Citation Distortion Through Graph Theory

Davide Aureli: 1651051

Alfonso D'Amelio: 1658170

Valerio Guarrasi: 1643257

Daniele Mocavini: 1475062

Federico Siciliano: 1604124

Are the authors' rankings based on citations really effective, as is commonly led to thinking?

With our analysis, based on Graph Theory, we were able to identify some critical issues (distortions) that we tried to solve for obtaining a more realistic idea of the importance that has an author in the field of computer science.
Analyzing the data provided by Incites we obtained temporal, geographical and distribution information of the citations of the authors of our interest, verifying the presence of possible correlations.
To recreate a possible situation in which there is presence of distortion, we have generated a subgraph starting from an author and using hop distance equal to 5; through the latter we managed to extract the characteristics that highlight the presence of distortion within the scientific field of interest.
After that, our team developed a model, using the graph theory, to solve or at least reduce, the two main types of distortion, the self-citation and the Exchanging Citation using a graph-based approach.

Table Of Contents

Introduction

We decided to tackle the third challenge, "**Challenges related to output measures and possible approaches**".
We have focused our attention on the data about the publications and citations of the authors related to the sphere of Computer Science.

We will try to analyze and organize millions of data that apparently seem to have no relation to each other but, we can represent them through a single structure: graphs.

Firstly, we have done a data analysis on publications data taken from Incites.
Once we have a complete view of our dataset, we can perform statistical analysis on the data we have treated.
At the end of the statistical part we will focus on the heart of our challenge, that is to analyze possible distortions both in the collection and in the analysis of our data. Obviously, our work, as mentioned previously, will focus on a specific sphere of study but at the same time it can be used on any dataset that InCites makes available to us, embracing the whole scientific world.

By studying the literature associated with this type of problem we found the work carried out by Greenberg SA. to be of interest and useful for our analysis. in its 2009 publication entitled "How citation distortions create unfounded authority: analysis of a citation network".
In this publication the author has highlighted 5 types of distortions associated with citations:
- Self Citation
- Citation Bias
- Amplification
- Similarity
- Invention

We decided to focus on the first type of distortion described by Greenberg and we have also identified and treated a new type of distortion, not present in the literature, or the Exchanging Citation.

Data Description

Researchers in science, communicate new discoveries, express ideas and measure success through publications and citations.
Citations are included in research papers (publications) for 2 main reasons:

- to build an argument
- to give credit to research done by others

Citations included in scientific writing are expected to be relevant and comprehensive; citation distortion is any misrepresentation of these features.
The most common citation distortions are:

- **Self-Citation**: professional researchers overly citing personal works. It is usually done to artificially inflate carrier statistics.
- **Exchanging Citation**: it is possible to have this type of distortion, especially in the academic field, between two authors who for example have collaborated in a particular publication and decide to refer to each other in future publications even if not relevant or inherent with their work.

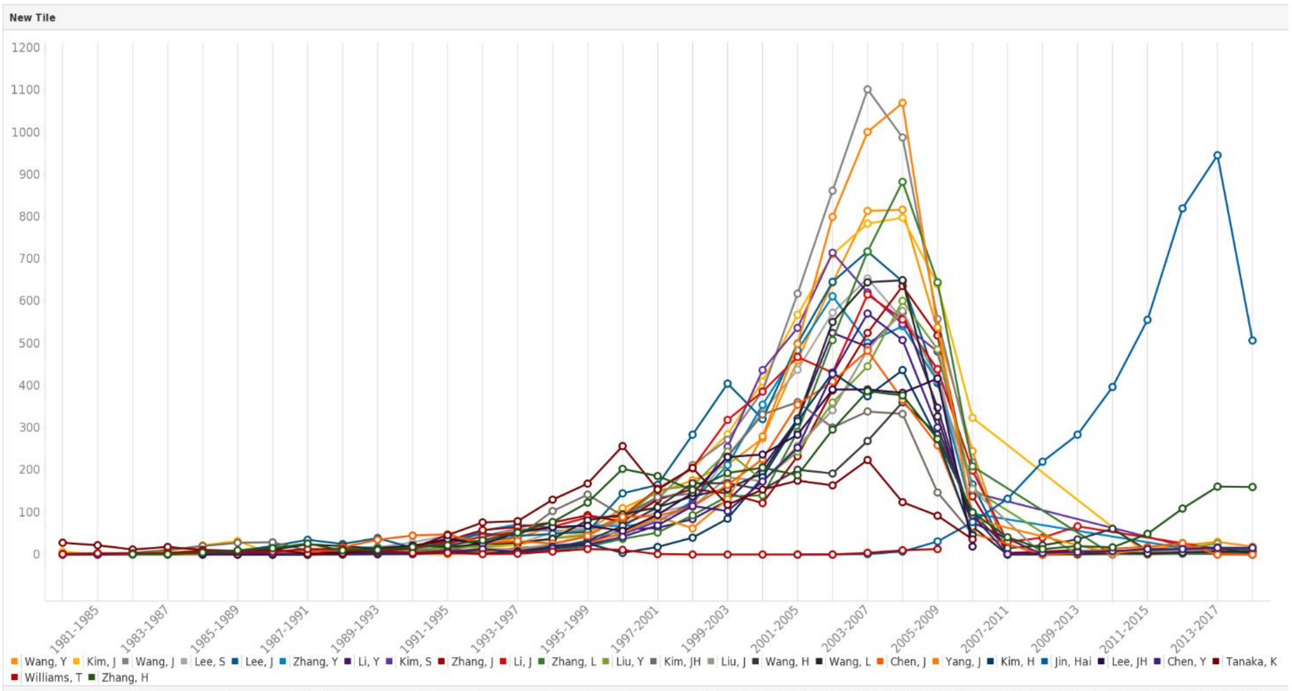
In our analysis we will try to identify and solve these two (and possibly other) types of distortions.

For our analysis of citations, we based on the data we found on the InCites site in the People section and applying the following filters:

- Computer Science, Artificial Intelligence
- Computer Science, Cybernetics
- Computer Science, Hardware & Architecture
- Computer Science, Information Systems
- Computer Science, Interdisciplinary Applications
- Computer Science, Software Engineering
- Computer Science, Theory & Methods

Getting a total of 2.700.774 results.

The first thing we verified is the trend that the citations concerning our branch of interest have had during the last 35 years.
For this purpose, in order to obtain a ranking based on the "number of Web of Science Documents" and taking the 25 most productive authors, we have obtained the following chart (Y: Times Cited) *[fig1]*

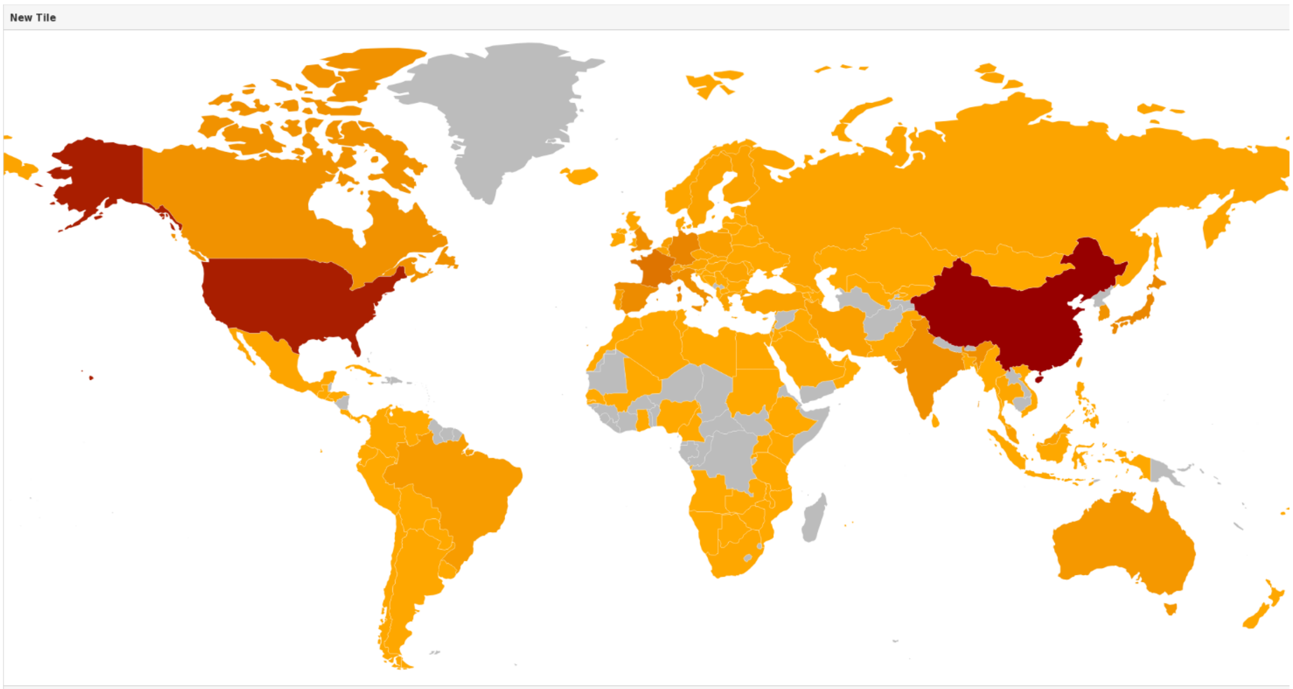


Indicators: Times Cited. **Person Name or ID:**NOT [Anonymous].**Schema:**Web of Science.**Research Area:**COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE,COMPUTER SCIENCE, CYBERNETICS,COMPUTER SCIENCE, HARDWARE & ARCHITECTURE,COMPUTER SCIENCE, INFORMATION SYSTEMS,COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS,COMPUTER SCIENCE, SOFTWARE ENGINEERING,COMPUTER SCIENCE, THEORY & METHODS
InCites dataset updated Apr 28, 2018. Includes Web of Science content indexed through Feb 27, 2018. Export Date: May 24, 2018.

[fig 1] – Time series of citations

Thanks to which we can observe the development that Computer Science has had since the second half of the 90s.

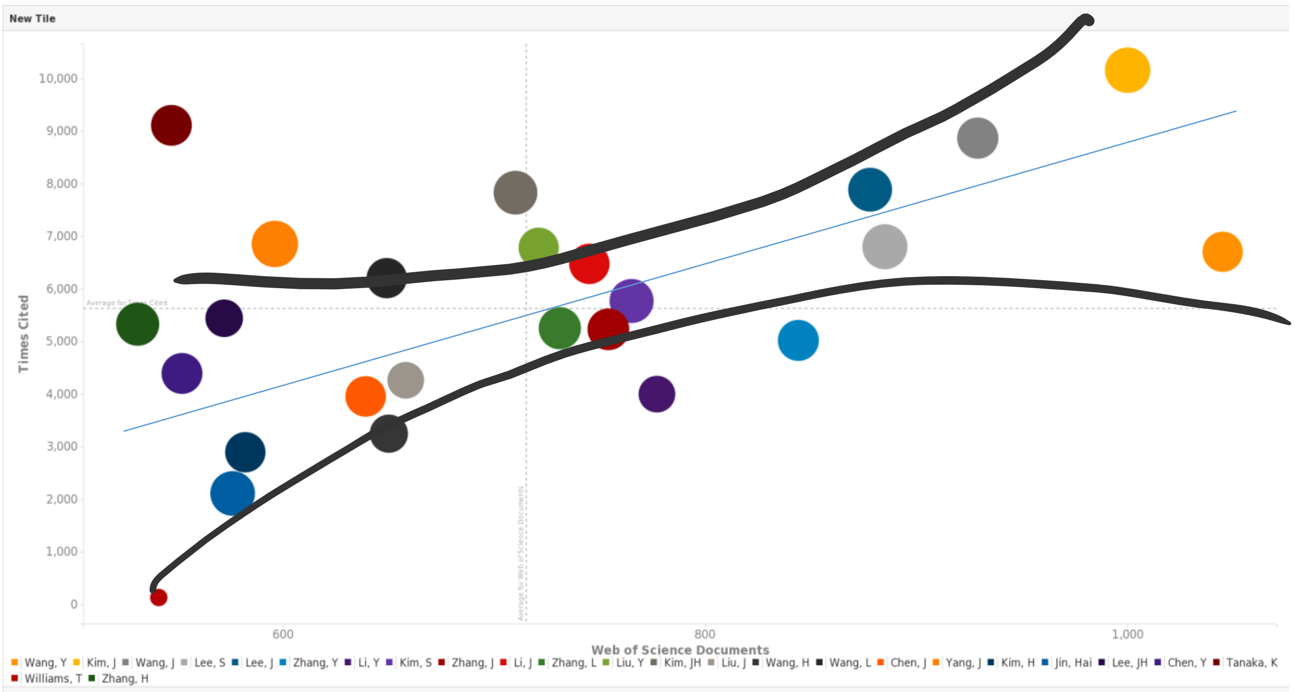
It is also interesting to note the country of origin *[fig 2]* of the most productive authors with a strong predominance of the USA and China.



Indicators: . **Person Name or ID:**NOT [Anonymous].**Schema:**Web of Science.**Research Area:**COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE,COMPUTER SCIENCE, CYBERNETICS,COMPUTER SCIENCE, HARDWARE & ARCHITECTURE,COMPUTER SCIENCE, INFORMATION SYSTEMS,COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS,COMPUTER SCIENCE, SOFTWARE ENGINEERING,COMPUTER SCIENCE, THEORY & METHODS
InCites dataset updated Apr 28, 2018. Includes Web of Science content indexed through Feb 27, 2018. Export Date: May 24, 2018.

[fig 2] – Geographical representation of author’s citations

We have therefore observed the relation between the number of document published (axes X) and the number of times that the author has been cited (axes Y) using as radius the % cited documents *[fig 3]*.

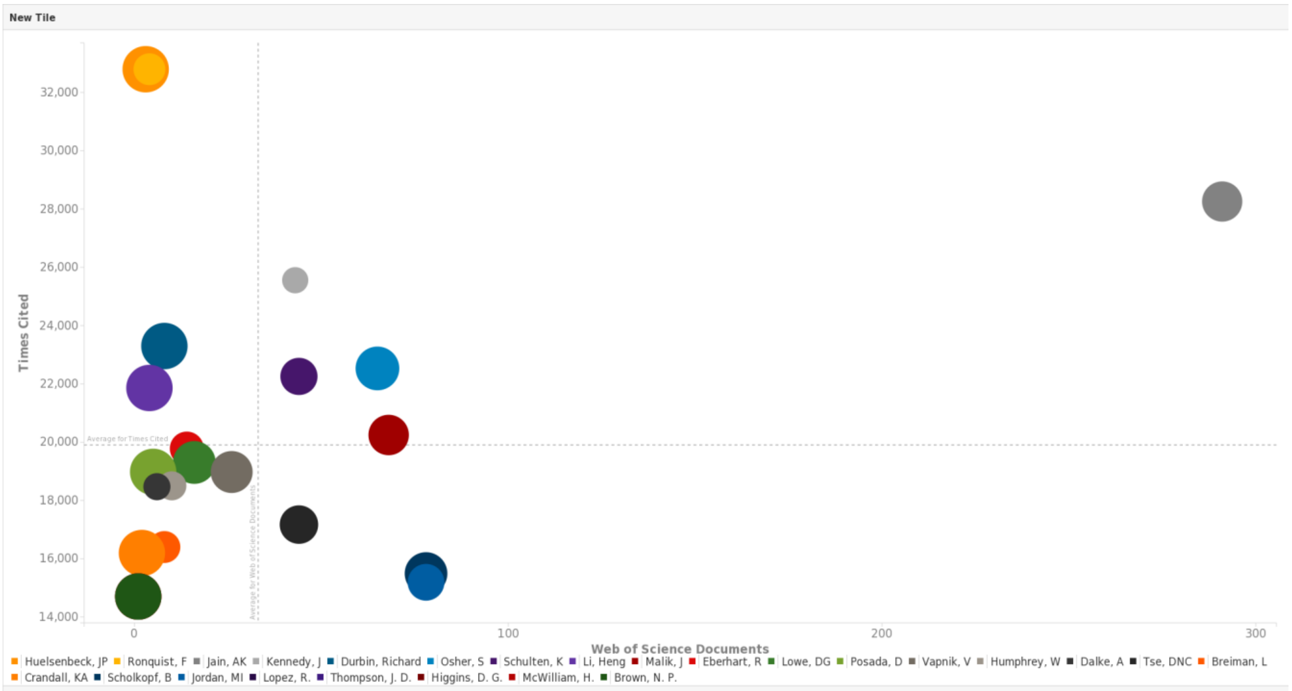


Indicators: Web of Science Documents,Times Cited,% Documents Cited. **Person Name or ID:**NOT [Anonymous].**Schema:**Web of Science.**Research Area:**COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE,COMPUTER SCIENCE, CYBERNETICS,COMPUTER SCIENCE, HARDWARE & ARCHITECTURE,COMPUTER SCIENCE, INFORMATION SYSTEMS,COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS,COMPUTER SCIENCE, SOFTWARE ENGINEERING,COMPUTER SCIENCE, THEORY & METHODS
InCites dataset updated Apr 28, 2018. Includes Web of Science content indexed through Feb 27, 2018. Export Date: May 24, 2018.

[fig 3] – Relations between # of documents and # of citations

But as we will see later this data can be subject to distortion.

We have therefore drawn up again the ranking of the top 25 authors sorted according to the number of times that have been cited and we have noticed how the number of documents that have been written is not predominant.
As we can see from the next graph *[fig 4]* (axes X = Web of Science Documents, axes Y = number of times that the author has been cited, radius = % cited documents) only 3 of the 15 authors cited several times have written more than 50 articles.
Ordering the ranking in this way we also note that the percentage of documents cited by author is almost always higher than 75%, whereas previously (so with the ranking based on the number of written documents was below 50%).

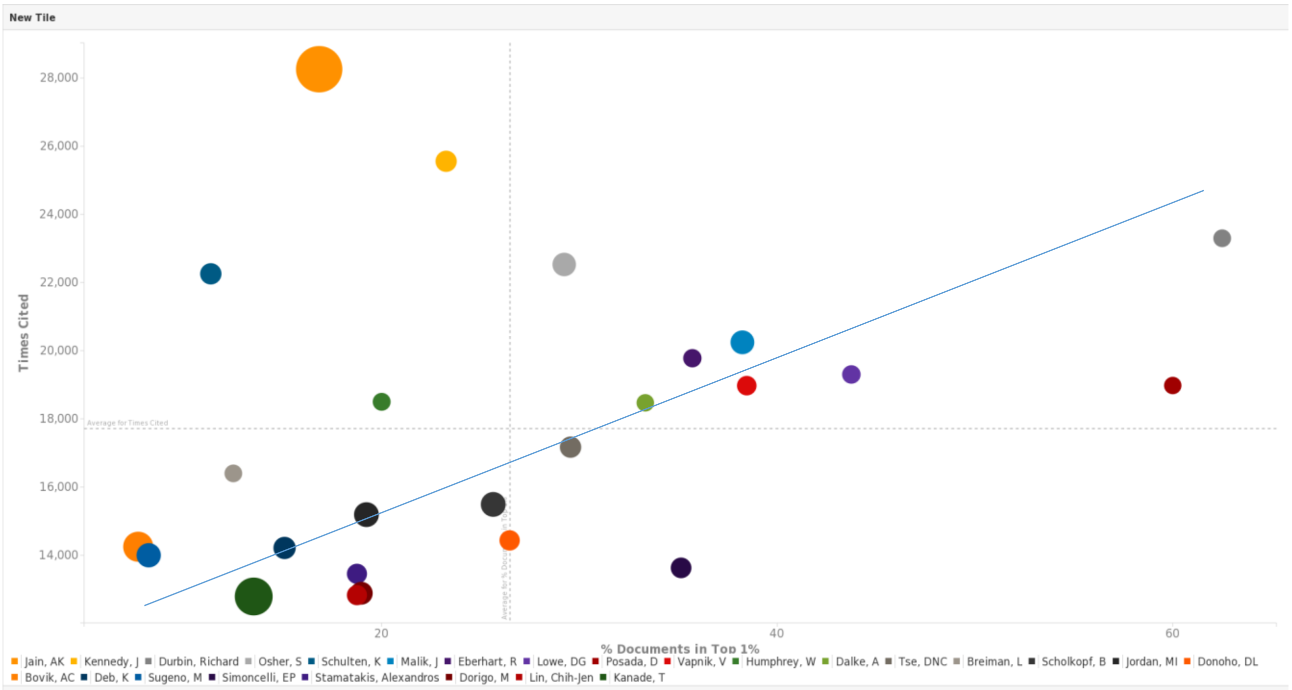


Indicators: Web of Science Documents,Times Cited,% Documents Cited. **Person Name or ID:**NOT [Anonymous].**Schema:**Web of Science.**Research Area:**COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE,COMPUTER SCIENCE, CYBERNETICS,COMPUTER SCIENCE, HARDWARE & ARCHITECTURE,COMPUTER SCIENCE, INFORMATION SYSTEMS,COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS,COMPUTER SCIENCE, SOFTWARE ENGINEERING,COMPUTER SCIENCE, THEORY & METHODS

InCites dataset updated Apr 28, 2018. Includes Web of Science content indexed through Feb 27, 2018. Export Date: May 24, 2018.

[fig 4] – Relations between # of documents and # of citations ordered by time cited

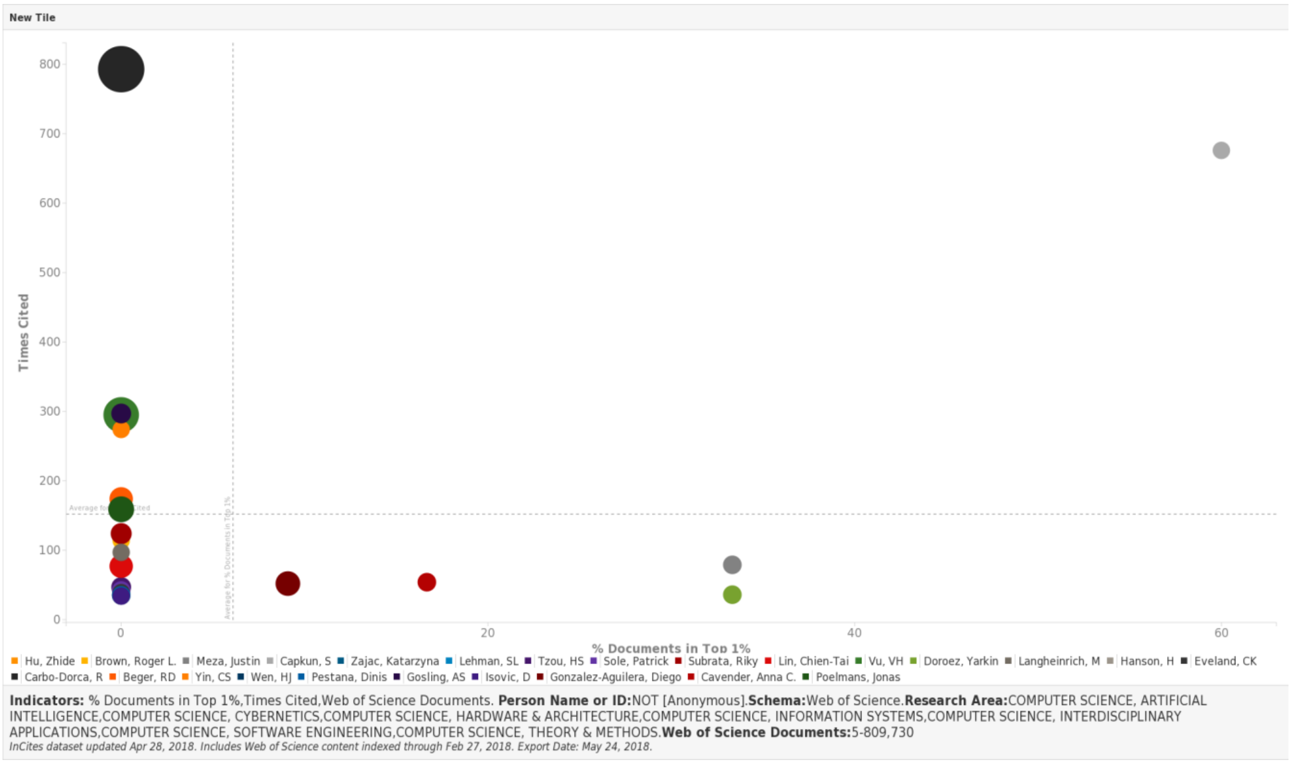
In order to see the presence of the Exchanging Citation we have created these two new graphs (radius = # doc), the first *[fig 5]* ordered according to the time cited while the second *[fig 6]* and more representative ordered according to the percentage of documents cited (taking therefore only the authors with 100% of the documents cited) and omitting in both cases the authors who wrote less than 5 documents.



Indicators: % Documents in Top 1%,Times Cited,Web of Science Documents. **Person Name or ID:**NOT [Anonymous].**Schema:**Web of Science.**Research Area:**COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE,COMPUTER SCIENCE, CYBERNETICS,COMPUTER SCIENCE, HARDWARE & ARCHITECTURE,COMPUTER SCIENCE, INFORMATION SYSTEMS,COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS,COMPUTER SCIENCE, SOFTWARE ENGINEERING,COMPUTER SCIENCE, THEORY & METHODS.**Web of Science Documents:**5-809,730

InCites dataset updated Apr 28, 2018. Includes Web of Science content indexed through Feb 27, 2018. Export Date: May 24, 2018.

[fig 5] – Relations between # time cited and % documents in 1% ordered by time cited



Indicators: % Documents in Top 1%,Times Cited,Web of Science Documents. **Person Name or ID:**NOT [Anonymous].**Schema:**Web of Science.**Research Area:**COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE,COMPUTER SCIENCE, CYBERNETICS,COMPUTER SCIENCE, HARDWARE & ARCHITECTURE,COMPUTER SCIENCE, INFORMATION SYSTEMS,COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS,COMPUTER SCIENCE, SOFTWARE ENGINEERING,COMPUTER SCIENCE, THEORY & METHODS.**Web of Science Documents:**5-809,730
InCites dataset updated Apr 28, 2018. Includes Web of Science content indexed through Feb 27, 2018. Export Date: May 24, 2018.

[fig 6] – Relations between # time cited and % documents in 1% ordered by percentage document cited

From which we can observe how only a few authors cited many times have actually written some very important documents (i.e. belonging to the Percentage of publications in the top 1% based on citations by category, year, and document type), observation also highlighted by the low number of documents that have actually been written.

Data Integration

1.
Scraping from Incites (http://incites.thomsonreuters.com/#/signin) and creating a Json file.
In the People section we applied as filters in the Research Area: Computer Science Artificial Intelligence, Computer Science Cybernetics, Computer Science Hardware & Architecture, Computer Science Information Systems, Computer Science Interdisciplinary Applications, Computer Science Software Engineering and Computer Science Theory & Methods. In this way we selected all the authors of our interests.

2.
Merging this data with the DBLP dataset of Computer Scientists (https://old.datahub.io/dataset/dblp). Obviously, we select only the common information in both datasets.

3.
DBLP
DBLP is a computer science bibliography website. Starting in 1993 at the University of Trier, Germany, it grew from a small collection of HTML files and became an organization hosting a database and logic programming bibliography site. DBLP listed more than 3.66 million journal articles, conference papers, and other publications on computer. All of the important journals on computer science are tracked. Proceedings papers of many conferences are also tracked.

Methodology

Hypothesis

Given the size and structure of the data in question, we decided to adopt a representation using graphs.
The Theory of the Graph constitutes, like the Mathematical Programming, a methodological body for modeling and solving decision problems.

Many optimization problems have a natural graphical representation and efficient resolving techniques based on Graphs, explained in this chapter.

Graphs can be used to model many types of relations and processes in physical, biological, social and information systems. Many practical problems can be represented by graphs. Emphasizing their application to real-world systems, the term network is sometimes defined as a graph in which attributes (e.g. names) are associated with the nodes and/or edges.

In computer science, graphs are used to represent networks of communication, data organization, computational devices and the flow of computation.

Once our graph is built we expect:

<u>1- Clusters between authors who publish together.</u>

this happens because often the people who publish together belong to the same universities or research groups and are therefore more inclined to collaborate with people "close" to them (those that in the graph we will call neighbors).

<u>2- Densification of the clusters.</u>

Authors who have already worked together are more likely to cite each other.

<u>3- Clusters between authors who publish in the same area.</u>

Authors belonging to two different clusters (countries or universities) will find a link between them if they deal with the same topic.

Having selected as research field Computer Science, we expect all these points listed above to happen.

The last one could be less obvious, as there is no real distinction in sub-topics of the subject treated (Computer science).

Networkx

NetworkX is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and function of complex networks. With NetworkX you can load and store networks in standard and nonstandard data formats, generate many types of random and classic networks, analyze network structure, build network models, design new network algorithms, draw networks, and much more. NetworkX provides data structures for graphs (or networks) along with graph algorithms, generators, and drawing tools.

Graph, Node and Edge

A graph is an ordered pair G=(V,E) where,
V is the vertex set whose elements are the vertices, or nodes of the graph.
E is the edge set whose elements are the edges, or connections between vertices, of the graph. If the graph is undirected, individual edges are unordered pairs {u,v} where u and v are vertices in V. If the graph is directed, edges are ordered pairs (u,v).
The order of a graph is the number of vertices in it, usually denoted |G| or n. The size of a graph is the number of edges in it, denoted |E| or m.

Undirected graph

An undirected graph is one in which edges have no orientation. The edge (a, b) is identical to the edge (b, a), i.e., they are not ordered pairs, but sets {u, v} (or 2-multisets) of vertices.

Directed graph

A directed graph or digraph is an ordered pair D = (V, A) with
V a set whose elements are called vertices or nodes, and
A a set of ordered pairs of vertices, called arcs, directed edges, or arrows.
An edge a = (x, y) is considered to be directed from x to y; y is called the head and x is called the tail of the arc; y is said to be a direct successor of x, and x is said to be a direct predecessor of y. If a path leads from x to y, then y is said to be a successor of x and reachable from x, and x is said to be a predecessor of y.

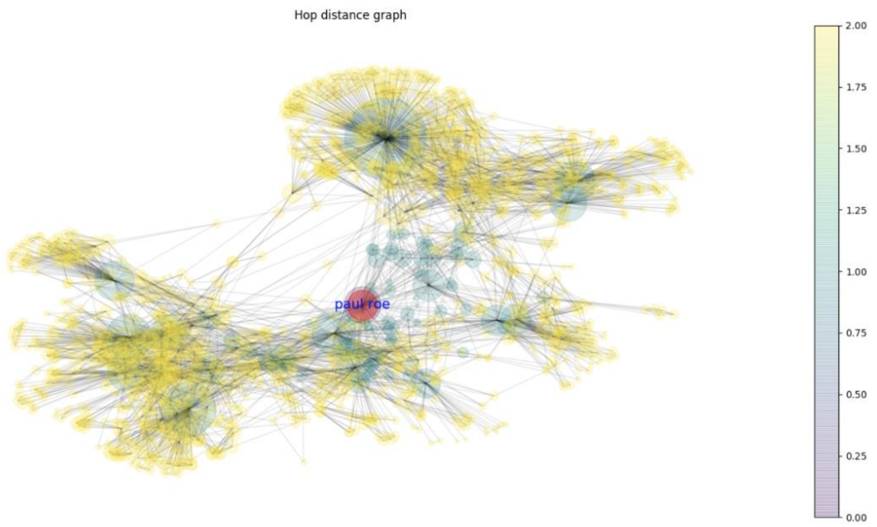
Path

In graph theory, a path in a graph is a finite or infinite sequence of edges which connect a sequence of vertices which are all distinct from one another. A path is a trail in which all vertices (except possibly the first and last) are distinct. A trail is a walk in which all edges are distinct. A walk of length k in a graph is an alternating sequence of vertices and edges, v_0, e_0, v_1, e_1, v_2, ... v_{k-1}, e_{k-1}, v_k, which begins and ends with vertices. If the graph is undirected, then the endpoints of e_i are v_i and v_{i+1}. If the graph is directed, then e_i is an arc from v_i to v_{i+1}.

Shortest Path

In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.

Hop Distance



[fig 7] – Hop Distance

In graph theory, the distance between two vertices in a graph is the number of edges in a shortest path connecting them. If there is no path connecting the two vertices, i.e., if they belong to different connected components, then conventionally the distance is defined as infinite.

Jaccard Similarity

The Jaccard index, also known as Intersection over Union and the Jaccard similarity coefficient, is a statistic used for comparing the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

$$0 \leq J(A,B) \leq 1$$

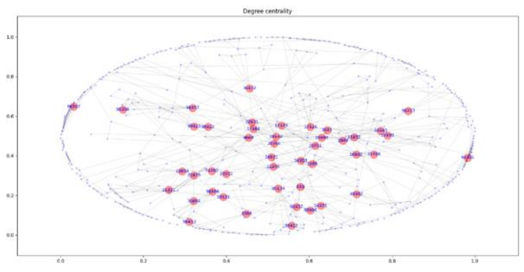
The Jaccard distance, which measures dissimilarity between sample sets, is complementary to the Jaccard coefficient and is obtained by subtracting the Jaccard coefficient from 1, or, equivalently, by dividing the difference of the sizes of the union and the intersection of two sets by the size of the union:

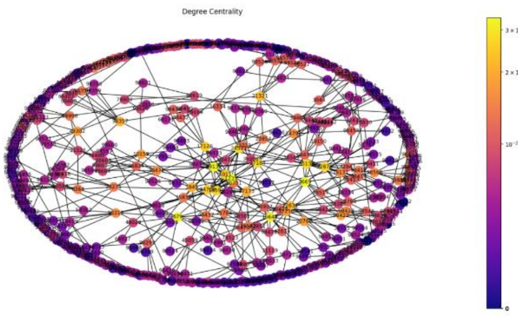
$$d_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

Centrality Measures

In graph theory and network analysis, indicators of centrality identify the most important vertices within a graph. Applications include identifying the most influential person(s) in a social network. Centrality indices are answers to the question "What characterizes an important vertex?" The answer is given in terms of a real-valued function on the vertices of a graph, where the values produced are expected to provide a ranking which identifies the most important nodes.

Degree Centrality





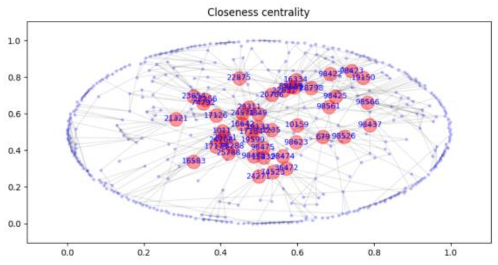
[fig 8] – Degree Centrality

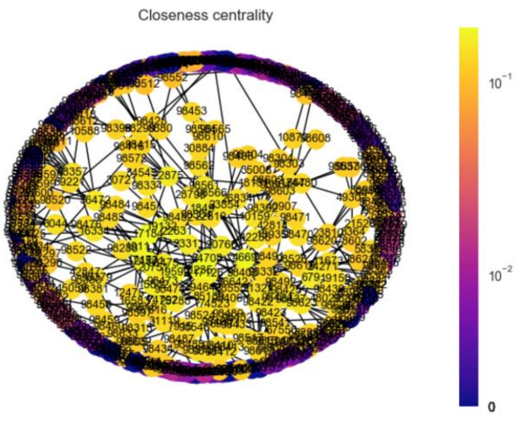
The degree centrality for a node v is the fraction of nodes it is connected to. The degree centrality values are normalized by dividing by the maximum possible degree in a simple graph n-1 where n is the number of nodes in G.

The in-degree centrality for a node v is the fraction of nodes its incoming edges are connected to.

The out-degree centrality for a node v is the fraction of nodes its outgoing edges are connected to.

Closeness Centrality





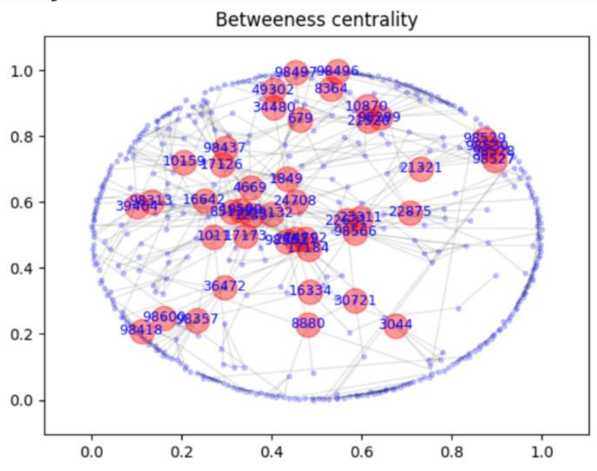
[fig 9] – Closeness Centrality

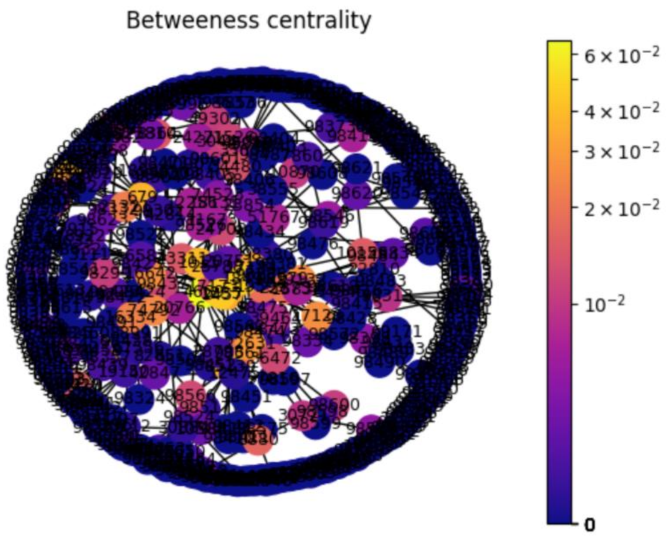
Closeness centrality of a node u is the reciprocal of the sum of the shortest path distances from u to all $n-1$ other nodes. Since the sum of distances depends on the number of nodes in the graph, closeness is normalized by the sum of minimum possible distances $n-1$.

$$C(u) = \frac{n-1}{\sum_{v=1}^{n-1} d(v,u)},$$

where $d(v,u)$ is the shortest-path distance between v and u, and n is the number of nodes of the graph.
Notice that higher values of closeness indicate higher centrality.
The closeness centrality is normalized to $(n-1)/(|G|-1)$ where n is the number of nodes in the connected part of graph containing the node. If the graph is not completely connected, this algorithm computes the closeness centrality for each connected part separately.

Betweenness Centrality





[fig 10] - Betweeness Centrality

Betweenness centrality of a node v is the sum of the fraction of all-pairs shortest paths that pass through v:

$$c_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

where V is the set of nodes, $\sigma(s,t)$ is the number of shortest (s,t)-paths, and $\sigma(s,t|v)$ is the number of those paths passing through some node v other than s,t. If $s=t$, $\sigma(s,t)=1$, and if $v \in s,t$, $\sigma(s,t|v)=0$.

Induced Subgraph

In graph theory, an induced subgraph of a graph is another graph, formed from a subset of the vertices of the graph and all of the edges connecting pairs of vertices in that subset. Formally, let G = (V, E) be any graph, and let S ⊂ V be any subset of vertices of G. Then the induced subgraph G[S] is the graph whose vertex set is S and whose edge set consists of all of the edges in E that have both endpoints in S.

Dijkstra’s Algorithm

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph. For a given source node in the graph, the algorithm finds the shortest path between that node and every other. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined.
Let the node at which we are starting to be called the initial node. Let the distance of node Y be the distance from the initial node to Y. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Mark all nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.
2. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes. Set the initial node as current.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances through the current node. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B through A will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.
4. When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.
5. Move to the next unvisited node with the smallest tentative distance and repeat the above steps which check neighbors and mark visited.
6. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
7. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

Erdos Number
The Erdős number describes the "collaborative distance" between two people. To be assigned an Erdős number, someone must be a coauthor of a research paper with another person who has a finite Erdős number. Paul Erdős has an Erdős number of zero. Anybody else's Erdős number is k + 1 where k is the lowest Erdős number of any coauthor. Erdős wrote around 1,500 mathematical articles in his lifetime, mostly co-written. He had 511 direct collaborators; these are the people with Erdős number 1. The people who have collaborated with them (but not with Erdős himself) have an Erdős number of 2 (9267 people as of 2010[9]), those who have collaborated with people who have an Erdős number of 2 (but not with Erdős or anyone with an Erdős number of 1) have an Erdős number of 3, and so forth. A person with no such co-authorship chain connecting to Erdős has an Erdős number of infinity (or an undefined one). Since the death of Paul Erdős, the lowest Erdős number that a new researcher can obtain is 2.

Resulting and Comments

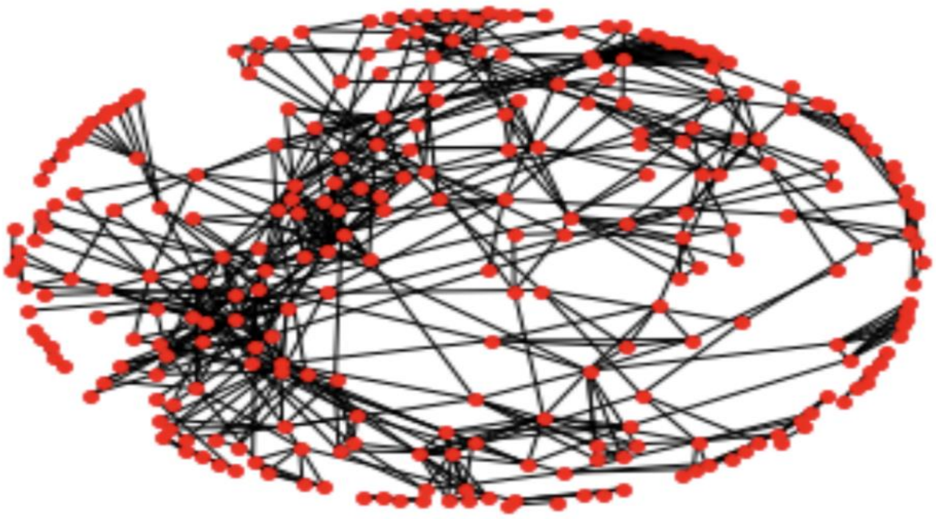
1.
Code
The JSON file that we created has this structure:

```
[{"authors": [
{"author": first name and last name of author1,
"author_id": author1 id},
{"author": first name and last name of author2,
"author_id": author2 id},…],
"id_conference": conference id name,
"id_conference_int": conference id,
"id_publication": publication id name,
"id_publication_int": publication id,
"title": title of the publication},…
```

Here we have a list of dictionaries where each one of them represents a publication. For each publication we have information on its title, name id, number id, the conference name and id in which it was presented and the authors' complete name and id that wrote the publication.
The idea is to create 3 back-up structure (dictionaries), used as support while executing the script. In details these are the structures for each dictionary:

```
CONFERENCES: {"id conf int" : {"conf": id conf, "title": title conf, "pub_int": [ id pub_int_1,
id pub_int_2, id pub_int_3, .....]}}
AUTHORS: { " aut_id" : { "name": author name, "pub":[ pub_int_1, pub_int_2, pub_int_3,....]}}
PUBLICATIONS: { "id pub int" : { "pub":id_pub, "authors":[aut _id _1, aut _id_2, aut_id_3,...]}}
```

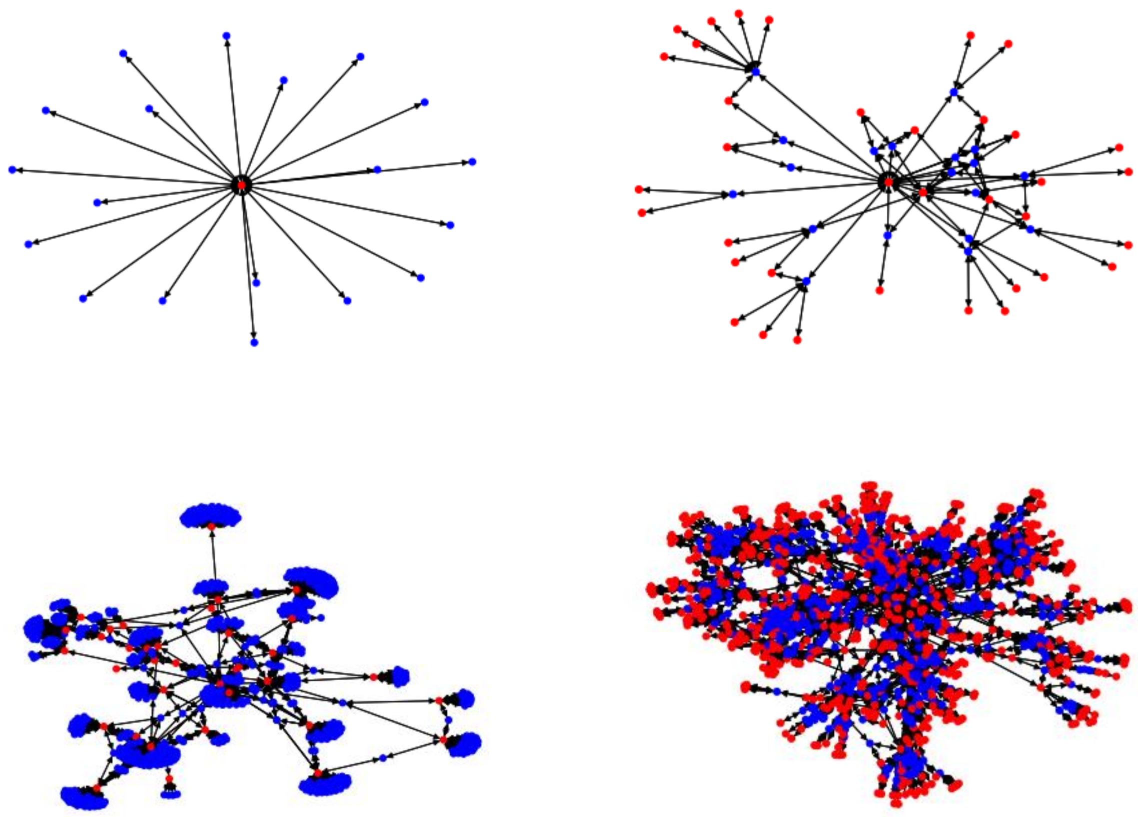
2.
At this point after processing the JSON file we **created a** graph *[fig 11]*, G, whose nodes are authors and publications. Every authors are connected to his publications by an undirected edge.



[fig 11] – The Graph

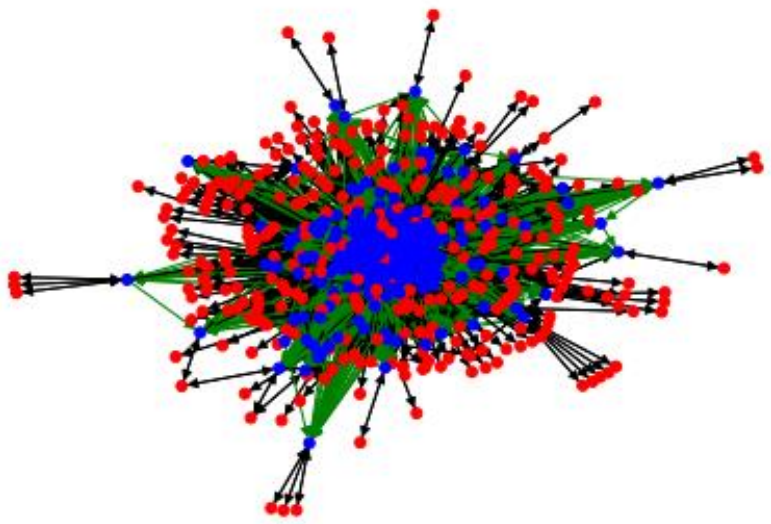
To do this we started looking in each row line (first FOR loop) and picking from the raw data the values under the id conference int and id publication int labels and stored as key of the dictionaries conferences and publications respectively. Those values were chosen for their unicity and for a faster research in our back-up dictionaries. Stored the keys the dictionaries are now filled: values under id conference, title, and pub int labels of the row data are stored as values of conf, title and pub int in conference dictionary; id publication values are stored in pub key in publications dictionary together with an empty list titled authors. Scraping the row data, we focus now to the authors section, adding each author to the previous empty list. We want to fill the authors dictionary as well, so we need first an IF condition to check if the author already exists as key or not and then choose if create a new instance for the author (if not already present) or just update it (the instance) with a new publication. If it's a new entry we create also a new node of the graph G with the aut id. Once executed the first for loop we have, until now, the three dictionaries and only the nodes of the final graph. The edges in the graph link only authors that share at least one publication; for this reason, we rely on the second FOR loop to get inside the publications dictionary created before to take a look at the authors that participate at the same publication. If more than one author worked to the publication we start focusing on the list of the authors. Given that authors are already sharing one publication (exactly the publication we are looking at) it'll be created an edge that link each writer to the other ones. However, the edge between two authors will be weighted, so taken two authors we start looking at their publication using the authors dictionary and, once got into the list of publication we calculate the Jaccard similarity between the list. The result of the Jaccard similarity it'll be the weight of the edge.

To get a more detailed description we have included the publications in the graph [fig 12].



[fig 12] - Subgraph evolutions according to Hop Distance

Then we added a directed edge between two publications when one cited the other. This operation was done only for a subgraph of the original one, due to time and complexity issues *[fig 13]*.



[fig 13] - Subgraph evolutions according to Hop Distance with citations

3.
We created a Python function where:
we ask the user to type the query, in particular, in the first case, we ask for an id of the conference in order to retrieve a subgraph induced of the authors that have published at least one work in the given conference, and in the second case, we ask for an id author and a d value in order to execute the algorithm and get the subgraph induced of all the authors linked to the input author that have at least hop distance equal to d. First case. Once acquired the query typed by the user we start looking inside the conferences dictionary accessing into the values of the given conference and we focus in ”pub int” to see which are the publications discussed in our conference. As we are interested in the authors that took part at each publication, and consequently at the conference as well, we dig a little bit down to the publications dictionary (in particular to the list of the authors with the second for loop) once acquired the id of the publication in the first for loop. All the authors that satisfy the request are stored in a list of authors; list then used to create the subgraph of G using the appropriate method G.subgraph. To plot the result we get, with a for loop, the degree of each node of the subgraph that we use then in the .draw method to plot the network putting in evidence the nodes linking their size to their degree. To get the degree, closeness and betweenness centrality we used the built-in function of Networkx package (.degree centrality, .closeness centrality, .betweenness centrality respectively) and plotted the results using the plt.histogram. Second case. Once acquired the author id and the hop distance typed by the user we create the subgraph induced as explained in the introduction. To deal with this task we used another built-in function, .egograph, that take as variable the original graph, the hop distance and radius. Again, we rely on the degree of each node to draw properly the graph adapting the size of each node to the degree.

4.
We implemented a function that replicates Dijkstra algorithm for shortest path. This function takes as input a source node and a target node, and is divided into 3 steps: Step 1) This is the instancing step: G neighbors: a matrix that stores, for every node, its neighbours and the weight of each edge reached: a boolean variable, set to False, that represents if the target node is reached. temp distances: a dict that contains, for each node, the temporary weight of the shortest path from the source (for source is 0). final distances: a dict that contains the weight of shortest path from the source. It starts with the initial node, with a value of 0. t distances: a heap with tuples, in the way (weight of path from source, node). It starts with all the neighbours of source node and their corresponding edge weight. J: a dict, for every node it stores its predecessor in the shortest path. It starts with None values for every node in the graph, 0 for source and source for every neighbours of source node. Step 2) Find the nearest node: In this step, an element is popped out from t distances (surely it is the one with the minimum distance). If the node related has been already been processed (is in final distances keys), another element is taken from t distances. When a proper node is found, its final distance is the first element of the tuple; if its the target, the algorithm stops. Step 3) Update distances: For every node neighbour of the node selected in step 2 that has not been processed, it is computed its distances from source going through step 2-node. If it’s lower than the one in temp distances, this is updated, along with its predecessor in J, and weight of the path and node are pushed into the heap. Step 2 is then repeated. Final Step): If target node was not reached, ’PATH NOT FOUND’ is printed. Otherwise, path from source to target and its weight are returned as a tuple. Only the weight of the shortest path is printed. There is also a check before the execution of the function, to verify that author id of the source node is in the database and is not equal to the target node.

5.
We created another function that is simply a modified version of the function used in the previous point. This new function, named multi source shortest path alg, takes as argument a list of sources (these are the input nodes). In Step 1, the difference is that t distances are pushed the neighbours of all the sources and their corresponding distance from them (if any node is neighbour of more than one source the nearest source is its predecessor). Step 2 and 3 need no changes. A dict is returned, the keys are all the nodes in the graphs and the values are tuples where the first element is the shortest path, and the second is the weight of said path. This way the algorithm feels more complete. Before applying the function, it is check if all the input nodes are in the graph. It was also feasible to write a single function for both points, but we felt the code to be more readable this way.

Conclusions

How to solve the two problems of distortion that we found and analyzed?
Since there is not a single answer we have therefore tried to elaborate our own theory which could in any case be reviewed in an empirical way by associating the weights of the edges in a different way.
We always start from the graph built and analyzed previously and then re-map the weight of the various quotation arches:
In the simplest case of self-citations, the weight is assigned to that particular edge

- 0.5 for the first self-citation
- 0.25 for the second self-citation
- 0 for all subsequent self-citation

At first, we also thought to implement a negative measure of penalization for all cases of abuse, by removing 0.25 for all self-citation that exceed 70% of the author's overall work from the last work done (if an author has written for example 10 articles, it might make little sense to find in his last publication 7 citations of his previous works).

Finally, we decided not to use this negative measure as it could be too penalizing at a "reputational" level for the continuation of discoveries made but which may have developments not yet covered (even by the same author).

The most complex case is certainly the exchanging citations or how to prevent the exchange of citations even if these do not have relevance with the treated, with the sole aim of artificially increasing the influence of the two authors.
Also, in this case we decided to remap the weights of the various edges of the citations of our graph, starting by identifying all those authors who have mentioned each other (and therefore the edges of the direct graph will have both directions), at this point we assign weight

- 1 for the first quote

Then we check the dates of the other citations, if the interval is less than 2 years (and is therefore more likely to be an exchange of favors), we could also include more complex conditions like the university to which they belong to and/or the country where they operate in.

- We reduce exponentially or proportionally the weights of the edges of the following citations

Remapping the graph in this way we should be able to solve the two problems of distortion that we have detected and analyzed, in fact the sum of citations, having no more unit weight, and being "normalized" will certainly be lower than before which should lead to having a ranking that respects the effective importance of an author in a more congruous manner.

Bibliography:

- https://en.wikipedia.org/wiki/DBLP
- http://chadwick.library.emory.edu/elearning-solutions/WebPrimer-SC/01-Introduction-B.html
- Greenberg SA. How citation distortions create unfounded authority: analysis of a citation network. BMJ. 2009
- https://networkx.github.io/documentation/networkx-1.10/index.html
- https://en.wikibooks.org/wiki/Graph_Theory/Definitions
- https://en.wikipedia.org/wiki/Path_(graph_theory)
- https://en.wikipedia.org/wiki/Distance_(graph_theory)
- https://en.wikipedia.org/wiki/Jaccard_index
- https://en.wikipedia.org/wiki/Centrality
- https://networkx.github.io/documentation/networkx-1.10/reference/algorithms.centrality.html?highlight=centrality
- https://en.wikipedia.org/wiki/Induced_subgraph#Definition
- https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
- http://www.pnas.org/content/101/suppl_1/5183.full
- https://link.springer.com/article/10.1007/s11192-005-0255-6
- http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0039464
- https://www.sciencedirect.com/science/article/pii/S0048733397000462
- https://ieeexplore.ieee.org/abstract/document/1672233/
- https://books.google.it/books?hl=it&lr=&id=5G4QBwAAQBAJ&oi=fnd&pg=PR15&dq=graph+theory+applications&ots=cVpZ1U3xbC&sig=ESFTYPniDGTZRrtsDKphYAaX8s0#v=onepage&q=graph%20theory%20applications&f=false
  http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.721.3161&rep=rep1&type=pdf
- https://www.bmj.com/content/339/bmj.b2680.full.pdf+html
- https://link.springer.com/article/10.1007/s11625-007-0027-8
- https://books.google.it/books?hl=it&lr=&id=nniSQ7ygxsEC&oi=fnd&pg=PA34&dq=erdos+number+graph&ots=Eb_emnDSwj&sig=ZxBJTPGApLGZJYUZFk9-Bp8JWfg#v=onepage&q=erdos%20number%20graph&f=false
  http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=F9800233F0ACF0DD75F5E1ACA310DDC3?doi=10.1.1.210.6669&rep=rep1&type=pdf