Computer Scientist Network Analysis by Using Graph methods

Amir Abbasinejad, Adriano Fragomeni, Sara Vozzella

I. Introduction

In this project we have carried out An analysis of a network of computer scientists, using data on their publications and the conferences they attended. The datasets used are the reduced dblp dataset for testing and debugging, and the full dataset for the analysis. Plots and graphs have been computed using the reduced dataset, in order to provide clearer visuals.

II. Graph Creation

The goal of the project was to create a weighted graph from the data, with the authors represented as nodes and their common publications as the edges. The weight of each edge was calculated by using the Jaccard Similarity formula:

$$w(a_1, a_2) = 1 - J(p_1, p_2)$$

where a_1, a_2 are authors, p_1 and p_2 are the set of publication of the two authors and, $J(p_1, p_2)$ represents the jaccard similarity between these two sets of publications.
The value of the weight of each edge is in the interval $[0, 1)$. The value 1 has been excluded, since it would mean that the nodes do not have a common publication (and thus no edge between them). The value of w would be strongly less than one.
The formula of Jaccard similarity is:

$$J(p_1, p_2) = \frac{p_1 \cap p_2}{p_1 \cup p_2}$$

III. Statistics

In this section we have analyzed and studied three types of centrality through our graph.
The Degree Centrality is a measurement based on the number of edges of each node. Two types of Degree Centrality are usually distinguished based on the in- and out-degree of each node, though since the graph is directed only the absolute value of the number of edges has been considered.
Closeness centrality of a node is the average length of the shortest path between the node and all other nodes in the graph.
Betweenness is a centrality measure of a vertex within a graph. This measure quantifies the number of times a node acts as a bridge along the shortest path between two other nodes.
Obtained results, from this analysis, will illustrated below:

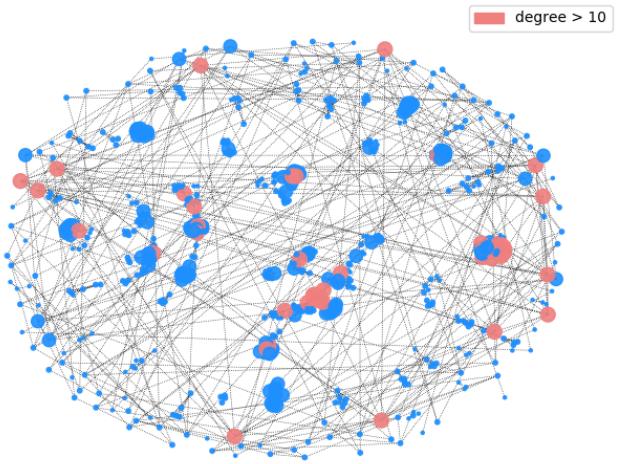

Fig. 1. **Sub-graph of the authors who participated at conference number 3345.**

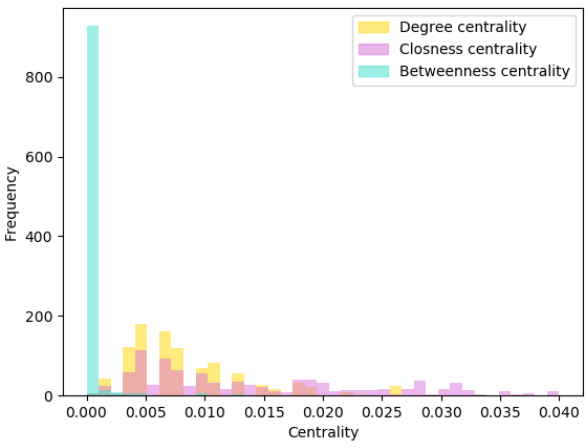

Fig. 2. **Statistics about sub-graph of the conference 3345.**

Fig. 1 shows all the participant of a conference, in this particular graph we analyze conference number

3345. The size of the nodes represents the number of the edges of each node, in particular the size of the node increase by increasing the number of edges that each node has.

Fig. 2 shows three types of centrality measurement for the sub-graph of the *Fig. 1*, and every value is normalized by $n - 1$.
It is clear that the betweenness centrality has a peak of zero value, because in this graph there are few nodes which act as a bridge. Furthermore, since the graph is undirected, the Closeness and Degree centralities are correlated, and thus show the same behaviour.

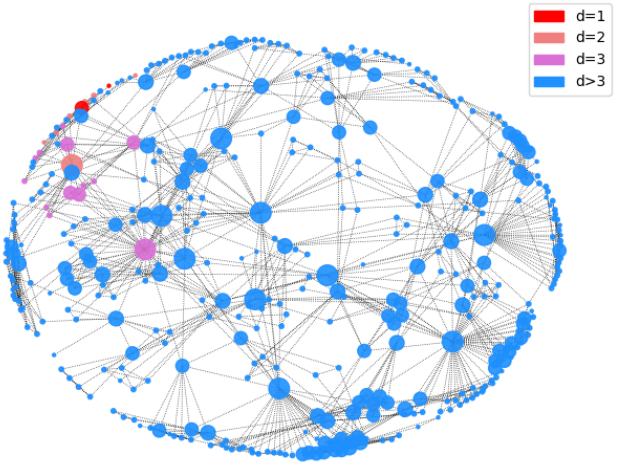

Fig. 3. **Sub-graph of level of hop distance from 18534 author id.**

In this part of the analysis the graph is unweighted and only the steps from the main node are considered. The number of nodes of the subgraph obtained present and exponential behaviour depending on the number of hops input by the user.

IV. Compute some generalized version of the Erdos number

As a last part of the project we have done a generalized version of Erdos number by using Dijkstra's algorithm, to calculate shortest path between two nodes, because every weight is more that zero and there is only one starter node. By using this approach we achieve very competitive time complexity to calculate the shortest path between on starter node and all the other nodes of the graph.

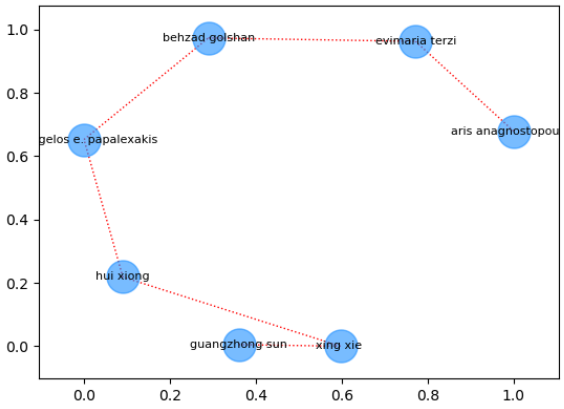

Fig. 4. **The shortest path between Guangzhong Sun and Aris Anagnostopoulos.**

This algorithm has been implemented using *heap* data structure because the time complexity is more optimized, more precisely this data structure is based on tree and priority queue, however in this case since we use min-heap our tree is minimum tree, hence we have minimum value at the root of the tree, so in each iteration we pop the root then we have new minimum in root and so on, until it reaches the node "Aris". Using a heap structure, the running time of the algorithm becomes $O((E * log(V)) + (V * log(V)))$ in the worst case scenario. As a matter of fact, the extract-min operation, as well as the decrease-key operation take time $O(log(V))$ as they present V operations and at most E operations respectively. Since it is reasonable to assume that $E > V$, the running time will be thus written $O(E * log(V))$.
Where V are verteces and E edges.
In the last point we have calculated the GroupNumber for each node of the graph, defined as:

$$GroupNumber(v) = \min_{u \in I}\{ShortestPath(v, u)\}$$

where v is a node in the graph and I is the set of input nodes.
One of the possible interpretation of the group number results is a clustering of the nodes based on distance between all nodes of the graph and the input set.
Since the table was too big it was impossible to visualize The result of this part, however it is possible to illustrate the result by running the code.

A possible way to avoid computing the shortest path algorithm each time to identify sub-paths between nodes would be to store the paths between

a node from the graph and a node from the set and their corresponding weights, and then referring back to the stored sub-paths when the same nodes appear again in the calculation. In this way, it would be possible to increase the speed of the computations, as the computer would already have shortest sub-paths saved in memory and it would only need to compute the difference in weight when analyzing other paths between other nodes. The rationale behind this is that since the algorithm used is time-consuming, by storing shorter distances the programme would not have to compute the same calculations each time.