COMP0005
Algorithm
Individual Coursework Report

Student Name: Tianchi Chen
Student Number: 20007914

If you need closer look at those pictures, they are in testdataPics.
All distance are in kilometers while testing for better understanding.

Introduction

In this coursework I delt with the TSP like problems and implemented my data structure to hold data.

Data structure

The data structure I implemented was a graph like data structure, the outer layer is the whole network and it holds a bunch of stations which are inside the network. In the inner layer, every station have 5 values hold inside, which are name, latitude, longitude, relation with other station(the connection of vertexes in graph) and which line it belongs to. This takes all the information the question needs. The Station class also holds several functions like getDistance is in charge of computing the exact distance between two stations, using latitude and longitude. addRelation function is in charge of putting the relation of the stationA and stationB using it's own getDistance function, by this way, when I need to use the distance between two stations, only thin I need to do is to find it out in the dictionary in stead of compute a known relation every single time. The addLineinfo function adds the Belonging line into the station, for security thought, as there might be some stations belong to more then 1 line(that happens a lot in Shanghai, my hometown), "line" is a set. Name, latitude, longitude are strings which are been initialized when the station is created. relation is a dictionary which its keys are the names of stations the have connection with the current station. In it's value's it holds a integer which is the exact distance in kilometer between current station and the connected station. As there is no direction of connection, if two stations are connected we can find them in each other's relation dictionary.

For the outer layer, Network. It's created with a dictionary "stations" with it's keys are name of stations and it's value is the station's class Station structure. It have function addStation which creates a station, this function have parameter: name, latitude, longitude which is exactly the same as what is needed for creating class Station. addLine function is in charge of dealing the relationship and lines which are contained in the londonrailwaylines.csv it takes in stationA, stationB and linename to addRelation and lineinfo to these two stations.

loadStationsAndLines

In this part firstly we create a class Network, then we will start two readers and read the csvfiles, with those csvfiles read in, we shall take away first row which are the titles and use addStation to loop over londonstations.csv to add stations and loop over londonrailwaylines.csv to put in addLine to get relations and lines done. In this part I chose to deal all the calculations on distance on known connections so it may take a bit longer, it will take 0.007secs to finish this part. It is an O(N) design this part. The actual time taken is 0.0066313seconds.

minStops

In this part I started with a BSF algorithm and found out that's pretty slow, as it have to compute every single station in the question, firstly what I did is to add an stopper so it will execute it self when toS appear in the relation keys of station that we are on, by this way some stations are been taken away. There's a queue I created to hold stations that are up to be compute, it's an dictionary which holds the keys are names and the values are number of steps we are up to use. The initialize of this is set the fromS to 0, as It takes 0 km to move. I also created a set seen, it's a set because it don't need to be sorted or anything, it just need to be quick. It holds all the stations that we've seen, it's initialize is also simply add fromS. In order to find out the order of stations we are going, I put an parent dictionary, it will hold the key value sets which keys are names of current station and values are tuples of where the last station is and the distance. Another dictionary minimum is used to hold the minimum distance(stops in this case) that takes to arrive certain station, name is hold in keys and numbers are hold in values. Firstly there will be a loop which loop over all stations to set the minimum of them to a huge number which I used 114514. Then the

If you need closer look at those pictures, they are in testdataPics.
All distance are in kilometers while testing for better understanding.

minimum[fromS] will be set to 0 as same reason as above.

At this stage all initialize is done and there is 1 object in queue which is fromS, firstly it will sort reversly the queue to find the lowest cost to choose but at this stage there's only one so we skip this, and the lowest cost item will popitem from the queue as vertex, and its name will be taken as vertexName. Then nodes will hold all the stations that have relation with the vertex station using the precalculated data in Network and Station. then It will see if the vertex is the toS(this can only happen if the fromS = toS in my algorithm), if it is, then the minimum of vertex will be returned.

Then it will test if it's fromS, if it is, the stops shall be it self as it's not moved yet and don't have a parent, else will plus one from the minimum of station's last station stored in parent. Then it will test if the minimum of vertex is bigger then the stops, this is used to turn the initialize 114514 into actual number. Then it shall test if toS is in the nodes, if there is, it shll return minimum of vertex plus one. Then it will loop over the stations in nodes to see if they are in seen which means if they are been tested before, if it's not been seen yet, it will be given a score equals the current stops, and add this to queue, seen and parent. To improve it's accuracy, from here it will also test if toS is in this station's node if it's in, it will return minimum of station's last station plus 2. After that it shall go for the next loop with new queue.

In this part I tried to make it into an A* algorithm but I found out that if I add the predict distance to the algorithm there is a chance that it's wrong, so it's more like an improved BSF. Some test data:

From	To	Station Tested (out of 653)	Minimum Stops	Time Taken(s)
Baker Street	North Wembley	115	6	0.0010125
Epping	Belsize Park	235	17	0.0024109
Canonbury	Balham	379	10	0.0051166
Vauxhall	Leytonstone	128	6	0.0009707

We can see that most stations are not been tested at all which can improve the execution time. The time complexity is lower then ordinary BFS which is O(V + E). The sorting I used in my algo is tim sort so should be O(NlogN).

minDistance

In this part I developed basically the same thing as I did in minStops, but in this part I'm able to put predicted distance into the score each station have and give a even faster execution time, when implementing the predicted distance, firstly I used the Manhattan distance but after some compare I found out the difference between using Manhattan distance is about the same as using the exact distance that I calculate. So I turn to use exact distance to make it more accurate. It's a bit of time-accuracy exchange. Also the add 1 when detected toS in nodes and add 2 when detected in node's nodes is changed into add the actual distance of those stations.

When returning the answer, I transferred the km I use in calculating to miles as asked in question.

Heres' some test data:

From	To	Station Tested (out of 653)	Minimum Distance (Miles)	Time Taken(s)
Baker Street	North Wembley	61	8.28619868659425	0.0008269
Epping	Belsize Park	85	20.649558536182322	0.0009837
Canonbury	Balham	90	8.840160762655128	0.0010695
Vauxhall	Leytonstone	43	8.844441240066589	0.0004938

If you need closer look at those pictures, they are in testdataPics.

All distance are in kilometers while testing for better understanding.

We can see when I'm using the full version of A* Algorithm, the speed is higher then the half-blood one for minStops, due to less stations tested. The time complexity depends on the graph, as which point we start at, and how many points it connect to. The worst case for this algorithm is when all points in the graph is connected with each other which means in number n iteration it needs to test (all + 1 - n) points

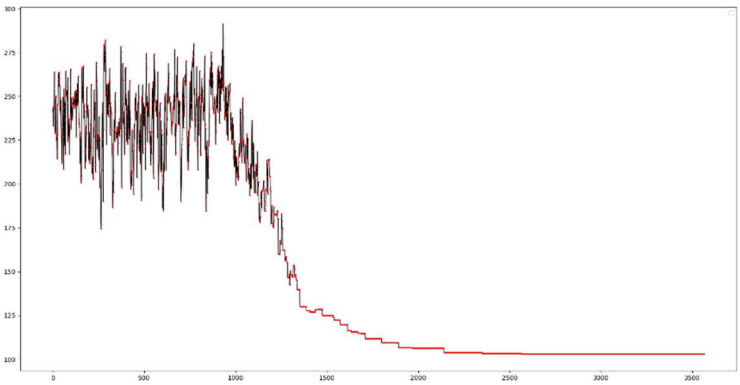
newRailwayline

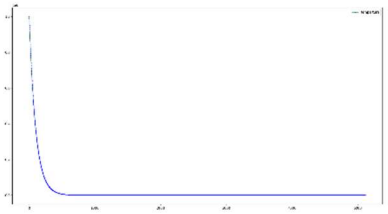
In this part I implemented an sumulate annealing algorithm, it's good for dealing this kind of question where there can be problem of partly best, and when see in total it's not good.

To implement this, 4 factors are needed, temperature, decreaseFactor, numrnage(max round) and sameCounter(number of iteration that result the same). In My algoritm firstly I copied the imputList and shuffle the list – inuse list. As it's been shuffled, it shall have a random start every single time I run it. After that I initialize the distanceA and samecounter to 0. Then the distance of inuse is computed and assign to distanceA. Then it shall generate two not equal integers in range of the length of inuse, those 2 stations with that index shall shift position, and been stored into a new list changelist. Then distance of changelist shall be distanceB. Then it will compare distanceA and distanceB if distanceB is smaller than distanceA, inuse shall be changelist, and distanceA shall be distanceB, as we think that the order in changelist(distanceB) is better then original one. On the other hand, if distanceB is bigger than distanceA, we shall use the equation p = exp((distanceA – disrtanceB)/temperature) and compare it with a random float between 0 and 1, if it's bigger, it shall be taken as well. If something is been taken, the samecounter will be set to 0 again, if not, samecounter += 1. If samecounter is larger than the preset factor number, it shall be ended as the algorithm thinks it haven't change for long enough so it's probably what's best we can get this time, this can save lots of time as the max round set is huge.

When it ends, the inuse set should be assign to outputList and returned as the best result this time. At the end of each iteration, the temperature shall times the decreaseFactor, so that in the next iteration the temperature is lower, so as the number of iteration increase the algorithm have lower possible to take in "worse result". Here's decreasing curves for distance:

Temperature = 100000
decreaseFactor = 0.99
maxround = 500000
sameCount = 1000

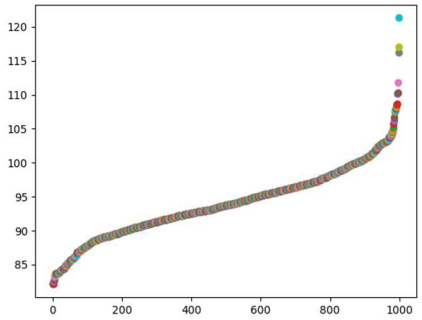




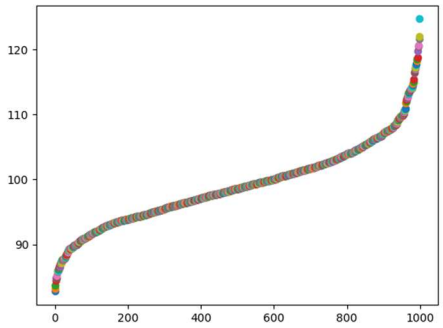
From this we can see that to avoid problem of partly best, when the temperature is high as shown on the blue curve, the line on the distance one(red: point, black: line), is not smooth, it's shifting up and down in order to find the best solution over all, and when the temperature decrease, it becomes harder for it to take in "worse results" and the distance decrease rapidly util it's near the final answer and lots of same results appears.

In order to get a better accuracy-time efficiency balance, we shall tune those factors. To increase accuracy, we shall increase the decreaseFactor and as we increase this, we can also increase the sameCounter.

If you need closer look at those pictures, they are in testdataPics.
All distance are in kilometers while testing for better understanding.

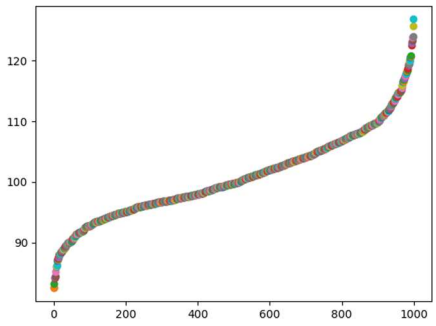


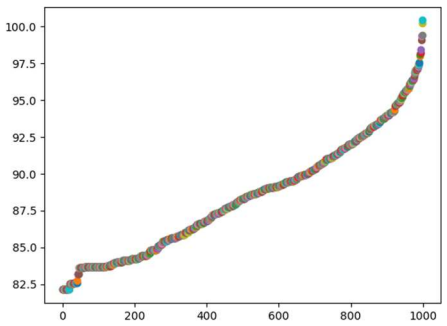
0.99



0.995

0.999





0.999

max: 122.878948663009
min: 82.55387847930878
average 101.13077964598949
0.29s average
3331 rounds average
1000000-1000-0.99

max: 112.85805024202276
min: 84.39243628519462
average 98.24608784043266
0.406s average
5070 rounds
1000-0.995

max: 103.85764007727147
min: 82.16742748384601
average 94.4791614298112
1.335s average
16931 rounds
1000000-1000-0.999

max: 101.0372944919416
min: 82.16742748384601
average 88.74643874861886
13.986s average
172053 rounds
1000000-10000-0.9999

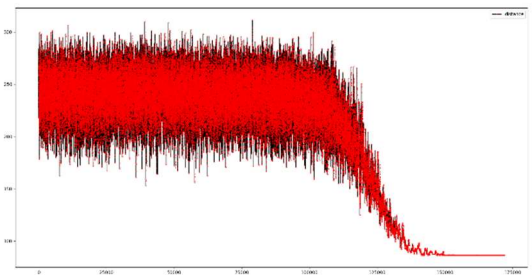
max: 116.49665726708923
min: 83.72312696102743
average 99.54827133728195
0.255s average
3154 rounds average
10000-1000-0.99

max: 118.64413605913128
min: 82.37561422684725
average 98.85587885086863
0.297
4202rounds
10000-1000-0.995

max: 106.94696286737512
min: 82.14076949297349
average 93.83933518624458
0.92s average
11398rounds
10000-1000-0.999

max: 96.6545727061853
min: 82.14076949297349
average 88.34356813540906
9.741s average
125418 rounds
10000-10000-0.9999

From the records I show here we can see that to make it accurate, higher temperature, larger decreaseFactor, bigger sameCount is needed. But as we get too strict with it, the average round to execute increase so that more compute is needed, more time is needed. So at this situation a nice balance is needed, especially decreaseFact, according to the data I collected, I would take 0.995 for an balanced choice(the setting in jupyter note is this version). and if I want most accurate solution, I would take 0.9999 for best effect. Blow is the graph for most accurate one, we see it takes lone time to slowly find out the best result.



If you need closer look at those pictures, they are in testdataPics.
All distance are in kilometers while testing for better understanding.