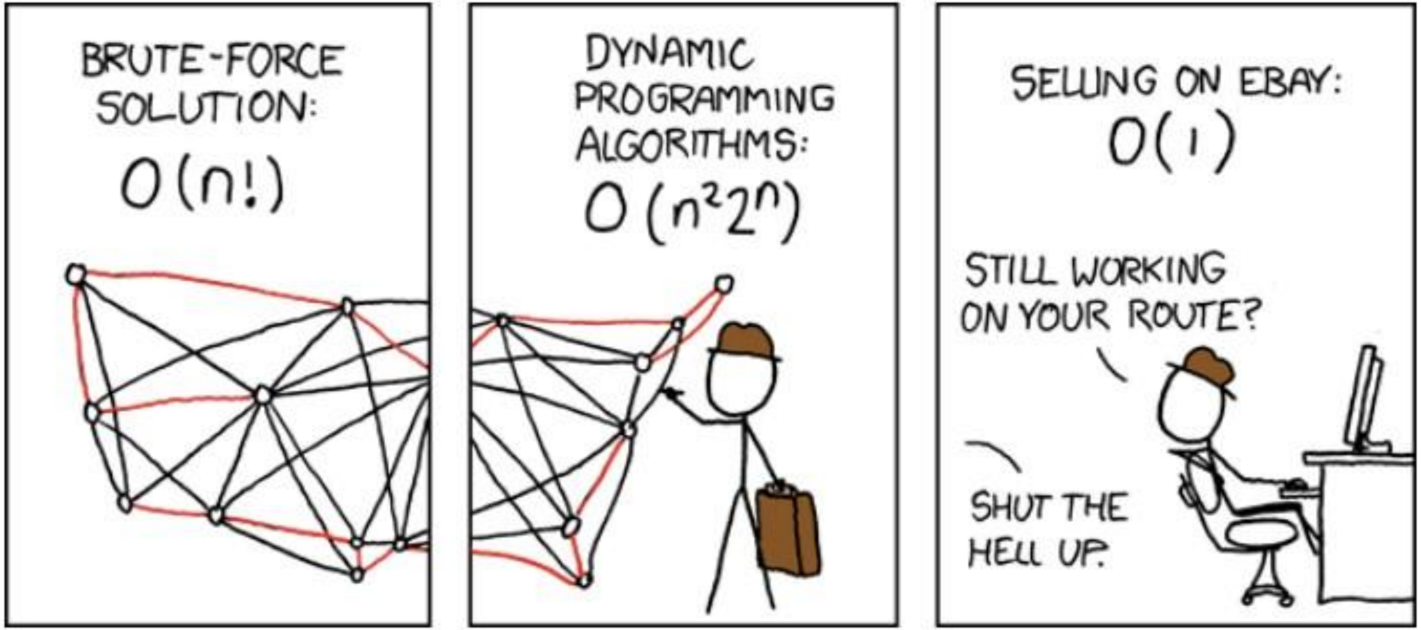
Davide Teixeira
up202109860

Class: 2LEIC05
Group: 2

Ana Ramos
up201904969

Travelling Salesperson Problem



Source: https://xkcd.com/399/

Datasets reading and parsing

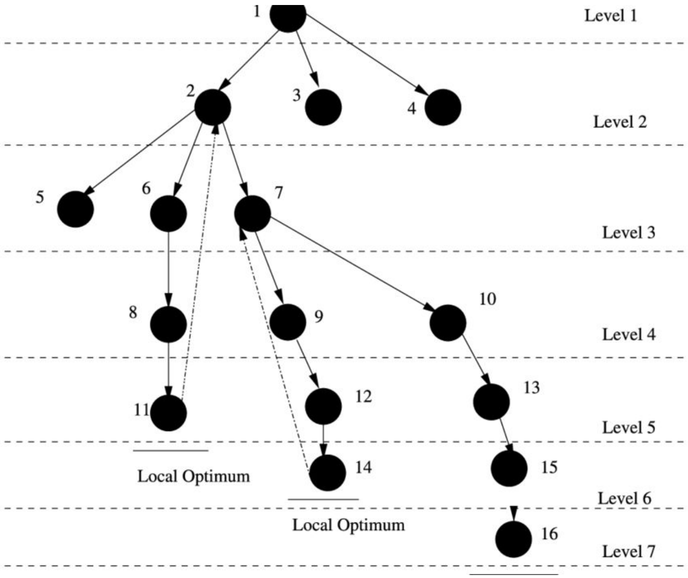
- **C++ streams:** Input file streams to read data from files
- Parsing line-by-line with **tokenization** (getline() and istringstream)
- **Data conversion** from string to integer
- **Graph and Map Creation** - create the graph and a hash table to map the vertex index with the geographical coordinates

Graphs Utilized

- It was used a single graph to represent the whole TSP problem
- The graph data structure that was used was the one that was given to us during the practical classes
- Some changes were made to the graph in order to make it more efficient
    - Replacement vector -> unordered map - In order to improve the functions findVertex and findEdge (O(n) -> O(1)), therefore making the whole problems more efficient

Implemented Functionalities

- **Backtracking**: O(V!)
- Brute force approach: Always gives the optimal solution



source

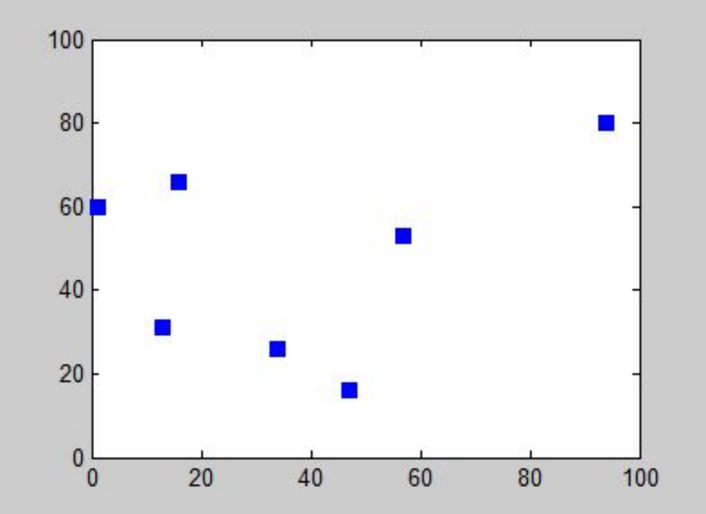
Implemented Functionalities

- **Triangular Inequality Algorithm**: O((V+E))
- It was implemented the triangular inequality algorithm that follows these steps:
  - 1. Prim's algorithm to get a MCST
  - 2. Double the edges to get an Eulerian graph
  - 3. DFS to get a preorder traversal of the MCST
  - 4. Sum the output

Source: https://www14.in.tum.de/personen/khan/Arindam%20Khan_files/2.%20metric%20TSP.pdf

Implemented Functionalities

- **Nearest Neighbour Algorithm: O(V+E)**
- Calculates, at each vertex, the closest vertex that was not visited yet
- Chosen by its efficiency



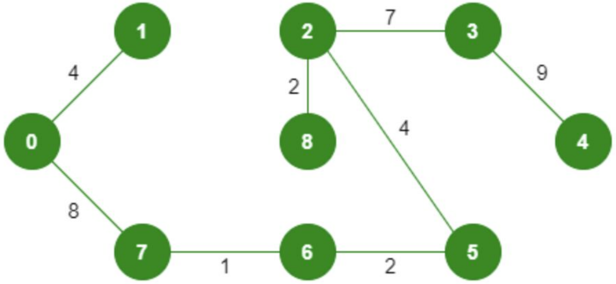
source

Implemented Functionalities

- **2-opt algorithm: O(V^2)**
- In order to increase the nearest neighbors accuracy, a 2-opt algorithm was used.
- It is based on a hill climbing algorithm, where at each state, the algorithm tries to find its neighbour states and does this iteratively until it reaches a local minimum or until the maximum number of iterations has been reached.

Implemented Functionalities

- Real-world TSP
- Used all of the algorithms, except the triangular inequality, since it might produce wrong results for non fully-connected graphs
- In the example below, the algorithm doesn't work if there isn't a direct connection from vertex 8 to vertex 3.



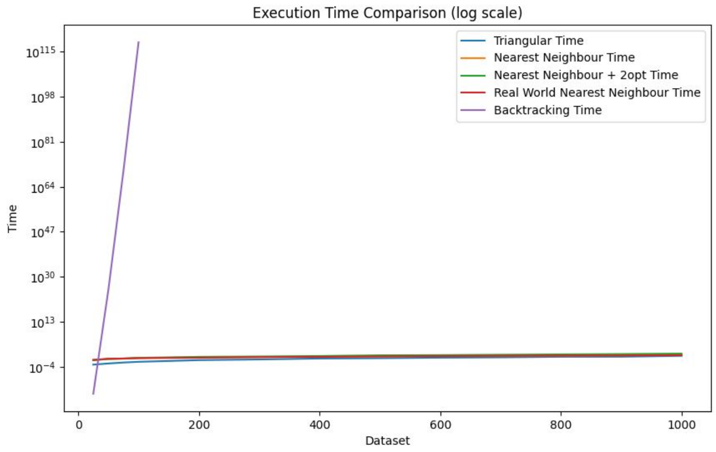
source

Algorithm Analysis

Backtracking algorithm:

- Its time complexity is factorial, so it is expected to not be able to execute for small graphs (for instance, a graph with 20 vertices is enough for a personal computer to not be able to compute it).
- So, the output was done by applying a formula that predicts the execution time of it based on the toy-graphs execution time.

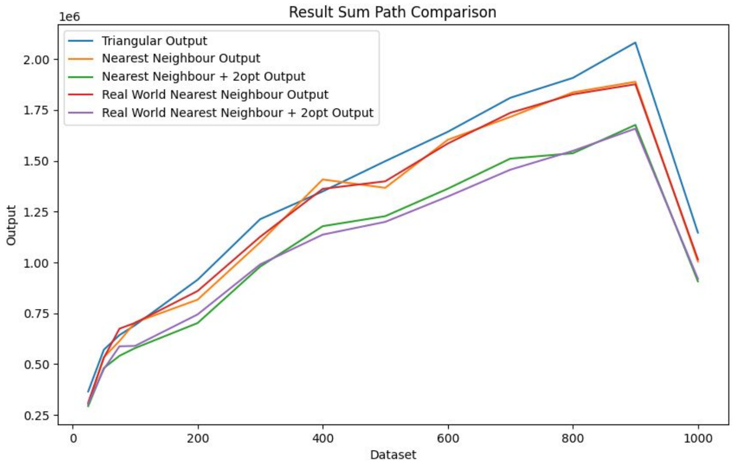
As seen by the estimate execution time, the Backtracking Algorithm isn’t usable for real-world graphs.

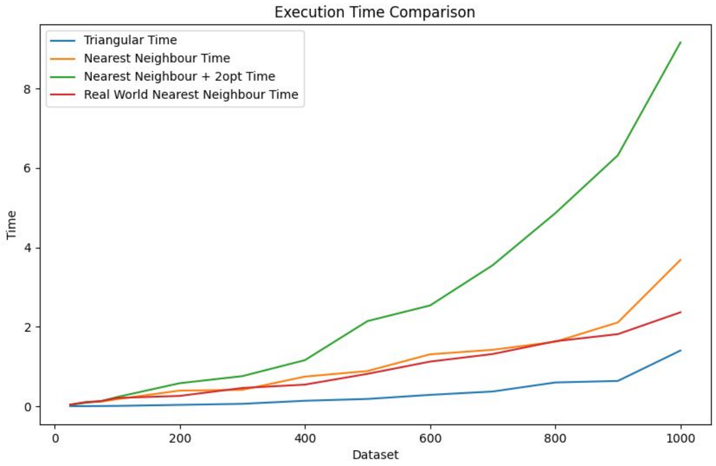


$$\text{time}(n_2) = f(n_2)/f(n_1) \times \text{time}(n_1)$$

where f(n) is n!

Algorithm Analysis





Algorithm Analysis

Main conclusions:

- The main conclusion is a clear trade-off between execution time and the minimization of sum-path, which makes it not trivial to choose the best algorithm just by looking at these two graphs

Algorithm Analysis

To analyse the trade-off, the following formula was used to calculate the ratio between two variables (execution time and output):

$$R = 0.5 \cdot \frac{x_1}{X_1} + 0.5 \cdot \frac{x_2}{X_2}$$

where X1 and X2 are the maximum value of each variable (we want the balance of the minimization of these two variables)

- The main conclusions are that real world nearest neighbour is usually better for fully connected graphs than triangular approximation and nearest neighbour without the optimization
- For real world TSP, the reasoning is the same