CST3170

Artificial Intelligence

Coursework One

Travelling Salesman Problem

Edoardo Fratantonio
M00702510

Introduction

This project focuses on using Java algorithms to solve the travelling salesman problem (TSP). The salesman begins his journey in any city and only visits the other cities once. When the journey is finished, he should return to the starting point, completing a full hamilton circuit. This problem is classified as an NP-hard problem. The following algorithms were used in this project:

- Modified Version of Greedy Best First Search. (Heuristic)
- Chained Lin-Khernigan. (Heuristic Optimiser)
- Quick-Sort Dual Pivot. (Divide & Conquer)

Proposed Solution

The use of brute force algorithms to find optimal solutions is well known. However, when it comes to large data sets and cities, the cost and time increase exponentially.

My solution will be using as a first tour a Modified Version of the Best First Search, which is a heuristic greedy algorithm, which looks for the shortest path possible even though by itself does not manage to give optimal results, that is the reason I chose to implement an extension, where I use the “Centroid Formula”, which works out the most centred coordinate among all points, and this will be the starting city.

After running the first tour, the optimisation, which is The Chained Lin-Khernigan, will begin. It is still a heuristic algorithm, but it is recognised as one of the best in order to solve the TSP. It is essentially a hybrid of a 2-Opt, 3-Opt, and K-Opt, as it will try as many combinations as possible in order to achieve an optimal result, by calculating a gain, for the best five cities, or three, depending on how many cities there will be.

[Fig 1.0 Euclidean Distance]

$$d(x,\ y)\ =\ \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

[Centroid Formula]

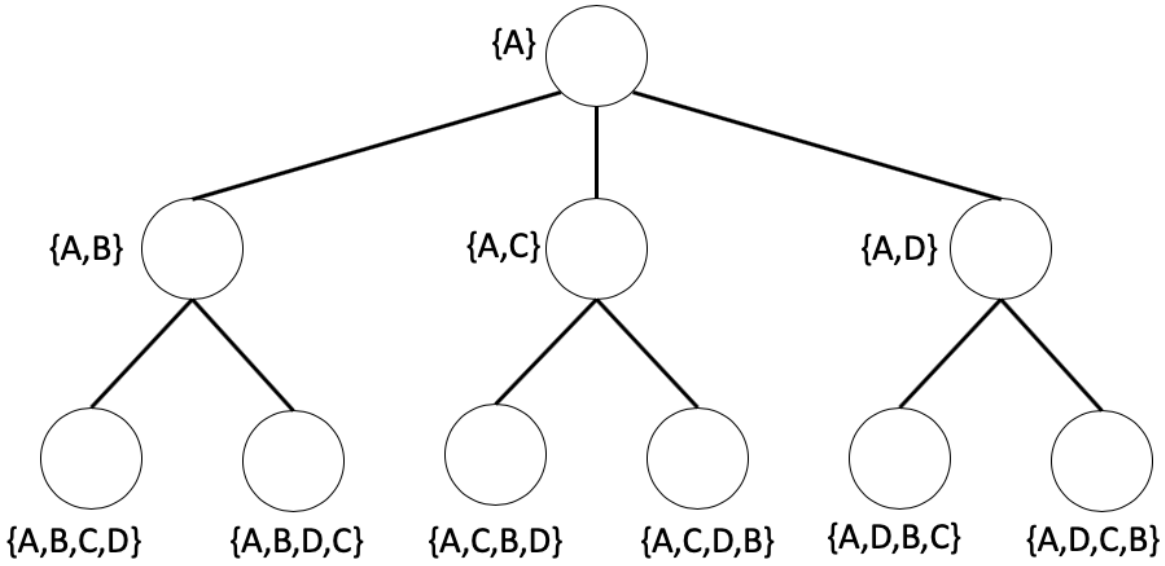
$$Centroid\ =\ \left(X_{mean}\ =\ \frac{\Sigma_i^n\ (x_{i0}\ +\ x_{i1}\ +\ \ldots.\ x_{in})}{n},\ Y_{mean}\ =\ \frac{\Sigma_i^n\ (y_{i0}\ +\ y_{i1}\ +\ \ldots.\ y_{in})}{n}\right)$$

Greedy Best First Search

The greedy best-first search algorithm always chooses the path that appears to be the most convenient at the time. It's the result of combining depth-first and breadth-first search algorithms. It makes use of the heuristic function as well as search. We can combine the benefits of both methods with best-first search.

At each stage, we can select the most promising node using best-first search. In the best initial search algorithm, we expand the nearest node to the goal node, and the closest cost is determined using a heuristic function.[1]

[Fig 1.1 Best First Search]

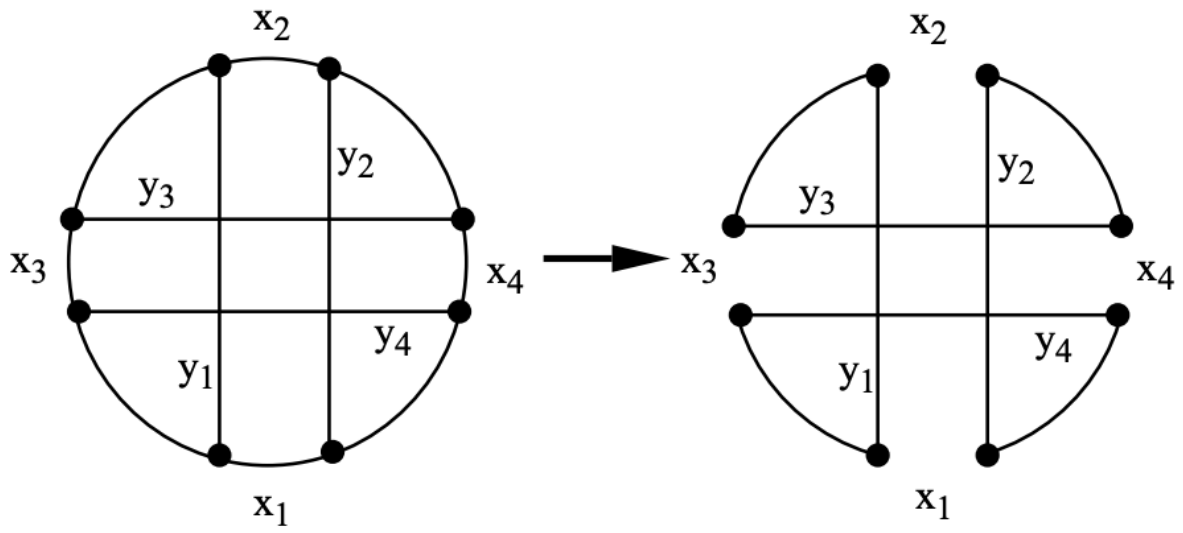


Lin-Kernighan & Chained-LK

Lin–Kernighan is one of the finest methods for solving the symmetric travelling salesman problem. In a nutshell, it entails exchanging two sub-tours to create a new tour. It is a generalisation of 2-opt and 3-opt. 2-opt and 3-opt function by switching two or three edges to shorten the tour. Although LKH may utilise "walk" sequences of 2-Opt, 3-Opt, 4-Opt, 5-Opt, "kicks" to avoid local minima, sensitivity analysis is used to direct and limit the search.[3]

The Chained-Lin-Kernighan takes an existing Lin-Kernighan heuristic-generated tour, alters it by "kicking" it, and then applies the Lin-Kernighan heuristic to it again. If the new tour is shorter than the old one, it keeps it, kicks it, and applies the Lin-Kernighan heuristic again. If the original tour is shorter, it re-kicks it and uses the Lin-Kernighan heuristic, so on and so forth.

[Fig 1.2 Chained-LK]



Program Structure

Main Class

This is the main class, where it will ask the user for a file to run and, if one is found, it will launch the Algorithms.

City Class

This class is where the object of a City is constructed and all the getter methods.

Travel Class

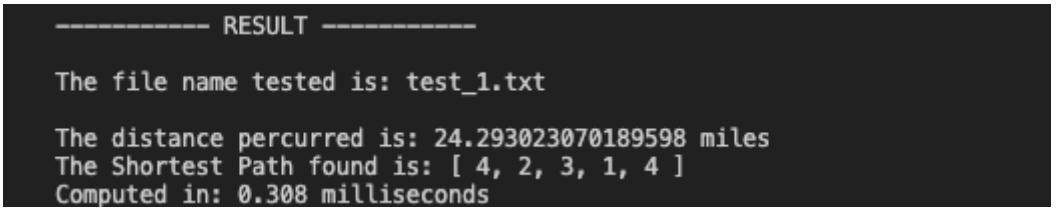
This class has all the algorithms used in order to solve the TSP.

Utils Class

This class contains all the utilities that in this program I needed to use for the Algorithms to run efficiently.

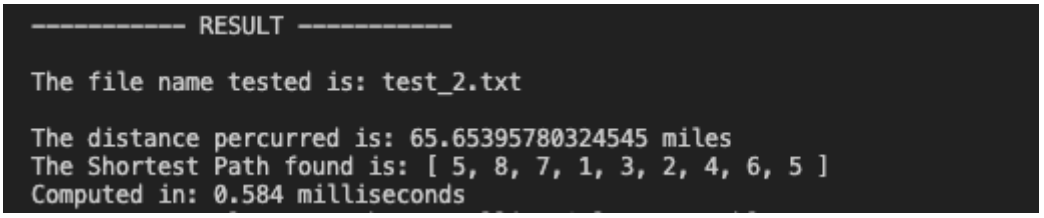
Results

Training Test 1



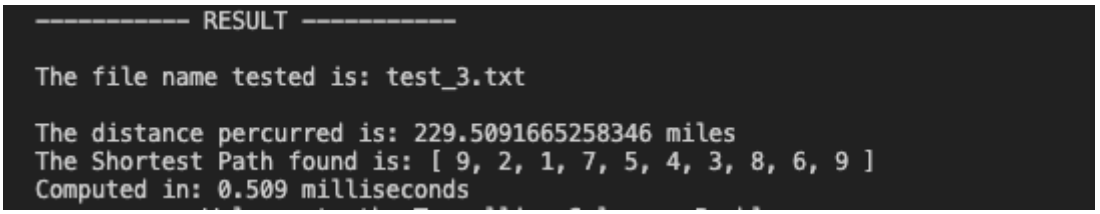
This is the result obtained after running it several times with the submitted code.

Training Test 2



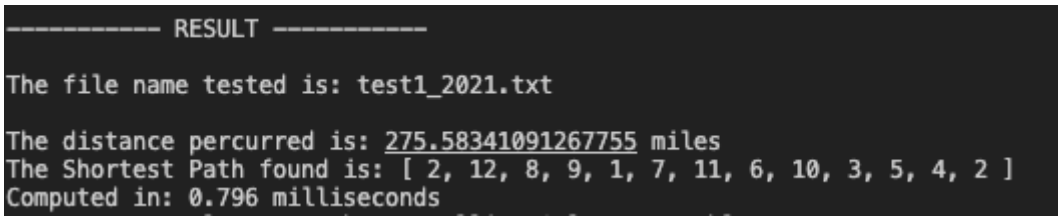
This is the result obtained after running it several times with the submitted code.

Training Test 3



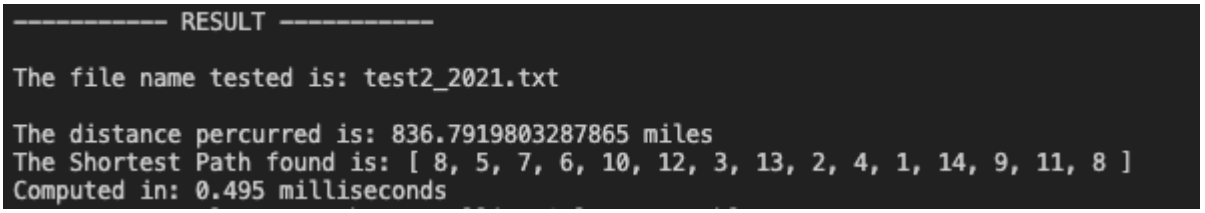
This is the result obtained after running it several times with the submitted code.

Test 1 2021



This is the result obtained after running it several times with the submitted code.

Test 2 2021

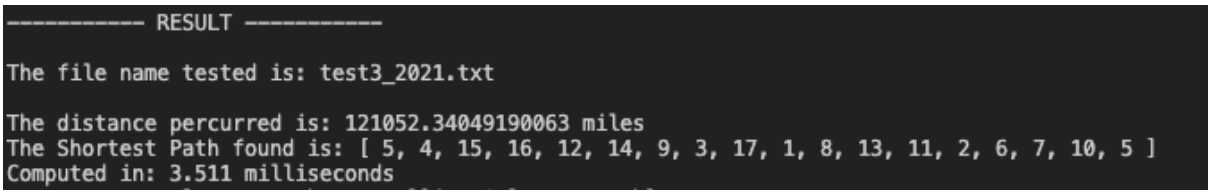


This is the result obtained after running it several times with an adjustment on the "K_OPT" variable from being "3" to "2", as it will lower the chance to overcome the local optimum.

Before Adjustment : 865.5833489870347.

After the Adjustment: 836.7919803287865

Test 3 2021



This is the result obtained after running it several times with an adjustment on the "K_OPT" variable from being "3" to "4", as it will increase the chance to overcome the global optimum.
Before Adjustment : 122176.76152975805.
After the Adjustment: 121052.34049190063.

Test 4 2021

This is the result obtained after running it several times with submitted code.

Self-Marking Sheet

Point	Self	Reason	Area
10	10	I have been very thoughtful about the marks, and I strived for the best.	**Self-Marking Sheet**
10	10	The training tests were repeated multiple times, and the results were the same each time.	**Solve First Training Problem**
10	10	The training tests were run several times and the outputs were all the same, assuming the result obtained was optimal.	**Get Optimal Result for All Three Training Problems.**
10	10	The algorithms used were described as much as they should have had.	**Describe Algorithm(s) Used**
10	9	The code is clean and well commented, however some variables might need to be renamed.	**Quality of Code**
20	19	The tests were repeated several times and also with minor adjustment and getting the same results over and over.	**Get Optimal Results for the First Three Tests**
20	20	The time obtained is under a minute.	**Get Optimal Results for First Three Tests in under a minute**
10	10	The system runs very fast and it might be one of the best results in terms of distance and time.	**Best system on Fourth Test (Path length times time.)**

Bibliography

[1] Informed Search Algorithms In AI - Javatpoint. (n.d.). www.javatpoint.com. https://www.javatpoint.com/ai-informed-search-algorithms.

[2] Lin–Kernighan Heuristic - Wikipedia. (n.d.). Lin–Kernighan heuristic - Wikipedia. https://en.wikipedia.org/wiki/Lin%E2%80%93Kernighan_heuristic#:~:text=In%20combinatorial%20optimization%2C%20Lin%E2%80%93Kernighan,to%20make%20the%20tour%20shorter..

[Fig 1.0] T. (2021, January 11). *Optimising Pairwise Euclidean Distance Calculations Using Python | By TU | Towards Data Science.* Medium. https://towardsdatascience.com/optimising-pairwise-euclidean-distance-calculations-using-python-fc020112c984.

[Fig 1.1 Best First Search] Operations Research: Lesson 19. RULES FOR DRAWING NETWORK ANALYSIS. (n.d.). Operations Research: Lesson 19. RULES FOR DRAWING NETWORK ANALYSIS. http://ecoursesonline.iasri.res.in/mod/resource/view.php?id=90040.

[Fig 1.2 Chained-LK]Helsgaun, K. (2009, January 1). *Figure 3.1 From An Effective Implementation Of the Lin-Kernighan Traveling Salesman Heuristic | Semantic Scholar.* Figure 3.1 from An effective implementation of the Lin-Kernighan traveling salesman heuristic | Semantic Scholar. https://www.semanticscholar.org/paper/An-effective-implementation-of-the-Lin-Kernighan-Helsgaun/e3bda84a829c75864bbeffe99ba90fa876b5ef07/figure/1.