Middlesex University London

CST3170 Artificial Intelligence

TRAVELING SALESMAN PROBLEM

Name: Francesco Arrabito
Student ID: M00696513

Tutor: Chris Huyck

Introduction

This project focuses on utilising Java algorithms to tackle the travelling salesman issue (TSP). The salesperson begins his voyage in any city and only visits the other cities once. When the journey is over, he should return to the starting point, completing a full cycle. Because this is an NP problem, there is no algorithm to solve it in polynomial time (Leena & Amit, 2004). This project made use of the following resources:

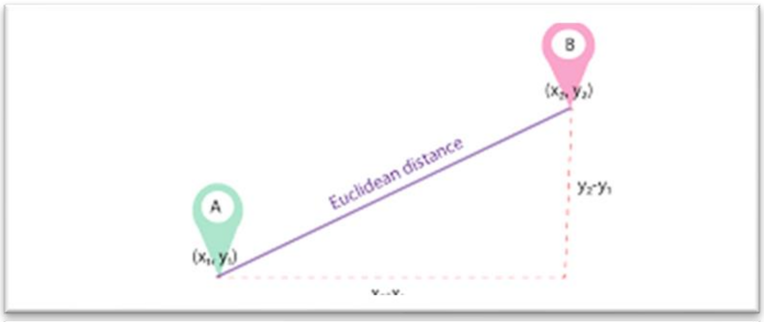
- Kruskal-TSP Greedy (Heuristic)
- Chained Lin-Kernighan (Heuristic)

Problem Definition

- The starting point can start at any given
- A city can only be visited once
- Optimal results need to be produced both in terms of time and distance
- All data sets should run in less than 1 min

Proposed Solution

Brute force approaches are universally acknowledged to get the best outcomes. However, when it comes to large data sets and cities, the cost and time significantly increase (*Lumburovska*, 2018). Furthermore, *Nearest Neighbour* algorithms produce appropriate responses in a relatively short amount of time. However, because the outcome is governed by the starting location, it may not deliver the optimal solution and instead reach the worst case. Because of the non-random nature of the starting point, the alternative provided solution will still be a Greedy algorithm known as **Kruskal-TSP**, which provides superior solutions in terms of path distance compared to the *Nearest Neighbour*. It is consequently enhanced by employing the **Chained Lin-Kernighan** Algorithms (CLK) to produce route improvements over time by lowering the crossover rate (Fig. 4).



The Euclidean distance formula is used to compute the distance between the cities:

Figure 1 - Visual Representation of Euclidean Distance

$$D_e = \left(\sum_{i=1}^{n} (p_i - q_i)^2 \right)^{1/2}$$

Where:
n = number of dimensions
pi, qi = data points

Figure 2 - Euclidean Distance Formula

Kruskal-TSP (Local tour)

ptwiddle.github.io = https://bit.ly/3lLXX3m

The provided Kruskal-TSP approach works in the same way as classical Kruskal in that it always selects the shortest link that does not produce a loop. Kruskal produces a spanning forest (a collection of spanning trees), but our Kruskal-TSP produces a collection of chains (sub-tours). The only limitation we must apply is that the merges must be conducted using the end points of the two chains. This reduces the number of merging alternatives and produces a suboptimal spanning tree, but being chain also gives a solution to the TSP.

Initially, each node is assigned to its own set. The connections are ordered and processed from shortest to longest. Every link between two distinct sets is merged, preventing cycles and forks.
Merges are carried out until only one set is left. Efficient implementation can achieve O(N2) time complexity, where N is the number of nodes. The benefit is that the number of options for the next merging is greatly decreased, but the disadvantage is that optimality is no longer guaranteed.

The Kruskal-TSP algorithm is similar to single link agglomerative clustering, except that the criteria must employ just the endpoints for the linkage. The advantage is that the merging is easier to accomplish than with a single connection. The disadvantage is that it does not ensure optimality. Early merges, like agglomerative clustering, cannot be reversed and might result in suboptimal decisions later.
An example of the method is displayed in Fig. 3, which depicts chosen steps of the process.

The sixth merge is a locally optimal decision at the moment, but it is not part of the ultimate solution. It will result in a less-than-ideal decision for the 15th merging. Most failures occur in the later stages of the procedure when there are fewer candidate pairs to be merged.

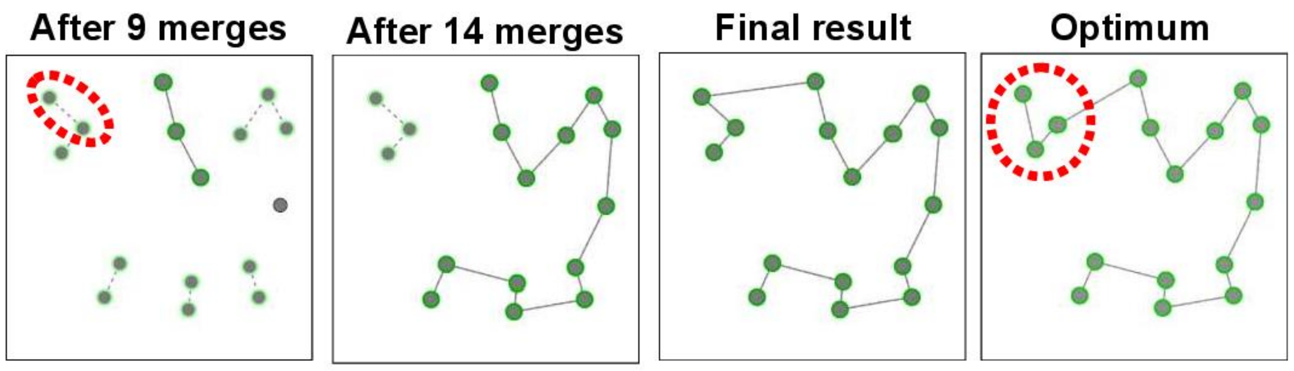


Figure 3 - Example of the Kruskal-TSP algorithm with minor difference on the top-left corner

Chained Lin-Kernighan (Optimiser)

stemlounge.com = https://bit.ly/3lHbiKj

Lin-Kernigan (LKH)

Lin–Kernighan is an optimised k-Opt tour-improvement heuristic, and it is one of the best combinatorial optimization approaches for solving the symmetric travelling salesman problem. In a nutshell, it takes an existing tour and attempts to enhance it by exchanging pairs of sub-tours to produce a new tour. It's a mix of two and three options. The 2-opt and 3-opt functions reduce the route by switching two or three edges. Lin–Kernighan is adaptive, choosing at each stage how many routes between cities must be switched in order to find quicker travel.

Implementations of the Lin-Kernighan heuristic, such as Keld Helsgaun's LKH, may employ "walk" sequences of 2-Opt, 3-Opt, 4-Opt, and 5-Opt, "kicks" to avoid local minima, sensitivity analysis to guide and constrain the search, and other techniques.

LKH features two versions: the original and the later-released LKH-2. Although it is a heuristic rather than a precise technique, it typically yields ideal results. It has found the optimal route for every trip with a known optimal length. It has also established records for any issue with unknown optimums, such as the World TSP, which has 1,900,000 sites, at some point in time.

Chained Lin-Kernighan (CLK)

Chained Lin-Kernighan is a tour optimization approach based on the Lin-Kernighan heuristic that takes an existing Lin-Kernighan heuristic-generated tour, alters it by "kicking," and then applies the Lin-Kernighan heuristic to it again. If the new tour is shorter, it keeps it, kicks it, and uses the Lin-Kernighan heuristic once again. If the original tour is shorter, the Lin-Kernighan heuristic is applied and the previous tour is kicked again.

In this solution, it terminates when there are no further improvements, rather than when it reaches a time restriction or a tour of a certain duration, and so on.

Because it is a heuristic, it does not solve the TSP optimally. It is, nevertheless, a subroutine utilised as part of the precise solution technique for the cutting-edge Concorde TSP solver.

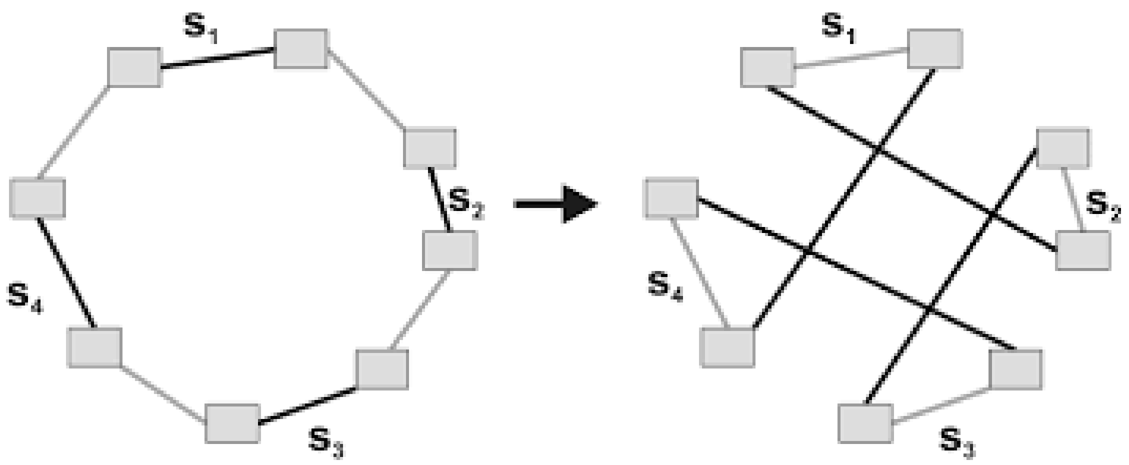


Figure 4 - Example of a double bridge move on a 3-optimal tour.

Program Structure

Main class
This class starts the ask the user to enter a file name via terminal.

Cities class
This class starts the timer, the file reader (generating all the City objects) and creates an Algo object containing the final solution.

City class
This class provides methods and variables that handle operations like “settingNeighbor”, “replacingNeighbor”, and relevant methods for dealing with cities.

Algo class
This class is a algorithms container which generates the final solution.

Score class
This class is a simple and small scores container.

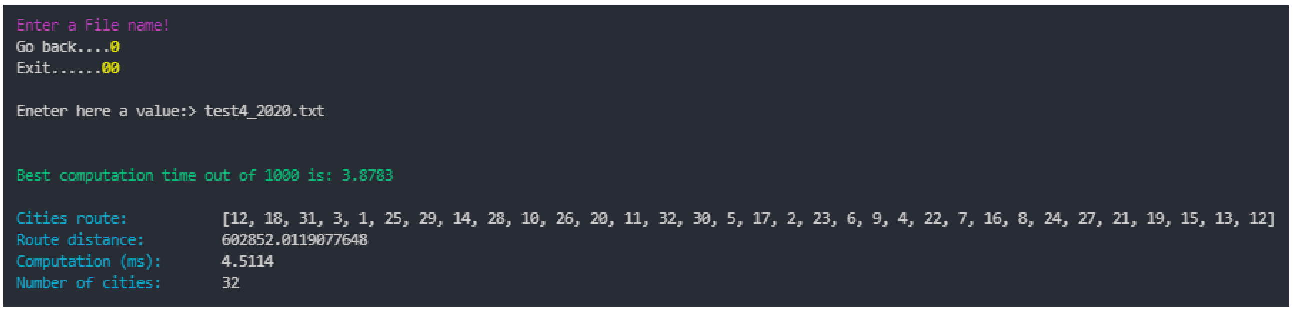
Util class (personal library)
This class is a collection of various tools, many of which are utilised in this project.

Self-Marking Sheet

Point	Self	Self	Area
10	10	A fair and critical marks allocation.	Self-Marking Sheet
10	10	The test files were run numerous times, and the output was the same each time.	Solve First Training Problem.
10	10	The test files were ran several times, and while the result distance was the same, the routes were not.	Get Optimal Result for All Three Training Problems.
10	10	The algorithm was discussed, and the methods utilised were examples to show how they function.	Describe Algorithm(s) Used.
10	8	The code is clean and well organised.	Quality of Code
20	20	The files were ran numerous times, and the distances were comparable; only the paths differed.	Get Optimal Results for the First Three Tests.
20	20	The files were ran numerous times, and the outcome was comparable in terms of distance, except that the paths changed and it took under one minute.	Get Optimal Results for First Three Tests in under a minute.
10	10	The outcome is optimal, as is the computation time.	Best system on Fourth Test (Path length times time).

TEST

A benchmark of the software carried out with 32 cities, shown in the picture below:



```
Enter a File name!
Go back....0
Exit......00

Eneter here a value:> test4_2020.txt


Best computation time out of 1000 is: 3.8783

Cities route:          [12, 18, 31, 3, 1, 25, 29, 14, 28, 10, 26, 20, 11, 32, 30, 5, 17, 2, 23, 6, 9, 4, 22, 7, 16, 8, 24, 27, 21, 19, 15, 13, 12]
Route distance:        602852.0119077648
Computation (ms):      4.5114
Number of cities:      32
```

Testing machine characteristics:

CPU	= Intel(R) Core(TM) i7-3632QM CPU @ 2.20GHz 2.20 GHz.
Physical processors	= 1.
Cores	= 4.
Logic processors	= 8.
CacheL1	= 256KB.
CacheL2	= 1,0 MB.
ChaceL3	= 6,0 MB.
RAM	= 8,00 GB (7,88 GB utilizzabile).