Analysis and Implementation of Counting Triangles and Eccentricity on the Graph of Italian Provinces

Noemi Benci
Federico Pirona

July, 2020

University of Florence

Table of contents

Introduction

Introduction

- Data: *`https://github.com/pcm-dpc/COVID-19/blob/master/dati-json/dpc-covid19-ita-province.json`*
- Provinces are vertexes of the graph *P* and are characterized by position; it is given by latitude and longitude
- Ties between vertexes *v* and w are created if *w* is into the square centered in *v* of side d = 0.8
- Generate a more populous random graph *R* with 2000 vertexes; edges are generated in the same way of *P*, but now d = 0.08
- Compute Euclidean Distance for each edge of *P* and *R*
- Global Indices and Structures: Counting Triangles *`http://timroughgarden.org/s14/l/l1.pdf`*
- Centrality: Eccentricity centrality *`https://en.wikipedia.org/wiki/Distance_(graph_theory)`*

Building the Graph of Provinces

Preliminaries and Adding vertexes to the Graph

Data description:

- Each record is a dictionary, where keys are variables names;
- Observations are information of a particular province in a day;
  - Because we need only latitude and longitude, we selected **only one date** to extract provinces.
- Some provinces have latitude and longitude equals to zero
  - Because none of the Italian cities stays on the Equator, we dropped these **fake records**.

Preliminaries and Adding vertexes to the Graph

Data description:

- Each record is a dictionary, where keys are variables names;
- Observations are information of a particular province in a day;
  - Because we need only latitude and longitude,
    we selected **only one date** to extract provinces.
- Some provinces have latitude and longitude equals to zero
  - Because none of the Italian cities stays on the Equator,
    we dropped these **fake records**.

Adding vertexes to the Graph: Among the provinces of a single day, we inserted in the list only those with latitude different from zero. We saved only information about names, latitude and longitude.

Vicinity of nodes

When two vertexes are said to be near?

- u, v are near by latitude if:

$$\left|u.lat - v.lat\right| < d$$

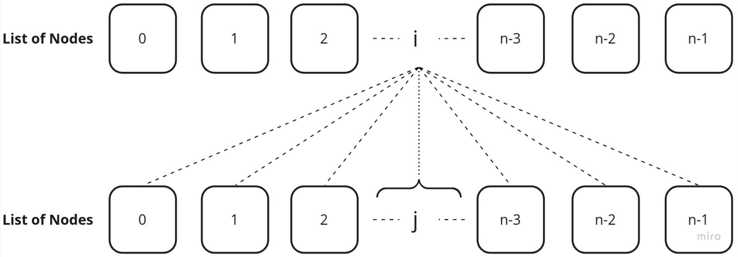
- u, v are near by longitude if:

$$\left|u.long - v.long\right| < d$$

- u, v are near if they are near by latitude and by longitude.
  If u, v are near, the edge (u, v) is added to the graph.

Adding edges to the Graph I: Trivial Solution

Compare each province with the others

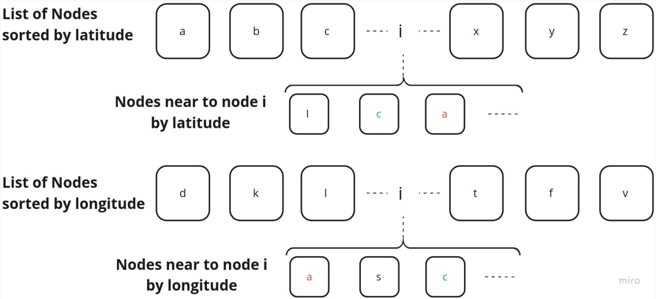


If two provinces are near for both latitude and longitude, add an edge.

Cost: $O(n^2)$.

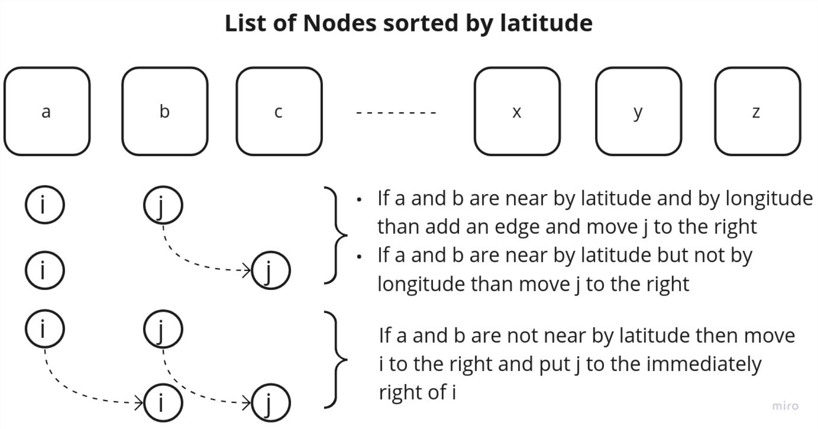
Adding edges to the Graph II: Better Solution

- Sort nodes by latitude; Sort nodes by longitude;
- For each node *i*:
  - Find all the nodes near to *i* by latitude;
  - Find all the nodes near to *i* by longitude;
  - Find the intersection between the two sets;
- Add edges between node *i* and each node in the intersection.



Cost: $O(n \log(n) + m)$.

Adding edges to the Graph III: Best Solution



Cost: $O(n \log(n) + m)$.

Building the Random Graph

Building Random Graph R

Every node of the Graph R is located in coordinates (x, y); with uniform probability $x \in [30, 50)$, $y \in [10, 20)$.
We can get random coordinates of Graph R in two steps:

1. Generating 2n random numbers: $a \in [0 - 1)$
2. Given numbers in [0-1], a lower bound *l* and upper bound *u*, apply:

$$b = l + a(u - l)$$

First *n* numbers have *l* = 30, *u* = 50.
Last *n* numbers have *l* = 10, *u* = 20.
b are the coordinates for nodes of the random Graph R!

Time needed to build the Graphs

Graph	function[1]	%timeit [2]
P	set_nodes	90.7 µs ± 2.93 µs
	sorting_by (lat)	61.8 µs ± 3.13 µs
	set_edges (Trivial)	12 ms ± 965 µs
	set_edges (Better)	10.7 ms ± 359 µs
	set_edges (Best)	4.66 ms ± 270 µs
R	set_nodes	1.85 ms ± 78 µs
	sorting_by (lat)	1.33 ms ± 60.3 µs
	set_edges (Trivial)	3.75 s ± 89.5 ms
	set_edges (Better)	2.72 s ± 114 ms
	set_edges (Best)	66.8 ms ± 3.61 ms

Trivial and **Better** solutions: expensive. **Best solution**: cheap.

[1] Total time of best solution is given by sorting_by + set_edges (Best)
[2] Times can vary using different computers

Adding weights

Adding weights to edges as Euclidean Distance I

This step is almost immediate:

- For each edge $(u, v) \in E$, compute Euclidean distance between the two nodes u, v:

$$d(u, v) = \sqrt{(u.lat - v.lat)^2 + (u.long - v.long)^2}$$

Adding weights to edges as Euclidean Distance II

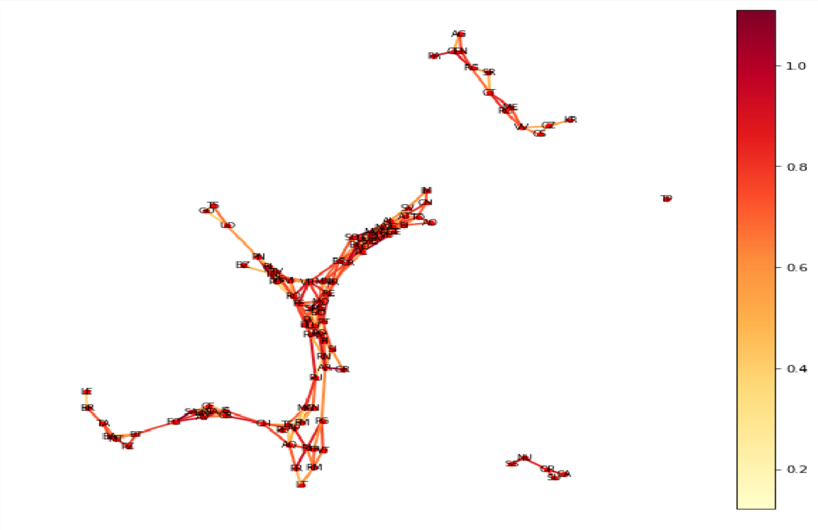


Figure 1: P Graph

Adding weights to edges as Euclidean Distance III

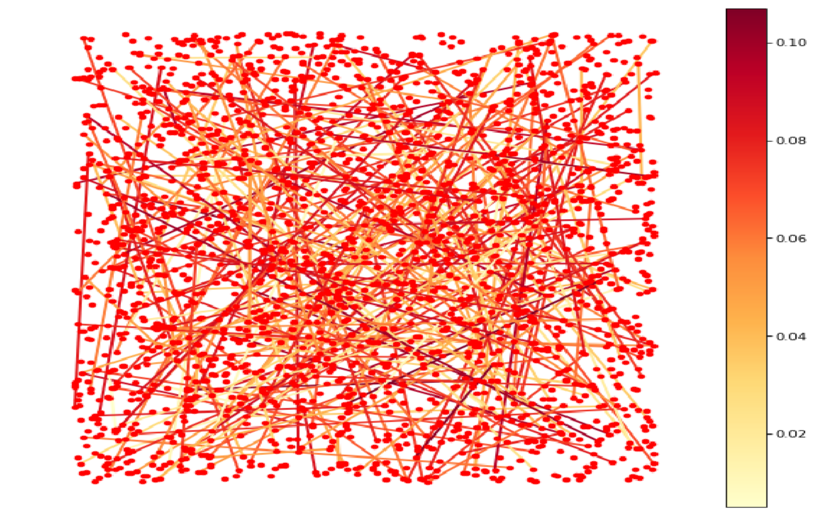


Figure 2: R Graph

Counting Triangles Algorithm

Counting Triangles I

Definition: a triangle is a set of three vertices that are mutually adjacent in G.

Two faces of the Counting Triangles problem:

- How many triangles does a graph G have?
- For every node *v*, how many triangles in G include the node *v*?

Three Solutions:

- Brute-force -> $O(n^3)$
- Better -> $\Theta\left(\sum_{v \in V} deg(v)^2\right)$
  - $O(n^3)$
  - $O(n)$ if every node has the same degree
- Best -> $O(m^{3/2})$

Counting Triangles II: Brute-force Solution

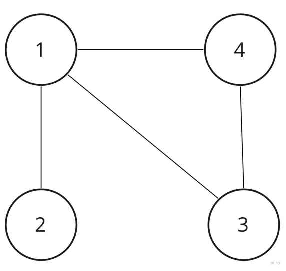
For every possible triple of nodes, check if it is actually present in the Graph G.

For example:

Given $G = (V, E)$ where $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (1, 3), (1, 4), (3, 4)\}$

Every possible triplet:
$C = \{(1, 2, 3), (1, 4, 3), (1, 3, 4), (2, 3, 4)\}$

We scan C and for every element we check if the triangle exists in G. In the example the only triangle is $\{(1, 4, 3)\}$ so the number of triangles is 1.



Counting Triangles III: Brute-force Solution

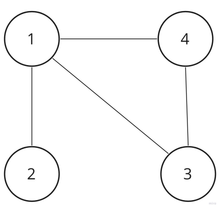
Every triple of distinct vertices is iterated, regardless of degree of the nodes.
The cost function is **always** $\binom{n}{3}$, which belongs to $O(n^3)$.

Counting Triangles IV: Better Solution

From the neighborhood of each node we check if there are triangles.

For example:

Given $G = (V, E)$ where $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (1, 3), (1, 4), (3, 4)\}$



Counting Triangles IV: Better Solution

From the neighborhood of each node we check if there are triangles.

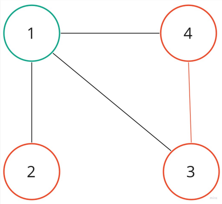
For example:

Given $G = (V, E)$ where $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (1, 3), (1, 4), (3, 4)\}$

- Neighbours of node 1: $ne(1) = \{2, 3, 4\}$.

Check if there are edges between $\{2, 3\}$, $\{2, 4\}$, $\{3, 4\}$.

The only link is between $\{3, 4\}$ -> Triangle found: +1



Counting Triangles IV: Better Solution

From the neighborhood of each node we check if there are triangles.

For example:

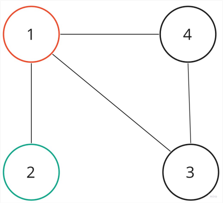
Given $G = (V, E)$ where $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (1, 3), (1, 4), (3, 4)\}$

- Neighbours of node 1: $ne(1) = \{2, 3, 4\}$.

Check if there are edges between $\{2, 3\}$, $\{2, 4\}$, $\{3, 4\}$.

The only link is between $\{3, 4\}$ -> Triangle found: +1

- Neighbours of node 2: $ne(2) = \{1\}$. Only 1 neighbour -> not a triangle.



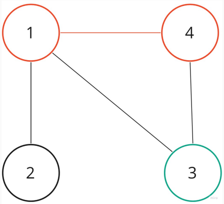
Counting Triangles IV: Better Solution

From the neighborhood of each node we check if there are triangles.

For example:

Given $G = (V, E)$ where $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (1, 3), (1, 4), (3, 4)\}$
- Neighbours of node 1: $ne(1) = \{2, 3, 4\}$.
Check if there are edges between $\{2, 3\}$, $\{2, 4\}$, $\{3, 4\}$.
The only link is between $\{3, 4\}$ -> Triangle found: +1

- Neighbours of node 2: $ne(2) = \{1\}$. Only 1 neighbour -> not a triangle.
- Neighbours of node 3: $ne(3) = \{1, 4\}$.
There is an edge between $\{1, 4\}$-> Triangle found: +1



Counting Triangles IV: Better Solution

From the neighborhood of each node we check if there are triangles.

For example:

Given $G = (V, E)$ where $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (1, 3), (1, 4), (3, 4)\}$

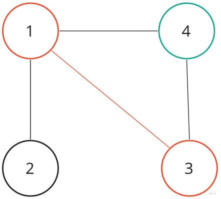
- Neighbours of node 1: $ne(1) = \{2, 3, 4\}$.

Check if there are edges between $\{2, 3\}$, $\{2, 4\}$, $\{3, 4\}$.

The only link is between $\{3, 4\}$ -> Triangle found: +1

- Neighbours of node 2: $ne(2) = \{1\}$. Only 1 neighbour -> not a triangle.
- Neighbours of node 3: $ne(3) = \{1, 4\}$.

There is an edge between $\{1, 4\}$-> Triangle found: +1



- Neighbours of node 4: $ne(4) = \{1, 3\}$.

There is an edge between $\{1, 3\}$-> Triangle found: +1

Counting Triangles IV: Better Solution

From the neighborhood of each node we check if there are triangles.

For example:

Given $G = (V, E)$ where $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (1, 3), (1, 4), (3, 4)\}$

- Neighbours of node 1: $ne(1) = \{2, 3, 4\}$.

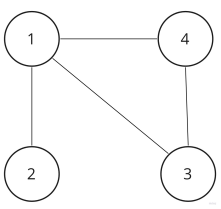
Check if there are edges between $\{2, 3\}$, $\{2, 4\}$, $\{3, 4\}$.

The only link is between $\{3, 4\}$ -> Triangle found: +1

- Neighbours of node 2: $ne(2) = \{1\}$. Only 1 neighbour -> not a triangle.

- Neighbours of node 3: $ne(3) = \{1, 4\}$.

There is an edge between $\{1, 4\}$-> Triangle found: +1



- Neighbours of node 4: $ne(4) = \{1, 3\}$.

There is an edge between $\{1, 3\}$-> Triangle found: +1

The final count is $3/3 = 1$ -> 1 Triangle found.

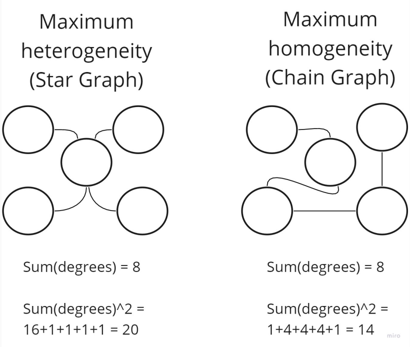
Counting Triangles V: Better Solution

For every vertex v, each pair $u, w \in N(v)$ is enumerated. The function cost is hence:

$$\sum_{v \in V} \binom{deg(v)}{2} \simeq \sum_{v \in V} deg(v)^2$$

Then we can conclude that:

- if the graph is complete, the cost is again $O(n^3)$
- Keeping fixed m ——>

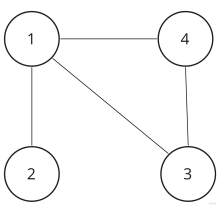


Counting Triangles VI: Best Solution

Sort the list of nodes by increasing degree. For each node in the sorted list, if the neighbours that have greater degree are linked, then a triangle is found.

For example:

Given $G = (V, E)$ where $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (1, 3), (1, 4), (3, 4)\}$



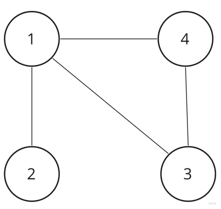
[3] Nodes with same degree are sorted using the alphanumeric order.

Counting Triangles VI: Best Solution

Sort the list of nodes by increasing degree. For each node in the sorted list, if the neighbours that have greater degree are linked, then a triangle is found.

For example:

Given $G = (V, E)$ where $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (1, 3), (1, 4), (3, 4)\}$
The list of nodes sorted by increasing degree is $\{2, 3, 4, 1\}$. [3]



[3] Nodes with same degree are sorted using the alphanumeric order.

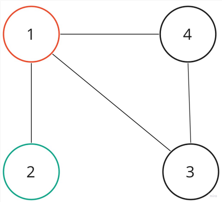
Counting Triangles VI: Best Solution

Sort the list of nodes by increasing degree. For each node in the sorted list, if the neighbours that have greater degree are linked, then a triangle is found.

For example:

Given $G = (V, E)$ where $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (1, 3), (1, 4), (3, 4)\}$
The list of nodes sorted by increasing degree is $\{2, 3, 4, 1\}$. [3]

- Neighbors of node 2: $ne(2) = \{1\}$. Only 1 neighbour -> not a triangle.



[3] Nodes with same degree are sorted using the alphanumeric order.

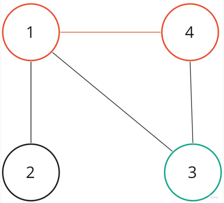
Counting Triangles VI: Best Solution

Sort the list of nodes by increasing degree. For each node in the sorted list, if the neighbours that have greater degree are linked, then a triangle is found.

For example:

Given $G = (V, E)$ where $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (1, 3), (1, 4), (3, 4)\}$
The list of nodes sorted by increasing degree is $\{2, 3, 4, 1\}$. [3]

- Neighbors of node 2: $ne(2) = \{1\}$. Only 1 neighbour -> not a triangle.
  - Neighbors of node 3: $ne(3) = \{4, 1\}$.
  Because $deg(4) > deg(3)$, $deg(1) > deg(3)$ and $\{4, 1\}$ are linked -> Triangle found +1



[3] Nodes with same degree are sorted using the alphanumeric order.

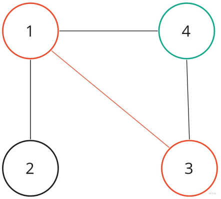
Counting Triangles VI: Best Solution

Sort the list of nodes by increasing degree. For each node in the sorted list, if the neighbours that have greater degree are linked, then a triangle is found.

For example:

Given $G = (V, E)$ where $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (1, 3), (1, 4), (3, 4)\}$
The list of nodes sorted by increasing degree is $\{2, 3, 4, 1\}$. [3]

- Neighbors of node 2: $ne(2) = \{1\}$. Only 1 neighbour -> not a triangle.
- Neighbors of node 3: $ne(3) = \{4, 1\}$.
  Because $deg(4) > deg(3)$, $deg(1) > deg(3)$ and $\{4, 1\}$ are linked -> Triangle found +1
- Neighbors of node 4: $ne(4) = \{3, 1\}$.
  Because $deg(3) < deg(4)$->next node



[3] Nodes with same degree are sorted using the alphanumeric order.

Counting Triangles VI: Best Solution

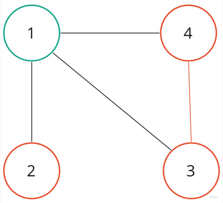
Sort the list of nodes by increasing degree. For each node in the sorted list, if the neighbours that have greater degree are linked, then a triangle is found.

For example:

Given $G = (V, E)$ where $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (1, 3), (1, 4), (3, 4)\}$
The list of nodes sorted by increasing degree is $\{2, 3, 4, 1\}$. [3]

- Neighbors of node 2: $ne(2) = \{1\}$. Only 1 neighbour -> not a triangle.

  - Neighbors of node 3: $ne(3) = \{4, 1\}$.
  Because $deg(4) > deg(3)$, $deg(1) > deg(3)$ and $\{4, 1\}$ are linked -> Triangle found +1
  - Neighbors of node 4: $ne(4) = \{3, 1\}$.
  Because $deg(3) < deg(4)$->next node
  - Neighbors of node 1: $ne(1) = \{2, 3, 4\}$.
  Because $deg(2) < deg(1)$->stop



[3] Nodes with same degree are sorted using the alphanumeric order.

Counting Triangles VI: Best Solution

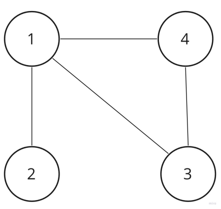
Sort the list of nodes by increasing degree. For each node in the sorted list, if the neighbours that have greater degree are linked, then a triangle is found.

For example:

Given $G = (V, E)$ where $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (1, 3), (1, 4), (3, 4)\}$
The list of nodes sorted by increasing degree is $\{2, 3, 4, 1\}$. [3]

- Neighbors of node 2: $ne(2) = \{1\}$. Only 1 neighbour -> not a triangle.

- Neighbors of node 3: $ne(3) = \{4, 1\}$.
Because $deg(4) > deg(3)$, $deg(1) > deg(3)$ and $\{4, 1\}$ are linked -> Triangle found +1
- Neighbors of node 4: $ne(4) = \{3, 1\}$.
Because $deg(3) < deg(4)$->next node
- Neighbors of node 1: $ne(1) = \{2, 3, 4\}$.
Because $deg(2) < deg(1)$->stop



Number of Triangles: 1.

[3]Nodes with same degree are sorted using the alphanumeric order.

Counting Triangles VII: Best Solution

Algorithm 1: Counting Triangles: Best Solution

Input : "sorted_list": list of nodes of a Graph G, sorted by increasing degree

Output: Number of triangles in the Graph G

n_triangles = 0

foreach *node in sorted_list* **do**

  foreach (i, j) *in neighbour(node)* **do**

    if $(i, j) \in E$ ***and*** $deg(i) > deg(node)$ ***and*** $deg(j) > deg(node)$ **then**

      n_triangles += 1

    end

  end

end

return n_triangles

Counting Triangles VIII: Best Solution Cost

- Every triangle is counted only once;
- No work done for nodes with highest degree;
- It is demonstrated that the cost of the algorithm is $O(m^{3/2})$:
  - The worst case is when there are $2\sqrt{m}$ nodes with degree $\sqrt{m}$ (other nodes are isolated).
  - If graph is complete, the complexity is again $O(n^3)$

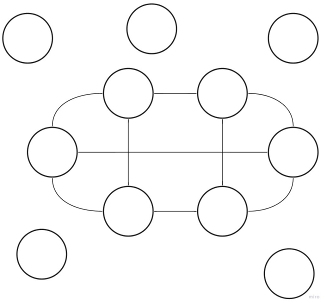


Figure 3: The worst case - Graphical example

Counting Triangles IX: Application to P and R

Graph	function	# Triangles	%timeit [4]
P	Best	352	1.01 ms ± 51.5 μs
	NetworkX	352	2.29 ms ± 138 μs
R	Best	12	4.52 ms ± 194 μs
	NetworkX	12	14 ms ± 1.17 ms

The Best Solution for counting triangles is cheaper than the solution of NetworkX.

[4] Times can vary using different computers

Eccentricity Algorithm

Eccentricity I

- The eccentricity $\epsilon(v)$ of a vertex v is the greatest distance between v and any other vertex. In symbols:

$$\epsilon(v) = \max_{u \in V} d(v, u)$$

- It can be thought of as how far a node is from the node most distant from it in the graph.
- If the graph is not connected, each node has eccentricity equal to ∞.
- A Breadth First Search on vertex v can give the distance between v and all the other $n-1$ nodes. In this way it is possible to take the maximum distance.

Eccentricity II

Algorithm 2: Eccentricity Algorithm, part 1

Input : a Graph G and one of his node v

Output: $\epsilon(v)$

foreach vertex $u_i \subset V$ **do**

  u$_i$.$level = \infty$

  u$_i$.$color = WHITE$

end

v.color = NOWHITE

v.level = 0

M = 0

Q = $\emptyset$

Q.enqueue(v)

Eccentricity III (continues...)

Algorithm 3: Eccentricity Algorithm, part 2

```
while Q ≠ ∅ do
    u = Q.dequeue()
    foreach ngbr ∈ G.adj[u] do
        if ngbr.color == WHITE then
            ngbr.color = NOWHITE
            ngbr.level = u.level + 1
            Q.enqueue(ngbr)
            if ngbr.level > M then
                M = ngbr.level
            end
        end
    end
end
return M
```

Eccentricity IV

- The cost of a single BFS is about $O(m)$ (the graph is connected).
- Because neither P nor R are connected, the eccentricity algorithm has been applied only to nodes belonging to the greatest connected component. We call k his cardinality.

Overall Cost: $O(km)$.

Eccentricity V: toy example

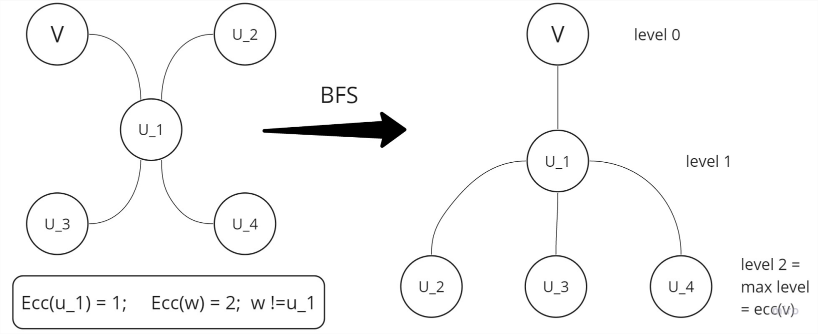


Figure 4: Toy Graph

Eccentricity V: Application to P and R

Graph	function	%timeit [5]
P	Our algorithm	63.3 ms ± 5.3 ms
	NetworkX	111 ms ± 7.54 ms
R	Our algorithm	158 μs ± 6.19 μs
	NetworkX	198 μs ± 11.2 μs

Our Algorithm performs better than the one used by the library 'NetworkX'.

[5] Times can vary using different computers

Conclusion

Conclusion

To save time, choose a smarter solution:

- The Best Solution for adding the edges to the Graph, saves a lot of time ($O(n \log(n) + m)$ instead of $O(n^2)$).
  In particular, when handling with large Graphs, the difference is evident.
- The Best Solution for the Counting Triangles Algorithm performs better than the algorithm provided by the library 'NetworkX'.
- Our Solution for computing the BFS (necessary to compute the eccentricity) performs better than the algorithm provided by the library 'NetworkX'.

Thank you for the attention!