Tangled Traffic

Agustin Iniguez

November 14, 2021

This documents presents the solution taken to solve the problem Tangled Traffic. First it introduces the objective of the problem, the rules and the desired input data. Then it shows how it solves the problem and presents the results. Finally it briefly discusses the scalability of the algorithm

Objective

Given a certain town with houses, malls and one city center, the objective is to create a road network that minimizes the construction cost. This cost is given by

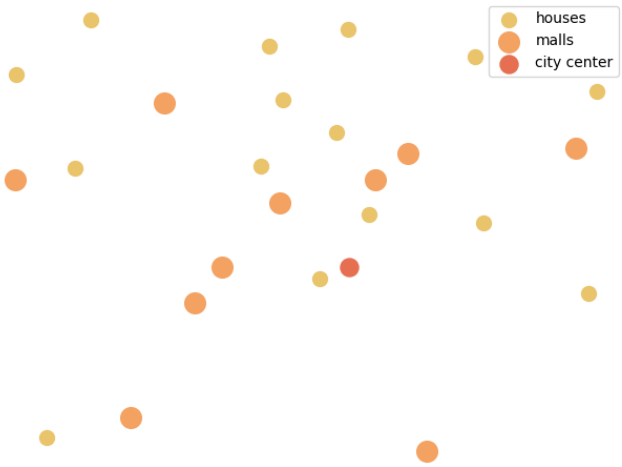
$$c = \alpha * l_l + (1-\alpha) * l_e$$

where c is the cost, l_l is the total length of local roads, l_e is the total length of express roads and α is a constant that varies between 0 and 1 (excluding 0 and 1).

Rules

The town has 3 types of buildings, homes, malls and one city center. All houses should be connected to the road network, together with the city center and at least one mall. The road network can have two types of roads, local and express roads. Local roads departs from every house, express roads connect malls and the city center. The city center is only connected to a mall.

Input data



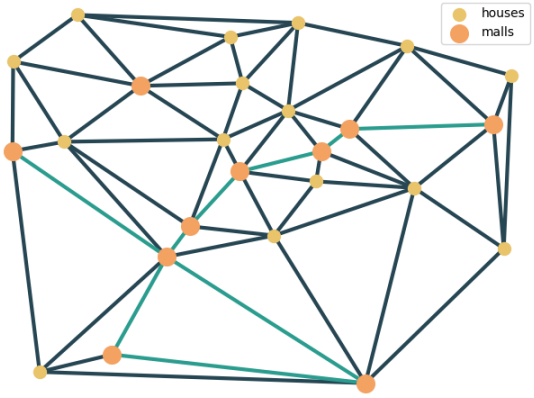
The input data is the location of the houses, malls and city center, together with the value of α. I selected random points that are distributed uniformly in a specific area for houses, malls and the city center. Here is the plot with their locations. This plot is an example that I will use to explain the algorithm.

Solution

To obtain the solution, I need to create a single connected graph. This graph has weighted connections and minimizes the total sum of the weights. The greedy algorithm is to calculate all the distances between all the points, weight them by their cost and take the cheapest connection for each point, making sure that all houses are connected. However, there is a less greedy algorithm. In this alternative algorithm, instead of calculating all distances between each point, a Delauney graph is created. After this, the weights of the connections in this graph are calculated and the cheapest network is obtained using a minimum spanning tree algorithm. Here I explain in more detail the process.

Delauney graph

After obtaining the positions of the houses, malls and city center, I created the Delaunay graph of only the houses and malls. The image shows this graph.



The Delauney graph has the property that the minimum spanning tree is a sub-graph of this graph. Obtaining this graph reduces the number of comparisons between points and reduces the computational time.

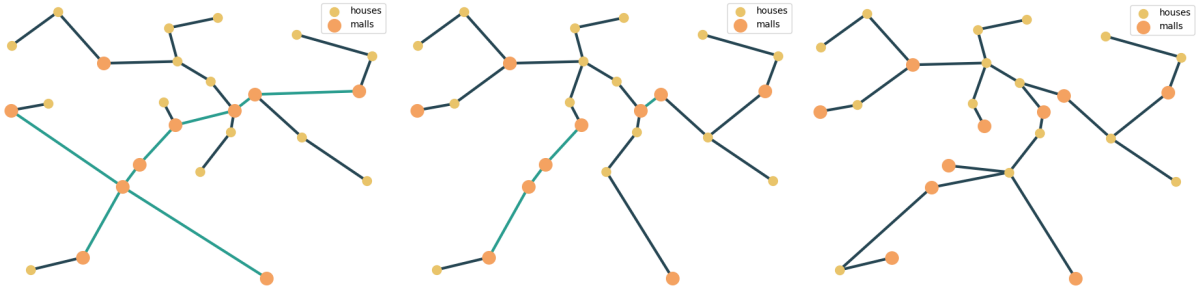
To calculate this graph I used the inbuilt algorithm in the package *scipy*.

Minimum Spanning Tree

After obtaining the Delaunay graph, I run the minimum spanning tree algorithm. This algorithm takes as input the connections between the points and their weights and finds a graph that minimizes the sum of the weights but has all the points connected to a single network.

This algorithm is also already written in the package *scipy*. To show the results, I use 3 different *α*. For each *α* a different minimum spanning tree is created.

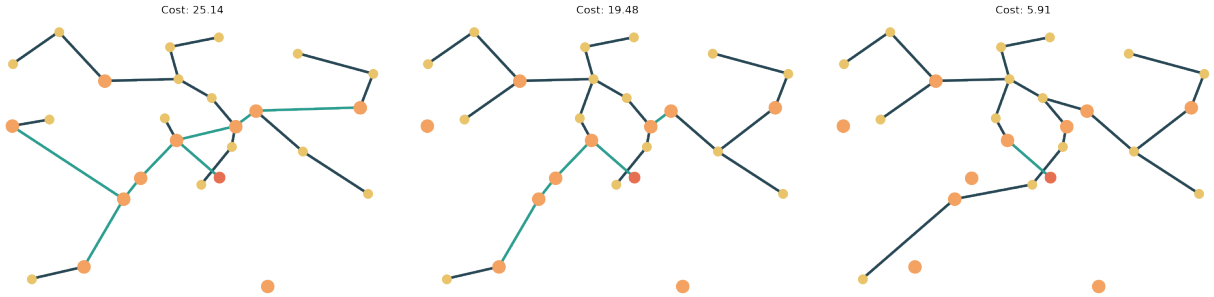
α=0.9 α=0.5 α=0.1



Adding and removing special connections

After the creation of the minimum spanning tree I added the city center. This is connected to its nearest mall. Afterwards I go through all the malls that are connected through a single road and remove them. I iterate this process until there is no more malls with single connections. The results are shown here for the different alphas.

α=0.9 α=0.5 α=0.1



Discussion

One thing to notice from the results is that the road network is different for the different values of α. For example when α=0.9 the network has a lot of express roads since these are the cheapest to build, while when α=0.1 there is only one express road that connects the city center with the mall, as expected. Note that the road that connects the city center is always the same since it is to the closest mall.

Scalability

The greedy algorithm described at the beginning has not a very good scalability since it runs in $O(n^2)$ time (n being the total number of buildings). The algorithm with the Delaunay graph reduces the number of comparisons and it needs only O(n log(n)) time. This is a big improvement. The downside, is that to incorporate the city center, we need to obtained its shortest distance to a mall. This is done calculation m distances (m being the number of malls). In total the proposed algorithm runs in O(m n log(n)) time. This is still almost immediate for values of n around the order of magnitude of 10.