demonstrating the level of testing you performed for Part 2;

9. Your code should be appropriately commented;

10. Your code must exhibit an easy to follow structure and re-usability.

3 Submission & Deadline

You are to push to BitBucket the directory which contains your development project. This should be a working application that can be compiled and executed, and which provides the functionality listed above. If there are any limitations and known issues with your code, briefly describing these in the README file is expected.

At this point you are not required to submit any further documentation, but your code should be commented. A written report is neither required.

Remember to make sure your code compiles and runs on the School of Informatics DiCE computer system. Then you should push the last version of your *working* code to your repository before the deadline.

The **deadline** for this part of the practical is

> **Friday 11th November, 2016 at 16:00**

4 Frequently Asked Questions

- *I haven't managed to implement all parts of the application required for Part 2. Can I still commit what I do have?*

  Yes. You will be marked on your submission as is, and receive feedback on the parts which are working. In addition there will be qualitative feedback which may be worthwhile to you in view of fixing your code before submitting Part 3. Note that it may be difficult to obtain marks for Part 3 if the functionality expected for Part 2 will not be working.

- *I want to explain which parts of my application are not finished. Can I submit documentation along with Part 2?*

  Yes, if you wish to then you can do this. If doing so then please also write this in your README file.

---

[1]At this stage you are not expected to have implemented support for experimentation, but you should be able to correctly parse scripts that will involve experiments.

- *If I don't complete Part 2 before the deadline, can I modify that part in the time before the deadline for Part 3?*

  Yes. You could even discard everything you have done for Part 2 and start again if you wish, though that would generally be inadvisable. Note that you may not be able to submit something meaningful for Part 3 if you do not manage to implement the features initially expected for Part 2.

Part 3

Computer Science Large Practical 2016–2017

Dr Paul Patras

School of Informatics

Issued on: Friday 2nd September, 2016

Deadline: Wed 21st December, 2016 at 16:00

1   Introduction

This is an assessed practical exercise, carrying 50% of the mark for the CSLP. Your mark for this part will be expressed as a percentage given as an integer between 0 and 50. Again it is not possible to score more than 50 and it is not possible to score less than zero.The work which you submit for assessment should be your own although you may make use of any part of any publicly-available source code which you find useful. You should mark clearly the parts of the code that are not your own. This means you are re-using code, which is a good thing and distinguishes it from plagiarism, which is a bad thing.

2   Assessment

Part three of the CSLP requires you to submit the finished version of your code. Your application should be able to simulate the existing example input scripts available on the course web page, **and** it should be able to simulate previously-unseen input scripts, including scripts you have generated yourself.

You will not be evaluated on the code alone. It is essential that you prepare a **written report** explaining your findings from implementing the simulator and the insights gained into the performance of the simulated system. These should be accompanied by plots you produced where appropriate.

— ◇ —

Below is a list of the items/functionality expected:

1. The submitted program must perform route planning and correctly schedule the service of bins;

2. After all simulation events have been output, your application must produce correct summary statistics, following the given output format specification;

3. Experiments should function as described. You should disable detailed output for experimentation;

4. A test suite with some sample area descriptions, which are non-trivially different from each other, must be submitted;

5. Your code must have reasonable run times and there should be evidence of optimisation;

6. Your code should have appropriate indentation and spacing, and must not have unused variables or dead code;

7. There should be evidence of appropriate use of source control with your submission;

8. You should document in a written report the architecture of your simulator, your design choices, and testing efforts. You should also discuss the experiments performed, results obtained, and insights gained.

3   Additional Credit

The requirements listed in the section above illustrate the core functionality which is required from your application. A well engineered solution which addresses all of the above requirements should expect to attract a very good or excellent mark. Additional credit will be awarded for additional useful features which are *not* on the above list. Thus, if you have time remaining before the submission deadline and you have already met all the requirements listed above, then you can attract additional marks by being creative, conceiving of new features which could be added to the application, and implementing these.

If you have added additional features to your implementation in order to attract extra credit then you should be sure to document these. The purpose of this part of the practical exercise is to allow you to exercise your own creativity and deliver a solution which is uniquely your own. Note that the total highest mark you can achieve is still 100, but this additional credit may help if you lose points due to some error found in the way you implemented standard requirements.

Examples of features you might like to consider adding could be the following:

- Allowing other types of distributions for the rubbish bin disposal times.

- Allowing community bins and lorries of different capacities.

- Using multiple lorries per area, sharing the collection demand.

If you include any additional features, first make sure that the simulator will still operate given basic input. You can either design the extra input to be compatible with the existing input, or you can provide a command-line flag to switch between basic and extended functionality and clearly documenting this in your README file.

4   Submission & Deadline

You are to submit the directory which contains your developed code and the written report. Your work will be assessed by compiling and executing your application and based on your report. You must ensure that all source code files needed to compile and run your application are submitted.

Your directory should contain a folder called `doc` where you should put a PDF version of your written report. You should also include an updated README file in your directory, where you should include:

- information about any parts of your application which are not finished or have non-obvious functionality;

- notes on any additional features of your application of which you are duly proud;

- indication of the input scripts you have created for testing purposes (including the location of such scripts).

You would additionally be well-advised to include some text files storing the output of your simulator operating over your test files.

Important: Once again, first make sure your code compiles and runs on the School of Informatics DiCE computer system. Then double check that all the necessary files have been added and committed to your `git` repository and push the final version before the deadline.

The deadline for this practical exercise is

$$\boxed{\textbf{Wed 21}^{\textbf{st}} \textbf{ December, 2016 at 16:00}}$$

5 Frequently Asked Questions

- *How long should be my written report?*

  - There is no minimum page limit. Your report should demonstrate you are familiar with discrete-event stochastic simulations, you can interpret the output, understand the system's performance and you can present your results in a clear and concise manner.

- *How much does the report weight?*

  - The written report carries 20 marks.