Delivery Hub Optimiser

Introduction

The Delivery Hub Optimiser program has been designed as per the specification provided for the assignment, with some assumptions changed or removed entirely. Some steps to optimise the process were taken, however some were left out for flexibilities sake, and no doubt some were missed due to no rigorous profiling.

Assumptions

The assumptions made in the creation of this program are as follows:

1. The cost of travel from A → B is proportional to the great circle distance.
2. The cost of travel from A → B is inversely proportional to the population of B.
    a) The demand from a location for goods is proportional to population.
3. Trucks only travel to one location from the hub at a time.
4. Only one hub will serve a location.
    a) It would be desired that hubs service similar numbers of people.

Justification:

(1) The distance to a point introduces a cost of fuel and driver time, both can be roughly approximated to be proportional to the distance travelled. Many other associated costs (i.e. the goods, warehouse costs etc.) are assumed to exist regardless, given the company wishes to operate.

(2) The population of a location is assumed to be proportional to the demand for goods from that location. As such would be proportional to number of trucks needed to service that location per unit time. As such one can see an inverse proportionality to the optimal distance of the hub, derived from (1).

(3) The implementation of the program without this assumption would have been drastically slower, and not feasible given time constraints.

(4) Having multiple hubs serve a location would require a specialised algorithm, and the results yielded with the assumption still yielded results similar to that seen by Amazon's locating of fulfilment centres. This was in part yielded by the assumption (a) which puts a constraint on regions that they should attempt to be of similar population sizes.

Code Design

An MVC architecture was utilised for the program as it suits the nature of the program very well: a set of data to be read, an algorithm or two to be ran, and an output to be displayed. The benefits being both to readability and the ease of changing out, or modifying, components of the code.

Hill Climb Algorithm

The hill climbing algorithm was implemented in a general sense, allowing it to be used with arbitrary fitness calculations provided by a function pointer—arguments, beyond the x and y coordinates in the hill's space, for which can be supplied via a variadic template argument list. The statefulness the variadic template argument list brings could arguably be more eloquently introduced with a callable std::function wrapper, but that would have hampered performance with unnecessary copies.

Lloyd's Algorithm

Lloyd's algorithm was utilised to construct regions in an arbitrary space that separate points into clusters in that space. Like the Hill Climb implementation, Lloyd's algorithm was implemented generally, with the program then utilising it in its specific case with a function pointer for calculating distance between points.

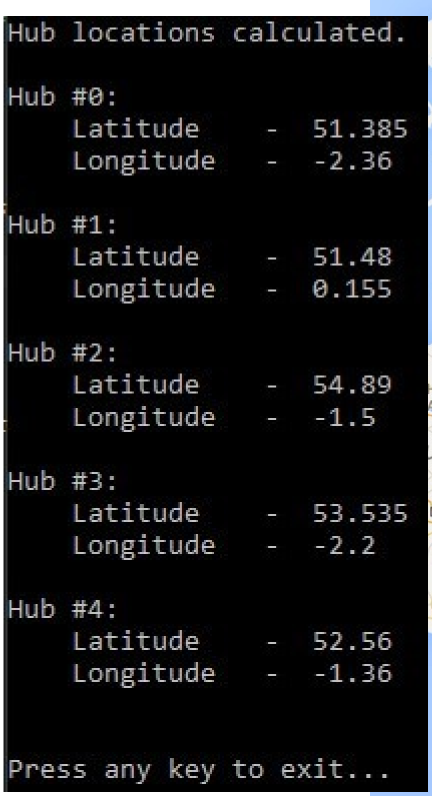
The specific use case of Lloyd's algorithm calculated distance in its space to be a value of:

$$distance = worldDistance * \sqrt{clusterPopulation + locationPopulation}$$

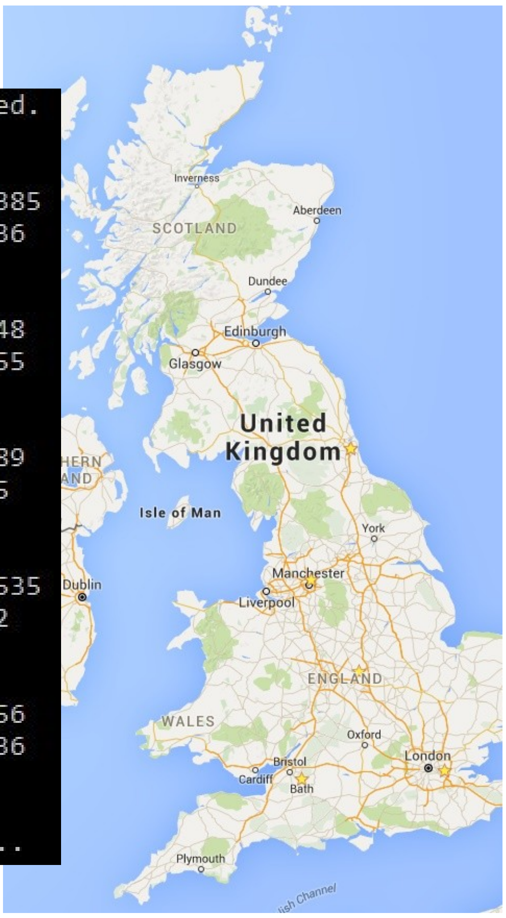
The justification being that it was undesirable for dense locations such as London to be matched up solely spatially with other nearby locations given assumption (4). The factor the physical distance is multiplied by ensures that all points are more likely to favour the clusters that are both near them and smaller over similarly close but much larger clusters. Why square root?

Result

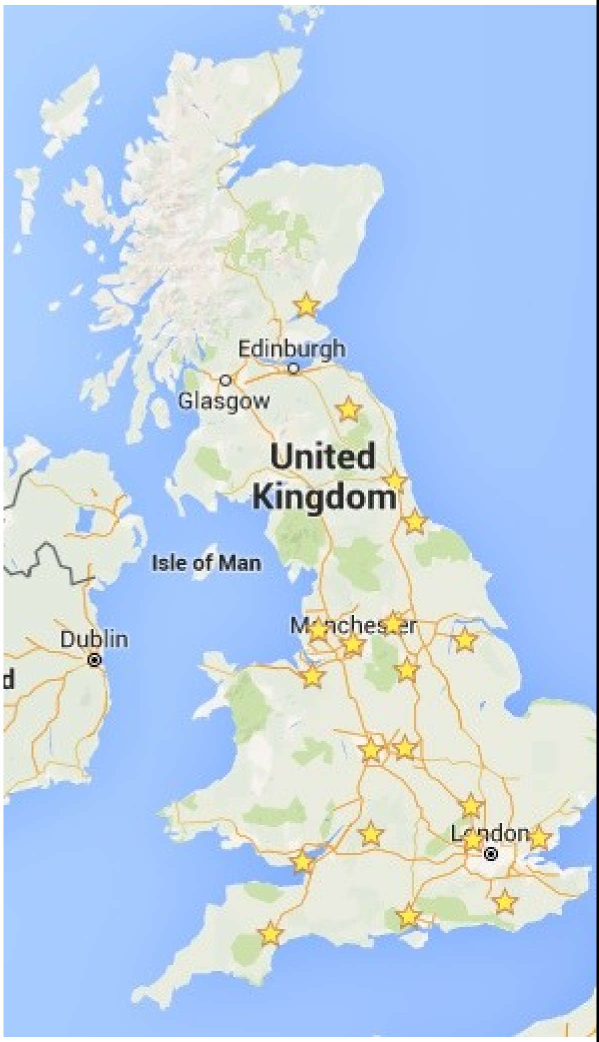
The resulting locations for hubs seem to be very reasonable. For five hubs, the locations are consistently close to the pictured locations (below). These include a location in Washington, in the North East, where Amazon just recently opened up a new hub of their own.

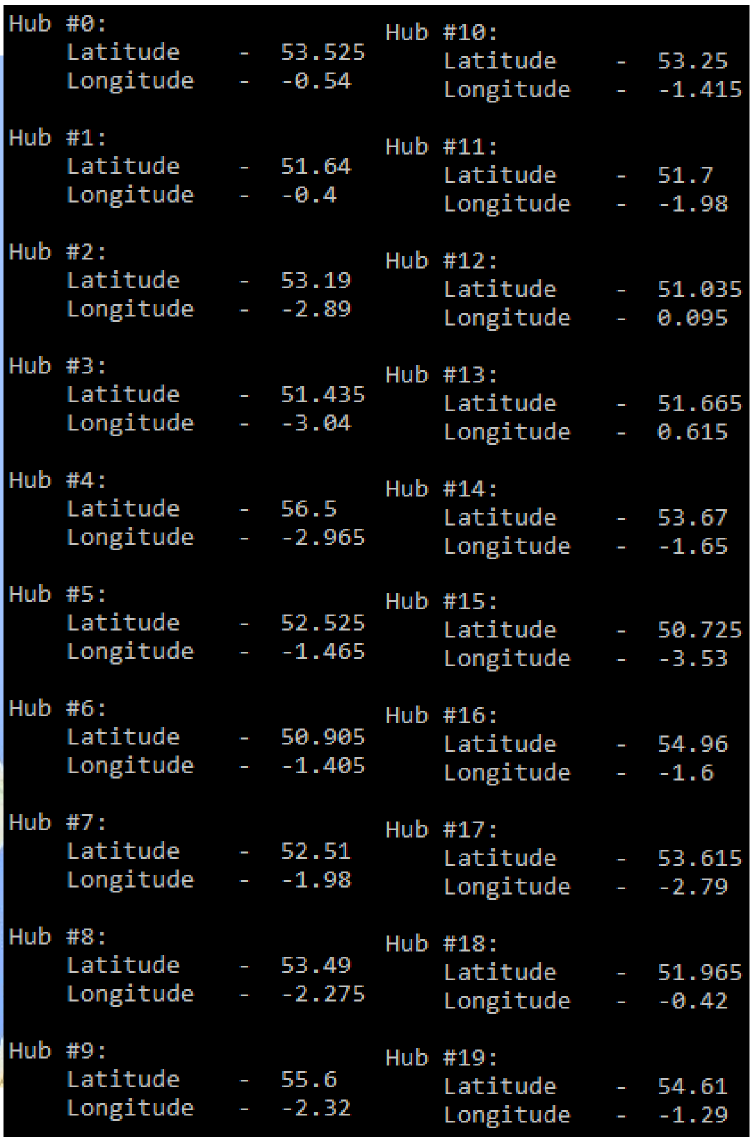


```
Hub locations calculated.

Hub #0:
    Latitude     -   51.385
    Longitude    -   -2.36

Hub #1:
    Latitude     -   51.48
    Longitude    -   0.155

Hub #2:
    Latitude     -   54.89
    Longitude    -   -1.5

Hub #3:
    Latitude     -   53.535
    Longitude    -   -2.2

Hub #4:
    Latitude     -   52.56
    Longitude    -   -1.36


Press any key to exit...
```



Another interesting result was if the hub count was set to a high number (i.e. 20). This yielded clustering similar to that seen by Amazon's locating of its fulfilment centres, as a consequence of population densities (images below).





```
Hub #0:                                 Hub #10:
    Latitude     -  53.525                  Latitude     -  53.25
    Longitude    -  -0.54                   Longitude    -  -1.415

Hub #1:                                 Hub #11:
    Latitude     -  51.64                   Latitude     -  51.7
    Longitude    -  -0.4                    Longitude    -  -1.98

Hub #2:                                 Hub #12:
    Latitude     -  53.19                   Latitude     -  51.035
    Longitude    -  -2.89                   Longitude    -  0.095

Hub #3:                                 Hub #13:
    Latitude     -  51.435                  Latitude     -  51.665
    Longitude    -  -3.04                   Longitude    -  0.615

Hub #4:                                 Hub #14:
    Latitude     -  56.5                    Latitude     -  53.67
    Longitude    -  -2.965                  Longitude    -  -1.65

Hub #5:                                 Hub #15:
    Latitude     -  52.525                  Latitude     -  50.725
    Longitude    -  -1.465                  Longitude    -  -3.53

Hub #6:                                 Hub #16:
    Latitude     -  50.905                  Latitude     -  54.96
    Longitude    -  -1.405                  Longitude    -  -1.6

Hub #7:                                 Hub #17:
    Latitude     -  52.51                   Latitude     -  53.615
    Longitude    -  -1.98                   Longitude    -  -2.79

Hub #8:                                 Hub #18:
    Latitude     -  53.49                   Latitude     -  51.965
    Longitude    -  -2.275                  Longitude    -  -0.42

Hub #9:                                 Hub #19:
    Latitude     -  55.6                    Latitude     -  54.61
    Longitude    -  -2.32                   Longitude    -  -1.29
```

Discussion

There is definitely room for improvement in the above solution. The most obvious that comes to mind is use a more accurate model of physical distance than the great circle distance (i.e. use an algorithm to work out the most optimal route by road between two locations). Another direction for improvement would be to use accurate, specialised models for demand driven by locations – as some rural locations may be less likely to desire deliveries for certain goods than urban locations, and vice versa. One last improvement considered would be to split up locations into sub locations that may be served by different hubs, this could be done at a macro-scale of just population, or in greater detail using an algorithm to determine which streets to best be handled by different hubs.