U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Design of Algorithms Second Project

Group G15_2
Bruno Oliveira up202208700
Rodrigo Silva up202205188
Vítor Pires up202207301

TABLE OF CONTENTS

Introduction

<u>Problem</u>:

- Solving the *Travelling Salesperson Problem;*

<u>Tools</u>:

- Implementation of a directed graph;
- Standard Template Library (STL);
- Appropriate data structures;
- Efficient algorithms.

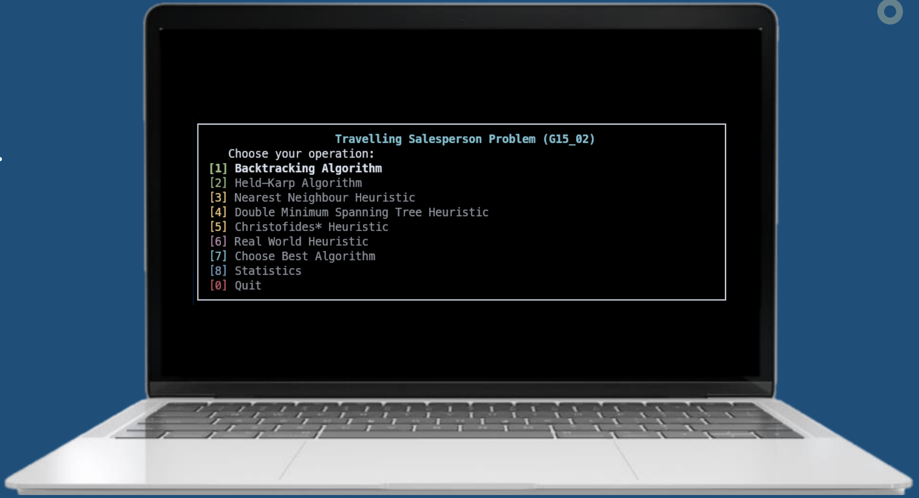


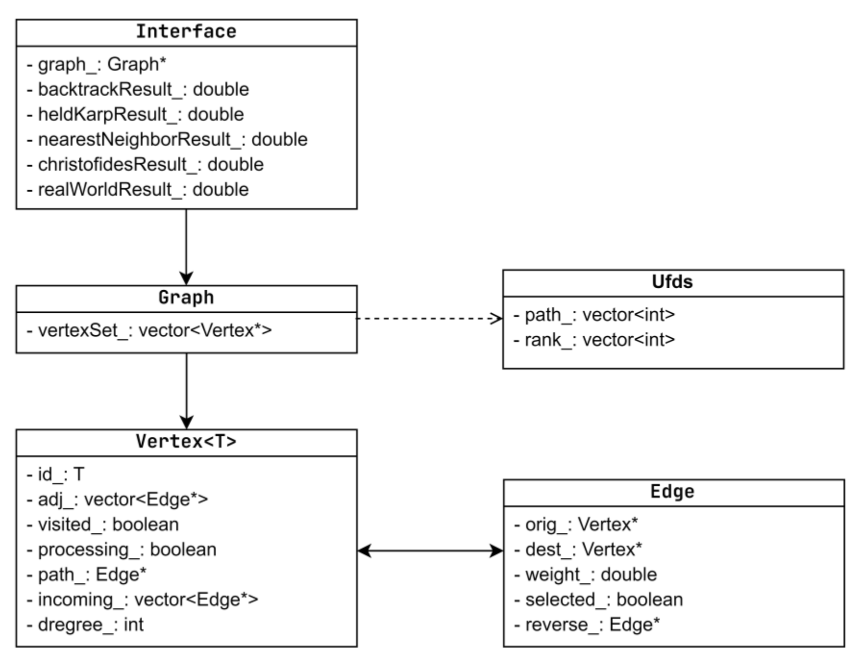
TABLE OF CONTENTS

Class Diagram

Each class:

- Is declared in a `.h` file with the same name (inside `include/`)
- Has its functionality defined in the respective `.cpp` file (inside `src/`, with the exception of the generic classes).

The app logic (main algorithms) is implemented as methods of `Graph`, and the `Interface` is responsible for the UI.

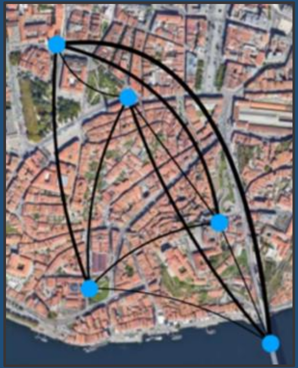


Graph Representation

- Adjacency list representation (similar to the one used in the practical classes);

Vertices | - Contain latitude and longitude (*).

Edges | - Contain distance between vertices.
- Undirected connections are represented using two (directed) edges.



origem	destino	distancia	label origem	label destino
0	1	1300	carmo	dLuis
0	2	1000	carmo	se
0	3	450	carmo	clerigos
0	4	750	carmo	bolsa
1	2	450	dLuis	se
1	3	950	dLuis	clerigos
1	4	450	dLuis	bolsa
2	3	500	se	clerigos
2	4	600	se	bolsa
3	4	750	clerigos	bolsa

TABLE OF CONTENTS

Reading the Dataset

```
Travelling Salesperson Problem (G15_02)
  Choose the Dataset
[1] Extra Fully Connected Graphs
[2] Real World Graphs
[3] Toy Graphs
[4] Custom dataset (see README)
[0] Quit
```

edges.csv
(nodes.csv)

Graph.h

- parseToyGraph(“edges.csv”)
- parseMediumGraph(“nodes.csv”, "edges.csv")
- ParseRealWorldGraph("nodes.csv", "edges.csv")

Parse the data files → Initialize data structures

Implemented Algorithms

```
Travelling Salesperson Problem (G15_02)
   Choose your operation:
[1] Backtracking Algorithm
[2] Held-Karp Algorithm
[3] Nearest Neighbour Heuristic
[4] Double Minimum Spanning Tree Heuristic
[5] Christofides* Heuristic
[6] Real World Heuristic
[7] Choose Best Algorithm
[8] Statistics
[0] Quit
```

Backtracking w/ Branch-and-Bound

```
function backtrackingTsp(Graph G):
  function bound(v, w, currDist, minDist):
    return currDist + d[v, w] < minDist;

  function backtrack(v, currDist, minDist):
    if all vertices are visited then
      minDist = min(minDist, currDist + d[v, 0]);
      return;
    visited[v] = true;
    foreach neighbor vertex w of vertex do
      if not visited[w] and bound(v, w, currDist, minDist) then
        backtrack(w, currDist + d[v, w], minDist);
    visited[v] = false;

  minDist = +inf;
  backtrack(V[G][0], 0, minDist);
  return minDist;
```

Complexity: $O(V!)$

Held-Karp Algorithm

```
function heldKarp(Graph G):
  for v = 1 to n - 1 do
    dp[{},v] = d[0,v];

  for k = 2 to n – 1 do
    for each subset S of {1,2,…,n−1} with |S| = k do
      for each v in S do
        dp[S\{v},v] = min {dp[S\{v,w},w] + d[v,w]: w in S\{v}}

  return min {dp[{1,2,…,n-1}\{v},v] + d[v,0]: v in {1,2,…,n-1}}
```

Time complexity: $O(n^2 * 2^n)$
Space complexity: $O(n * 2^n)$

This algorithm finds the exact solution using dynamic programming.
It builds a table dp that stores in $dp[S, v]$, for each $v \neq 0$ and S in $\{1, \dots, n-1\}\backslash\{v\}$, the shortest one-way path that goes from vertex labeled 0 to v, by going through all the vertices in S one time and one time only. The problem is then solved through the recurrence:

$$dp[S, v] = \begin{cases} d[S, v], & S = \emptyset \\ \min\limits_{w \in S}(dp[S\backslash\{w\}, w] + d[w, v]), & S \neq \emptyset \end{cases}$$

Double Minimum Spanning Tree

```
function doubleMst(Graph G):
  find a minimum spanning tree of G;
  construct an Euler cycle by duplicating the MST edges;
  traverse the cycle to build an Hamiltonian circuit using a DFS;
  return the Hamiltonian circuit as the solution;
```

Complexity: $O(V^2 * \log(V))$

It is proven, in the theoretical classes, that this is a 2-approximation algorithm for instances of the TSP that respect the triangular inequality

Nearest Neighbor

```
function nearestNeighbor(Graph G, Vertex start):
  v = start;
  dist = 0;
  While there are no more vertices unvisited
    w = nearest unvisited vertex;
    dist += d[v,w];
    v = w;

  dist += d[v,start];
  return dist;
```

Complexity: $O(V^2)$

This greedy algorithm connects a vertex to its nearest unvisited neighbor
It can be proven that this is a $(½ \lceil \log_2 V \rceil + ½)$-approximation algorithm to the TSP, for graphs that respect the triangular inequality[1]

[1] - Rosenkrantz, Daniel J., Richard E. Stearns, and Philip M. Lewis, II. "An analysis of several heuristics for the traveling salesman problem." *SIAM journal on computing* 6.3 (1977): 563-581.

Christofides* Heuristic

```
function minWeightPerfectMatching(Graph g):
  matching = {};
  for each edge in ascending order of weight
    if the edge matches two unmatched vertices
      append edge to matching;
   return matching;

function christofidesStar(Graph g):
  find a minimum spanning tree of G;
  construct an Euler cycle by duplicating the MST edges;
  traverse the cycle to build an Hamiltonian circuit using a DFS;

  Find matching of vertices with odd degree in MST using
minWeightPerfectMatching;
  join matching with MST to form an Euler cycle;
  traverse the cycle to build another Hamiltonian circuit;

  return smallest of the two Hamiltonian circuits;
```

Complexity: $O(V^2 * \log(V))$

Christofides* Heuristic

This algorithm is inspired on the Christofides heuristic, a $3/2$-approximation ratio for instances of the TSP that respect the triangular inequality. However, Christofides* finds the minimum weight perfect matching of the vertices with odd degree through a greedy heuristic instead of the Blossom Belief Propagation algorithm, thus not being able to ensure the approximation ratio of $3/2$.

Nonetheless, since it also performs the Double MST and compares it to the other result, it has a guaranteed approximation ratio of 2 with the triangular inequality.

Real World Heuristic

Finding a Hamiltonian cycle on non-fully connected graphs is a NP-complete problem. So, to be able to find always a solution to the TSP for this type of graphs requires us to relax a condition of the initial problem.

In the real world, when we want a through a set of cities, it is not very relevant if we need to visit a city more than once. This way, we relax the condition of the TSP stating that no vertex can be visited more than once.

Through this logic, we developed a heuristic that computes the graph of shortest distances, through the Floyd-Warshall algorithm, and performs the Christofides* heuristic. A nice property of shortest distances is that they always respect the triangular inequality: the shortest path from u to v must be always shorter than any two-step path between these vertices.

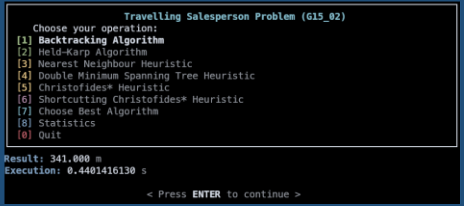
Shortcutting Christofides*

```
Function floydWarshall(Graph G):
  dist = |V|x|V| matrix, with 0 if i=j and +inf otherwise;
  for each edge (u,v) of E[G] do
    dist[u,v] = d[u,v];
  n = |V[G]|
  for w = 0 to n-1 do
    for u = 0 to n-1 do
      for v = 0 to n-1 do
        dist[u,v] = min(dist[u,v], dist[u,w] + dist[w,v]);
  return dist;

function shortcuttingChristofidesStar(Graph G):
  Calculate APSP using floydWarshall;
  Switch graph edge weights with the shortest distances;
  return solution to TSP using Christofides*;
```

Complexity: $O(V^3)$

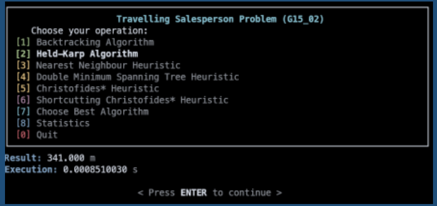
Implemented Algorithms

Backtracking



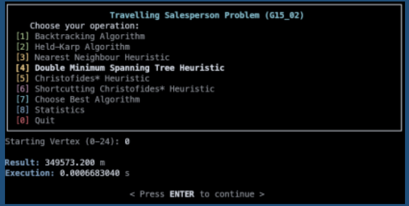
Stadiums

Held-Karp



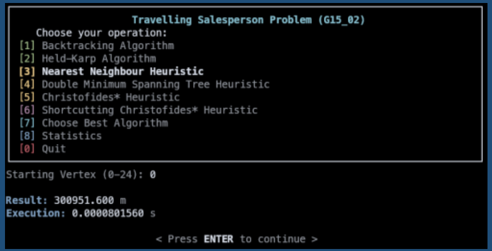
Stadiums

Double MST



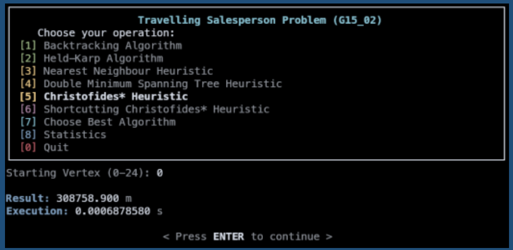
Extra 25 Nodes

Nearest Neighbor



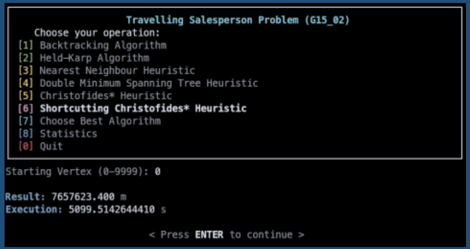
Extra 25 Nodes

Christofides*



Extra 25 Nodes

Shortcutting Christofides*



Real World 3

Other Features

Statistics

```
Travelling Salesperson Problem (G15_02)
  Choose your operation:
[1] Backtracking Algorithm
[2] Held-Karp Algorithm
[3] Nearest Neighbour Heuristic
[4] Double Minimum Spanning Tree Heuristic
[5] Christofides* Heuristic
[6] Shortcutting Christofides* Heuristic
[7] Choose Best Algorithm
[8] Statistics
[0] Quit
```

Algorithm	TSP Result	Time
Nearest Neighbor	300951.600	0.0000801560
Christofides*	308758.900	0.0006878580
Double MST	349573.200	0.0006683040

< Press **ENTER** to continue >

Choose Best Algorithm

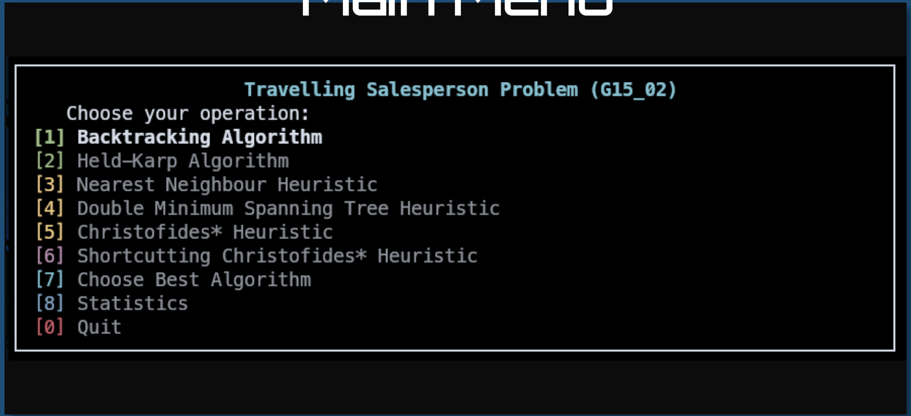
```
Travelling Salesperson Problem (G15_02)
  Choose your operation:
[1] Backtracking Algorithm
[2] Held-Karp Algorithm
[3] Nearest Neighbour Heuristic
[4] Double Minimum Spanning Tree Heuristic
[5] Christofides* Heuristic
[6] Shortcutting Christofides* Heuristic
[7] Choose Best Algorithm
[8] Statistics
[0] Quit

Number of vertices: 900 (>= 25 and < 1000)
Graph is fully connected
  Choosing Christofides* heuristic

Result: 2052974.000 m
Execution: 0.9579798500 s

< Press ENTER to continue >
```

Interface

- **Terminal User Interface** (TUI) that can be controlled with and Return and Arrow keys.
- There are input capture functions defined in input.h.

- To make the menus, box-drawing characters are used and to change the colors, we've used macros with ANSI escape sequences (library created in ansi.h).

Main Menu



Interface

Dataset Selection

Travelling Salesperson Problem (G15_02)

Choose the Dataset

[1] **Extra Fully Connected Graphs**
[2] Real World Graphs
[3] Toy Graphs
[4] Custom dataset (see README)
[0] Quit

Statistics for previous algorithms

Travelling Salesperson Problem (G15_02)

Choose your operation:

[1] Backtracking Algorithm
[2] Held-Karp Algorithm
[3] Nearest Neighbour Heuristic
[4] Double Minimum Spanning Tree Heuristic
[5] Christofides* Heuristic
[6] Shortcutting Christofides* Heuristic
[7] Choose Best Algorithm
[8] **Statistics**
[0] Quit

Algorithm	TSP Result	Time
Held-Karp	341.000	0.0007721970
Backtracking	341.000	0.4371560000
Christofides*	371.300	0.0001776290
Shortcutting Christofides*	371.300	0.0001944500
Double MST	398.100	0.0001622310
Nearest Neighbor	407.400	0.0000309050

< Press **ENTER** to continue >

TABLE OF CONTENTS

Highlighted Functionalities

The **implementation of multiple algorithms and heuristics** having in mind different graph constraints (translated to real world constraints). Due to the good documentation and information online about the TSP, it was easier to understand and develop multiple heuristics.

Responsive interface with relatively quick responses from the algorithms and relevant metrics/comparisons. Although not as important, it's always better to view the work and the information in a pleasant and organized manner.

Main difficulties

- Dealing with large datasets and algorithms with big time complexities.
- This forced us to adapt and find new approaches and heuristics to deal with the problems.

Contributions

- Each team member contributed significantly to different phases of the project, collectively trying to overcome these challenges.

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Bruno Oliveira – Rodrigo Silva – Vítor Pires
GROUP G15_2

May, 2024