Report – CSLP

1. Code structure and Design choices

System overview:

- Written in Python: The reason behind this choice is mainly, that I find the language fun, and I wanted to explore some of its features.

- Databases are not used as I felt in almost all cases, data amounts are not big enough to justify creating a separate database.

- Some of the input test files I have used have been put in folder named “test”. I am not submitting the output files for running test inputs.

Route planning overview:

- Dijkstra algorithm to calculate distances between bins Some of the reasons I chose Dijkstra are:

  - It is relatively fast, compared to other distance calculation algorithms.It can find the distances between an edge and all other edges in `O(ElogV)` where E is number of edges and V is number of vertices .
  - It can be very conveniently used to calculate distances between any two points in a graph. I use Dijkstra to create a hash map containing the distance between every two points if the road layout. While this may seem wasteful at first glance, for most input types it is much faster than calculating only the required distances every time they are needed, because my implementation only runs once, in the beginning pf the simulation.

- Nearest neighbour algorithms for route planning:

  - I use nearest neighbour to direct the lorry to the nearest bin which needs to be emptied, relative to the current location
  - While this may not produce the most optimal route, it is usually close to optimal, and its shortcomings are compensated by the algorithm’s incredible simplicity and speed.

Events generation:

- A priority queue is used for storing the events. This is a predefined module in python, an extension of stacks, allows me to use the time of every event as its priority – the smaller the time, the bigger the priority, this making it perfect for the task.

Code structure:

- I will not go into details of what each class is used for, because they have doc string in which can be easily ready, and I would just repeating myself if I write it here. Instead I will list the classes and outline their main purpose and features here

- Classes:

  - bin - contains the bin status, and functions for adding a bag and empting the bin
  - createObjects – this sets up areas, bins and lorries
  - dijkstra – self-explanatory, this is the Dijkstra algorithm implementation

- lorry – contains the lorry parameters and methods for lorry and bin empting, returning the lorry to depo. For simplicity most of the area statistics are collected here
- myAarser – parser the user input
- runSimulation – the biggest class, this is where most of the simulation takes place. Experimentation is supported by separating the input into parts and running each of the separately. It contains methods for route creation, thrash disposal and first events schedule. Only the first event of each type is scheduled in the beginning of the simulation and after that, every event creates its successor.
- serviceArea – A service area, with the lorry and bins in it. Has method which converts the road layout matrix into bin distance dictionary upon creation, using dijsktra.
- matrixGenerator – Test utility. Creates a square road layout matrix with specified length, max distance between points and chance that there is no road between tow bins

Tests:

Performance:

The Dijkstra algorithm has the biggest impact in performance. Therefore it was possible that big areas with many bins will slow down or even crash the algorithm. To test that I wrote a script which generates matrices of different sizes and started testing performance.

On my machine program runs instantaneously with small inputs (number of bins per area < 100). At around 100 bins per area, the execution takes less than a second. With 200 bins it takes 2-3 seconds. At 500 bins it start to take around a minute, and with 1000 bins it becomes slow - ~ 10 minutes to run.

Changing the number of areas and adding experimentation parameters did not have significant impact on performance.

Overall the algorithms runs relatively fast, and to the best of my knowledge its runtime cannot be noticeably improved without sacrificing heuristic accuracy.

Experiments:

During experimentation some interesting results appeared. Due to the limited time I had by the time I started writing this report I will only list the most important findings:

- Areas too big can interfere with statistics collection – It is possible, if a lorry is assigned too big an area that in never completes a schedule, before the simulation time runs out. To handle that I have added and error message and disabled statistics in such cases
- The most effective ways to improve bin overflow rate (a.k.a.) make it smaller, is to reduce lorry idle time, by increasing service frequency. This however is not effective is service frequency becomes big enough so that the lorry starts becoming late for schedules.
- Disposal distribution rate affects the weight and volume increase in a bin at a constant rate. If with disposal distribution rate 2.0 a lorry needs one trip to empty all bins that have been filled, with disposal distribution rate 4.0, the lorry will need 2 trips, with rate 6.0 it will need three trips, and so on, until the bins overflow rates become big (tested service freq 0.5). Bin overflow rate behaves in a similar fassion.
- Higher bin thresholds increase trip efficiency, but cause bigger overflows.