Eulerian Path and Clustering coefficient for Graph: Implementation and Evaluation using NetworkX

Iacopo Erpichini

Advanced Algorithms and Graph Mining
University of Florence

Outline

Introduction

Data mining is a process of discovering patterns in large data-sets, the focus of this report is to analyze two algorithm applied to the official Italian Civil Protection COVID-19 dataset.

Introduction

Data mining is a process of discovering patterns in large data-sets, the focus of this report is to analyze two algorithm applied to the official Italian Civil Protection COVID-19 dataset.

Data

The first consideration is about understanding how the data are collected and stored.

Introduction

Data mining is a process of discovering patterns in large data-sets, the focus of this report is to analyze two algorithm applied to the official Italian Civil Protection COVID-19 dataset.

Data

The first consideration is about understanding how the data are collected and stored.

- The data are JSON referred to region and provinces
- The analysis of the algorithms is based on provinces JSON
- Each province has: id,date,name,region,**latitude**,**longitude**,number of infected

Algorithm studied

- First algorithm: Analysis graph creation using NetworkX with the COVID-19 dataset
- Eulerian path: Algorithm implementation and analysis
- Clustering coefficient: Algorithm implementation and analysis

Test configuration

All the algorithms are tested on a machine with this specs:

- CPU: Intel® Core™ i7-10750H, 12 thread, up to 5.00 GHz
- RAM: 2x8 Gb DDR4 2666 MHz
- OS: Ubuntu 20.04 LTS
- Python 3.7

Problem

- Build the **graph of provinces P**.
  Each node corresponds to a city and two cities are connected by an edge if the euclidean distance between the latitude and longitude of the two city is under a selected threshold
- Generate 2000 pairs of double (x,y) with x in [30,50) and y in [10,20).
  Building a **graph R** where each pair is a node and two nodes are connected with the same rule reported above.

Problem

- Build the **graph of provinces P**.
  Each node corresponds to a city and two cities are connected by an edge if the euclidean distance between the latitude and longitude of the two city is under a selected threshold
- Generate 2000 pairs of double (x,y) with x in [30,50) and y in [10,20).
  Building a **graph R** where each pair is a node and two nodes are connected with the same rule reported above.

For create the P graph is used the JSON of provinces, in particular the **latitude and longitude** of each province

Algorithm and complexity

Two algorithms of creation are analyzed

- **Simple**: Generate all the nodes, so for each node calculate the euclidean distance from all other node and an edge is added if the distance between node is under the selected threshold
- **Cost**: Exactly $\Theta(v^2)$

Algorithm and complexity

Two algorithms of creation are analyzed

- **Simple**: Generate all the nodes, so for each node calculate the euclidean distance from all other node and an edge is added if the distance between node is under the selected threshold
- **Cost**: Exactly $\Theta(v^2)$

- **Sorting**: Sorting list of nodes by longitude or latitude, so for each node check if the previous and following nodes in the list satisfy the euclidean distance conditions, if they don't, stop the algorithm for the analyzed node
- **Cost**: $O(v^2)$ the sorting approach is better than the simple one because only in the worst case costs $\Theta(v^2)$

Results

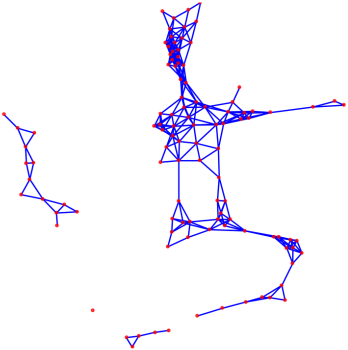
Time of execution on graph creation of Provinces and Random based on 100 test:

	Graph P	Graph R
Simpe algorithm	0.00391119 s	0.81928385 s
Sorting algorithm	0.00021886 s	0.00381270 s

The time analysis confirms the cost analysis of algorithm

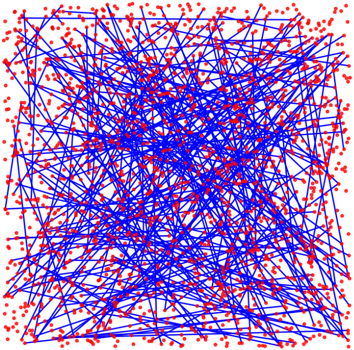
Results

- Graph of Provinces **P**: Represents the Italian provinces (node) and the edges are the connection between cities



Results

- Random Graph **R**



Definition

Eulerian path

It is a trail in a finite graph that visits every edge exactly once allowing for revisiting vertices

Eulerian circuit

It is an Eulerian trail that starts and ends on the same vertex.

Definition

Eulerian path

It is a trail in a finite graph that visits every edge exactly once allowing for revisiting vertices

Eulerian circuit

It is an Eulerian trail that starts and ends on the same vertex.

Necessary condition for the existence

All vertices in the graph have an even degree, and stated without proof that connected graphs with all vertices of even degree have an Eulerian circuit

Source: Wikipedia

Algorithm and Complexity

Hierholzer's algorithm

- Choose any starting vertex v, and follow a trail of edges from that vertex until returning to v. The tour formed in this way is a closed tour, but may not cover all the vertices and edges of the initial graph.
- As long as there exists a vertex u that belongs to the current tour but that has adjacent edges not part of the tour, start another trail from u, following unused edges until returning to u, and join the tour formed in this way to the previous tour.
- Since we assume the original graph is connected, repeating the previous step will exhaust all edges of the graph.

Algorithm and Complexity

Hierholzer's algorithm

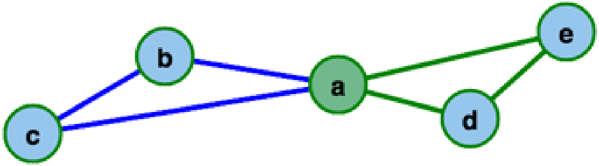
- Choose any starting vertex v, and follow a trail of edges from that vertex until returning to v. The tour formed in this way is a closed tour, but may not cover all the vertices and edges of the initial graph.
- As long as there exists a vertex u that belongs to the current tour but that has adjacent edges not part of the tour, start another trail from u, following unused edges until returning to u, and join the tour formed in this way to the previous tour.
- Since we assume the original graph is connected, repeating the previous step will exhaust all edges of the graph.

Complexity

$O(|E|)$ can be linear

Toy Example

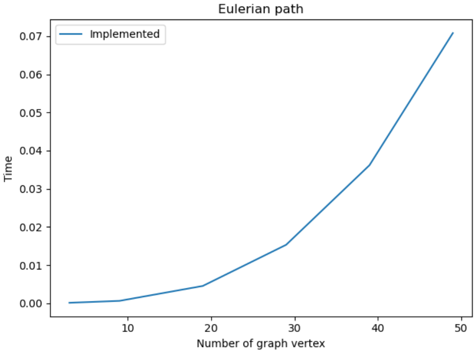
To execute Hierholzer's algorithm it's necessary to have a graph that is Eulerian or Semi-Eulerian (condition on vertex degree)
Hierholzer's algorithm tool



Eulerian path in P and R graphs doesn't exist because the graphs are not completely connected

Test and Results

Several Eulerian graphs of different sizes were generated with NetworkX to analyze the execution time as the number of nodes of the graph changes



Definition

Clustering coefficient is a measure of the degree to which nodes in a graph tend to cluster together

The focus is on local clustering coefficient of a node in a graph that quantifies how close its neighbours are to being a complete graph

Definition

Clustering coefficient is a measure of the degree to which nodes in a graph tend to cluster together

The focus is on local clustering coefficient of a node in a graph that quantifies how close its neighbours are to being a complete graph

$$C_i = \frac{2|\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\}|}{k_i(k_i - 1)}$$

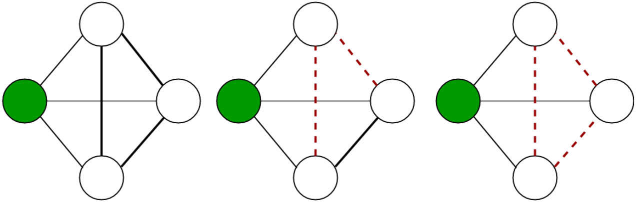
- N_i is the set of neighbors of node v_i
- e_{jk} represents the edge connecting node v_j the node v_k
- k_i is the number of neighbors of v_i

Source: Wikipedia

Toy Example

Example

The coefficient of the **green** node is computed as the proportion of connections among its neighbours which are actually realised compared with the number of all possible connections.



C = 1 C = 1/3 C = 0

Algorithm and complexity

- **Simple**: The coefficient is calculated for each node in the graph by counting the edges of all its neighbours cycling on the neighbours nodes
- **Cost**: Exactly $\Theta(V + E^2)$

- **With list intersection**: To calculate the dimension of the intersection between the neighborhood of v and the neighborhood of each of its neighbours uses the list intersection algorithm see at lesson
- **Cost**:$O(V + E * C)$, C is the List intersection cost that is $O(|list1| \log |list1| + |list2| \log |list2|)$

Results

Table show the execution time of the two algorithms implemented and the execution time of the library function

	Graph P	Graph R
Simple algorithm	0.0009195804 s	0.0015769004 s
With list intersection	0.0020079612 s	0.0015769004 s
NetworkX	0.0030195713 s	0.0125777721 s

The performances on clustering coefficient algorithms implemented are better than the NetworkX version maybe because the library calculate and returns some object that requires much calculation time

End

Thanks for the attention