Middlesex University London

CST3170 - Artificial Intelligence

TRAVELING SALESMAN PROBLEM

Name: Francesco Arrabito
Student ID: M00696513

Tutor: Chris Huyck

Introduction

The purpose of this project is to solve the travelling salesman problem (TSP) employing algorithms in Java language. The salesperson begins on his journey in any location and only visits the other cities once. When the voyage is over, he should return to the beginning, completing a full circle. There is no method that can solve this NP issue in polynomial time since it is an NP problem (Leena & Amit, 2004). The following resources were used in this project:

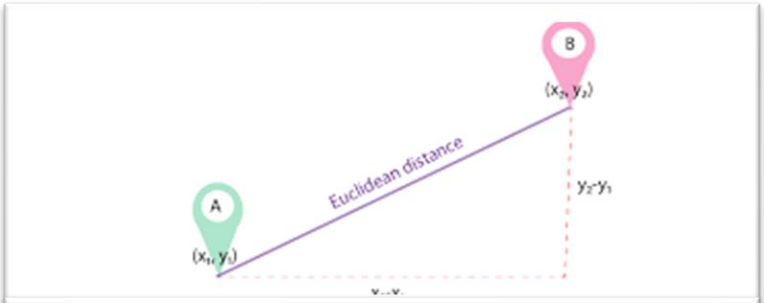
- **Kruskal-TSP Greedy** (Heuristic)
- **Chained Lin-Kernighan** (Heuristic)
- **Quick-Sort** (Divide & conquer)

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

The employment of brute force methods is commonly known to produce the greatest results. However, when it comes to large data sets and cities, the cost and time significantly increase (*Lumburovska*, 2018). Heuristic algorithms, on the other hand, such as *Nearest Neighbour*, produce acceptable results in a relatively short amount of time. Nevertheless, because the output dipends on the starting point, it may not produce the optimal result and instead yield the worst case scenario. Alternatively, a solution which does not involve a random starting point, is a Greedy algorithm known as ***Kruskal-TSP***, that delivers superior solutions in terms of path distance when compared to the Nearest Neighbour. It is consequently improved by using the ***Chained Lin-Kernighan*** Algorithms (CLK) to provide route improvements over time by decreasing 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. While the original Kruskal produces a spanning forest (a collection of spanning trees), the *Kruskal-TSP* generate a collection of chains (sub-tours). The only limitation 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.

To summarise, the Kruskal-TSP algorithm is similar to single-link agglomerative clustering, except that the criteria must use only the endpoints to link, with the advantage, as previously stated, that merging is easier than a single connection and the disadvantage that it does not guarantee 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.

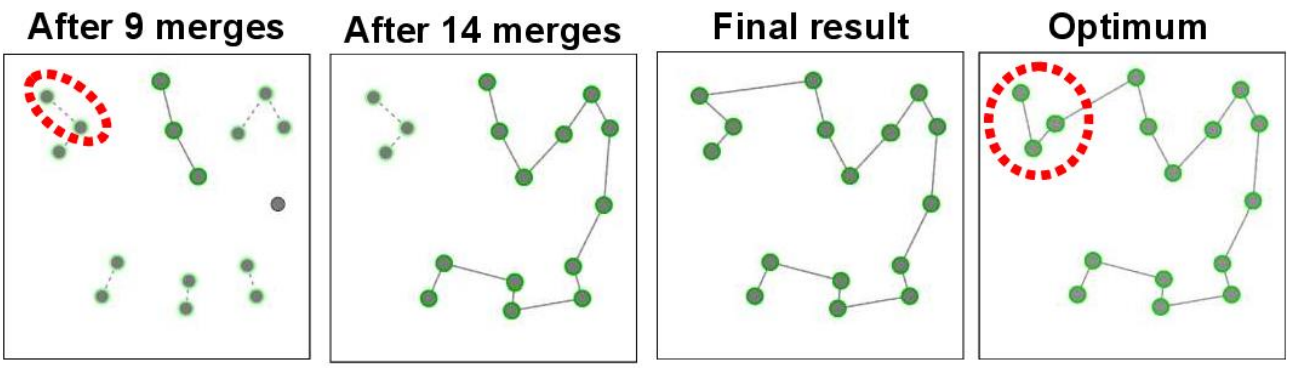


Figure 3 - Example of the Kruskal-TSP algorithm with minor difference on the top-left corner (https://bit.ly/3yeks5P)

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 Opt. 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.

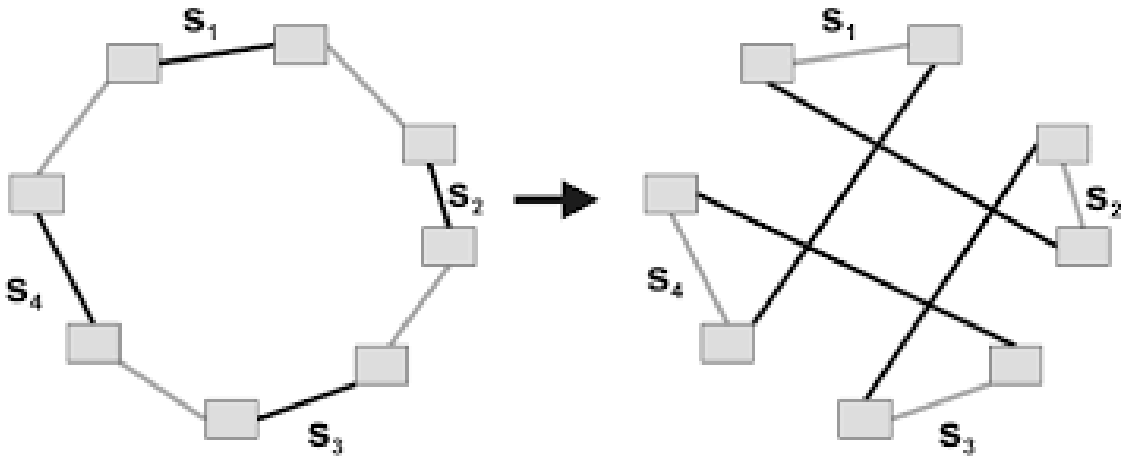


Figure 4 - Example of a double bridge move on a 3-optimal tour (https://bit.ly/3IDURbm).

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: Kruskal-TSP, Chained Lin-Kernighan and Quick-Sort (mono pivot).

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 1000 times. The distance and route output were the same each time.	Solve First Training Problem.
10	**10**	The test files were run 1000 times. The distance and route output were the same each time.	Get Optimal Result for All Three Training Problems.
10	**10**	The algorithms were discussed, and some method mentioned.	Describe Algorithm(s) Used.
10	**8**	The code is clean and well organised.	Quality of Code
20	**19**	The test files were run 1000 times. The distance and route output were the same each time.	Get Optimal Results for the First Three Tests.
20	**20**	The test files were run a total of 1000 times. Every time, the mileage and route output were the same. The timing (ms) improved marginally by each try.	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).

TESTS

The benchmarks for the following tests were determined by taking the best time (ms) from 1000 attempts using the same file, distance result and route.

Training files

```
Enter here a value:> training_1.txt


Best computation time out of 1000 is: 0.2651

Cities route:           [1, 3, 2, 4, 1]
Route distance:         24.293023070189598
Computation (ms):       0.2737
Number of cities:       4
```

```
Enter here a value:> training_2.txt


Best computation time out of 1000 is: 0.5839

Cities route:           [6, 4, 2, 3, 1, 7, 8, 5, 6]
Route distance:         65.65395780324546
Computation (ms):       0.6197
Number of cities:       8
```

```
Enter here a value:> training_3.txt


Best computation time out of 1000 is: 0.6786

Cities route:           [6, 8, 3, 4, 5, 7, 1, 2, 9, 6]
Route distance:         229.5091665258346
Computation (ms):       0.950501
Number of cities:       9
```

Tests files

```
Enter here a value:> test1_2021.txt


Best computation time out of 1000 is: 0.818899

Cities route:           [2, 12, 8, 9, 1, 7, 11, 6, 10, 3, 5, 4, 2]
Route distance:         275.58341091267755
Computation (ms):       0.8333
Number of cities:       12
```

```
Enter here a value:> test2_2021.txt


Best computation time out of 1000 is: 1.066499

Cities route:           [9, 14, 1, 4, 2, 13, 3, 12, 10, 6, 7, 5, 8, 11, 9]
Route distance:         836.7919803287864
Computation (ms):       1.087
Number of cities:       14
```

Optimal results for Test 3 were reached by setting the *Kruscal* (first tour) algorithm in reverse mode
Before the adjustment: 122176.76152975805**(mi)** – After the adjustment:121052.34049190063**(mi)**

```
Enter here a value:> test3_2021.txt


Best computation time out of 1000 is: 1.4292

Cities route:           [8, 1, 17, 3, 9, 14, 12, 16, 15, 4, 5, 10, 7, 6, 2, 11, 13, 8]
Route distance:         121052.34049190063
Computation (ms):       1.5333
Number of cities:       17
```

Optimal results for Test 4 were reached by setting the *Kruscal* (first tour) algorithm in reverse mode
Before the adjustment: 2072358.6799312572**(mi)** – After the adjustment:2065652.1356549377**(mi)**

```
Enter here a value:> test4_2021.txt


Best computation time out of 1000 is: 2.702799

Cities route:           [11, 8, 16, 9, 27, 3, 19, 13, 5, 21, 26, 6, 7, 23, 12, 17, 20, 22, 1, 15, 10, 14, 25, 24, 18, 4, 2, 11]
Route distance:         2065652.1356549377
Computation (ms):       2.7811
Number of cities:       27
```

TESTING ENVIRONMENT

Testing machine characteristics:

CPU	: Intel(R) Core(TM) i7-3632QM CPU @ 2.20GHz 2.20 GHz.
System architecture	: 64 bit, x64-based processor.
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 available).

Operative System:

Operative-System	: Windows.
Edition	: 10 Pro.
Version	: 21H1
Build	: 19043.1348