Graph Anonymization:
Degree Anonymization with Dynamic Programming and Realizability of Degree Sequences with Constraints

TERRACCIANO Vincenzo
USAI Giacomo

2019/2020

Starting Point

DYNAMIC PROGRAM

Graph Anonymization Problem

1. First, starting from $\mathbf{d}$, we construct a new degree sequence $\widehat{\mathbf{d}}$ that is k-anonymous and such that the *degree-anonymization* cost

$$\mathrm{DA}(\widehat{\mathbf{d}}, \mathbf{d}) = L_1(\widehat{\mathbf{d}} - \mathbf{d}),$$

is minimized.

2. Given the new degree sequence $\widehat{\mathbf{d}}$, we then construct a graph $\widehat{G}(V, \widehat{E})$ such that $\mathbf{d}_{\widehat{G}} = \widehat{\mathbf{d}}$ and $\widehat{E} \cap E = E$ (or $\widehat{E} \cap E \approx E$ in the relaxed version).

$$L_1\left(\widehat{\mathbf{d}} - \mathbf{d}\right) = \sum_i \left|\widehat{\mathbf{d}}(i) - \mathbf{d}(i)\right|$$

Dynamic Programming Algorithm

for $i < 2k$,

$$\mathrm{DA}\left(\mathbf{d}[1, i]\right) = I\left(\mathbf{d}[1, i]\right). \tag{2}$$

For $i \geq 2k$,

$$\mathrm{DA}\left(\mathbf{d}[1, i]\right) = \min \left\{ \min_{k \leq t \leq i-k} \left\{ \mathrm{DA}\left(\mathbf{d}[1, t]\right) + I\left(\mathbf{d}[t+1, i]\right) \right\}, I\left(\mathbf{d}[1, i]\right) \right\}. \tag{3}$$

$$I\left(\mathbf{d}[i, j]\right) = \sum_{\ell=i}^{j} \left(\mathbf{d}(i) - \mathbf{d}(\ell)\right)$$

What do we want to do?

- We will solve the problem with DP algorithm and we will show how to improve the running time complexity of the DP algorithm from $O(n^2)$ to $O(nk)$.
- The core idea for this speedup lies in the simple observation that no anonymous group should be of size larger than $2k - 1$. If any group is larger than or equal to $2k$, it can be broken into two subgroups with equal or lower overall degree-anonymization cost.

RECURSION CAN BE WRITTEN AS FALLOWS

$$\textsc{Da}\left(\mathbf{d}[1,i]\right) = \min_{\max\{k,i-2k+1\}\le t\le i-k}\left\{\textsc{Da}\left(\mathbf{d}[1,t]\right) + I\left(\mathbf{d}[t+1,i]\right)\right\}$$

- The substantial difference between **Greedy** and **Dynamic Programming** is the approach adopted:

  Greedy algorithm makes an optimal local choice;

  Dynamic Programming algorithm makes a choice based on the one considered above

- This also affects the optimization of the set of edges obtained.

- Our solution is tested on a fake Dataset of graph friend (in addition, we used the datasets saw in class) because with real Dataset the greedy algorithm doesnt'work.

```
2020-01-17 13:05:54.711 | INFO     | __main__:<module>:214 - Start compute greedy alghoritm
2020-01-17 13:05:54.729 | INFO     | __main__:<module>:217 - Finish compute greedy alghoritm
2020-01-17 13:05:54.729 | INFO     | __main__:<module>:220 - Start compute dp alghoritm
8 Dataset/trialdataset/graph_friend_1000_10_100.csv
Number of edges of degree alghoritm:50556
Number of edges of dp alghoritm:50831
Number of edges of greedy alghoritm:50886
2020-01-17 13:05:54.766 | INFO     | __main__:<module>:222 - Finish compute dp alghoritm
```

```
6 Dataset/trialdataset/graph_friend_100_10_100.csv
Array of degrees sorted (array_degrees_greedy) : [97 97 97 97 97 97 76 76 76 76 76 76 72 72 72 72 72 72 64 64 64 64 64 64
 64 62 62 62 62 62 62 57 57 57 57 57 57 55 55 55 55 55 55 55 51 51 51 51
 51 51 49 49 49 49 49 49 49 44 44 44 44 44 44 42 42 42 42 42 42 38 38 38
 38 38 38 34 34 34 34 34 34 30 30 30 30 30 30 30 23 23 23 23 23 23 14 14
 14 14 14 14]
Array of dp sorted (vertex_degree_dp) : [97 97 97 97 97 76 76 76 76 76 76 76 76 76 65 65 65 65 65 65 64 64 64 64
 64 64 59 59 59 59 59 59 56 56 56 56 56 56 53 53 53 53 53 53 51 51 51 51
 51 51 48 48 48 48 48 48 44 44 44 44 44 44 42 42 42 42 42 42 38 38 38 38
 38 38 34 34 34 34 34 34 30 30 30 30 30 30 25 25 25 25 25 25 14 14 14 14
 14 14 14 14]
Number of edges of degree alghoritm:4820
Number of edges of dp alghoritm:4935
Number of edges of greedy alghoritm:5046
2020-01-17 13:07:37.879 | INFO     | __main__:<module>:215 - Start compute greedy alghoritm
2020-01-17 13:07:37.880 | INFO     | __main__:<module>:218 - Finish compute greedy alghoritm
2020-01-17 13:07:37.880 | INFO     | __main__:<module>:221 - Start compute dp alghoritm
2020-01-17 13:07:37.882 | INFO     | __main__:<module>:223 - Finish compute dp alghoritm
```

*In this case, we don’t use the Construct Graph for create a Graph, but we show only the degree anonymization

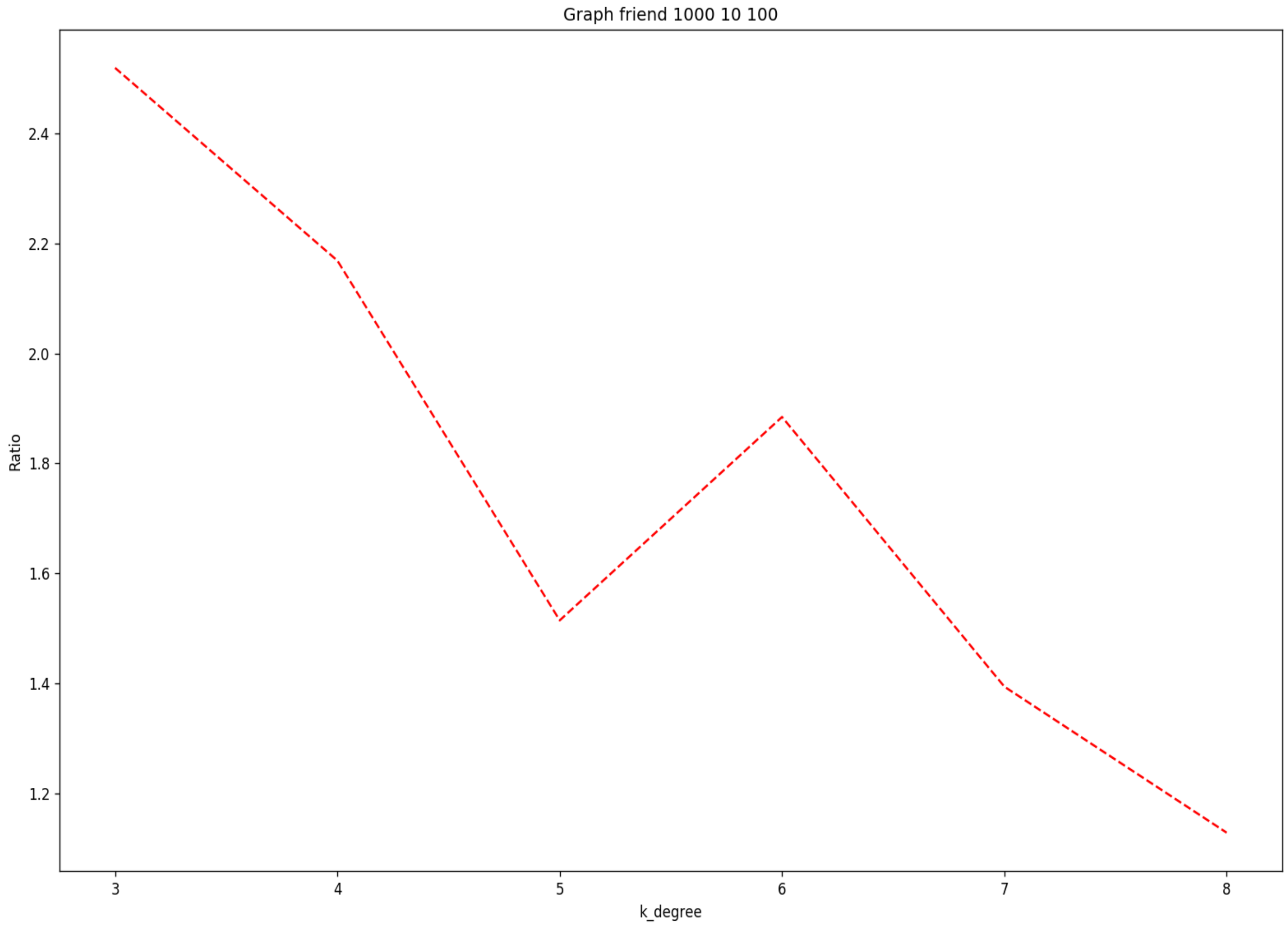
RESULT

- *Greedy* algorithm is a linear-time algorithm but this algorithm is not guaranteed to find the optimal anonymization of the input sequence. Our experiments show that it performs extremely well in practice, achieving anonymizations with costs very close to the optimal.
- The *greedy* algorithm presents some computational problems with high k values.
- By testing with the values that return a correct result, we have found that the greedy algorithm is much more powerful than the dynamic programming, even if the latter is still fast enough.
- *Greedy* algorithm is on the order of hundredths of a second, the second is on the order of tenths of a second.
- This higher cost is due to the construction of the indexes for the array at each step with the values to be considered for anonymization.

METRIC

- We report the results in terms of the performance ratio R which is the ratio of the cost of the solution obtained by the Greedy algorithm to the optimal cost obtained by the DP algorithm.
- This is $R = \frac{L_1(\hat{d}_{greedy} - d)}{L_1(\hat{d}_{dp} - d)}$, where d is the input degree sequence; $\hat{d}_{greedy}$ and $\hat{d}_{dp}$ are the k-anonymous degree sequences output by the Greedy and the DP algorithms respectively. L_1 is a norm of differences between two vector.
- The values of R close to 1 imply that the two algorithms achieve exactly the same cost, in which case Greedy performs optimally. The closer R is to 1, and better is the performance of the Greedy algorithm.

Created with fake dataset : Graph friend with 1000 nodes, minimum 10 edges and maximum 100 edges for node



Values of R close to 1 imply that the two algorithms achieve exactly the same cost, in which case Greedy performs optimally. The closer R is to 1, the better the performance of the Greedy algorithm.

Second Point

Graph construction with ***Constraints***

ConstructGraph Algorithm

Algorithm 1 The ConstructGraph algorithm.

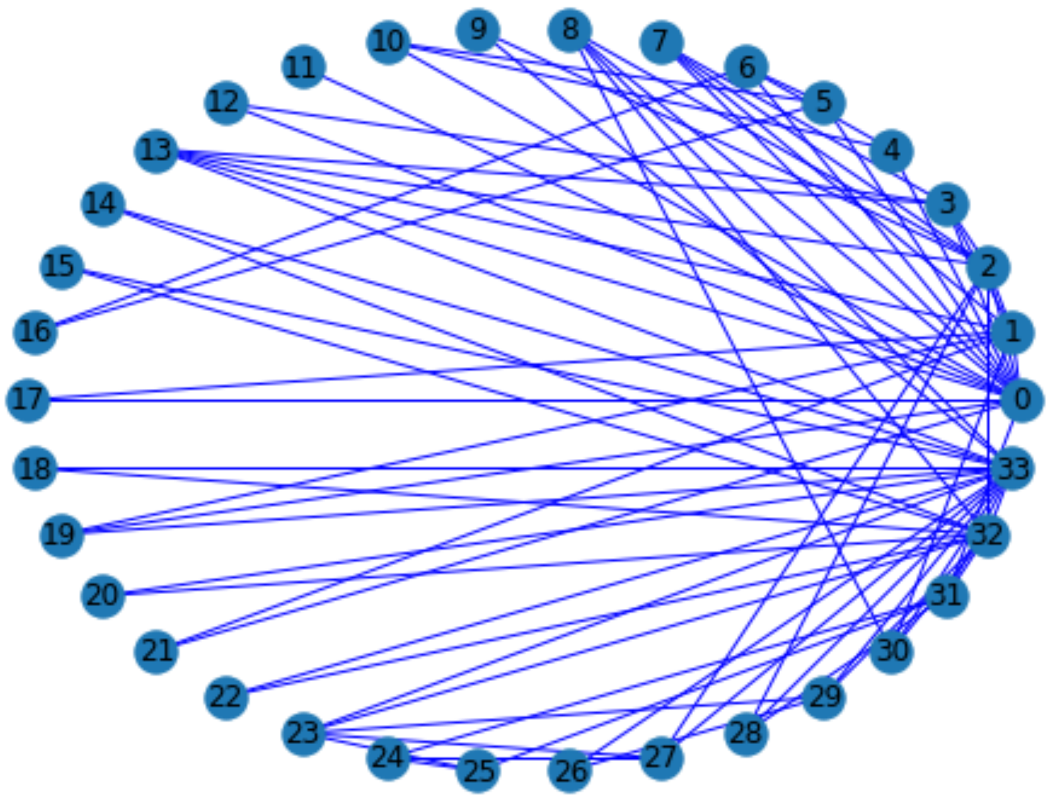
Input: A degree sequence $\mathbf{d}$ of length n.
Output: A graph $G(V, E)$ with nodes having degree sequence $\mathbf{d}$ or "No" if the input sequence is not realizable.

$V \leftarrow \{1, \ldots, n\}$, $E \leftarrow \emptyset$
if $\sum_i \mathbf{d}(i)$ is odd **then**
    Halt and return "No"
while 1 **do**
    if there exists $\mathbf{d}(i)$ such that $\mathbf{d}(i) < 0$ **then**
        Halt and return "No"
    if the sequence $\mathbf{d}$ are all zeros **then**
        Halt and return $G(V, E)$
    Pick a random node v with $\mathbf{d}(v) > 0$
    Set $\mathbf{d}(v) = 0$
    $V_{\mathbf{d}(v)} \leftarrow$ the $\mathbf{d}(v)$-highest entries in $\mathbf{d}$ (other than v)
    for each node $w \in V_{\mathbf{d}(v)}$ **do**
        $E \leftarrow E \cup (v, w)$
        $\mathbf{d}(w) \leftarrow \mathbf{d}(w) - 1$

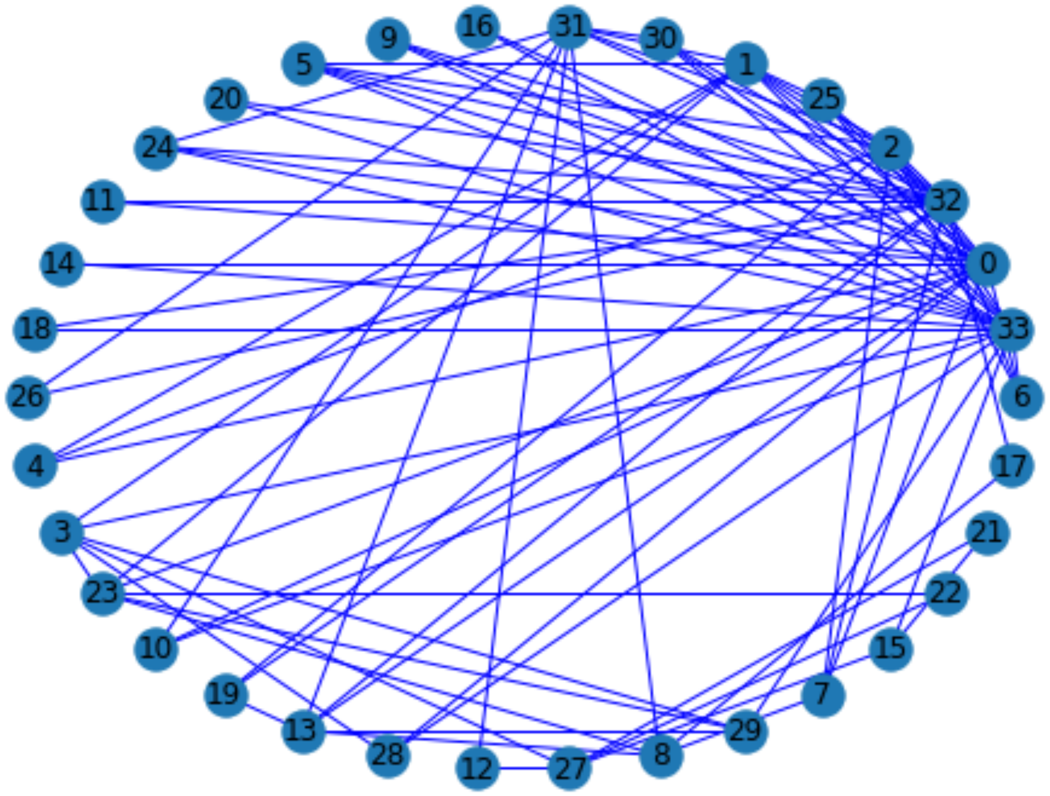
In class we saw the algorithm on the left. This algorithm allows to create a graph with the same nodes (label with number instead real name) and for each node consider the degree of anonymization and add edge with no constraints.

IS IT POSSIBLE CREATE AN ANONYMIZATION GRAPH WITH THE START EDGE?

Starting graph



Graph create with ConstructGraph algorithm



*this image are made on no real dataset (karate_club_graph is a graph inside networkX library of Pyhton) because on a real dataset is more complex understand the different (too edges!)

ASSUMPTIONS

Given input graph G(V,E), we say that degree sequence $\hat{d}$ is realizable subject to G , if and only if there exists a simple graph G~(V,E~) whose nodes have precisely the degrees suggested by $\hat{d}$ and E $\subseteq$ E~.

Consider :

- the vector a = $\hat{d}$- d;
- the set V_l that is an *ordered set of l* nodes with l largest a(i) values, sorted in decreasing order;
- $d^l(i)$ is the degree of node i in the input graph G when counting only edges in G that connect node i to one of the nodes in V_l;
- l = | V_l | is a number prior the we decide.

If $\sum_{i \in V_\ell} \mathbf{a}(i)$ **is even** **&&** $$\sum_{i \in V_\ell} \mathbf{a}(i) \leq \sum_{i \in V_\ell} \left(\ell - 1 - \mathbf{d}^\ell(i)\right) + \sum_{i \in V - V_\ell} \min\{\ell - \mathbf{d}^\ell(i), \mathbf{a}(i)\},$$

We can consider the **Supergraph** algorithm

The inputs to the `Supergraph` are the original graph G and the desired k-anonymous degree distribution $\widehat{\mathbf{d}}$. The algorithm operates on the sequence of *additional degrees* $\mathbf{a} = \widehat{\mathbf{d}} - \mathbf{d}_G$ in a manner similar to the one the `ConstructGraph` algorithm operates on the degrees $\mathbf{d}$. However, since $\widehat{G}$ is drawn on top of the original graph G, we have the additional constraint that edges already in G cannot be drawn again.

Otherwise it proceeds iteratively and in each step it maintains the residual additional degrees $\mathbf{a}$ of the vertices. In each iteration it picks an arbitrary vertex v and adds edges from v to $\mathbf{a}(v)$ vertices of *highest* residual additional degree, ignoring nodes v' that are already connected to v in G. For every new edge (v, v'), $\mathbf{a}(v')$ is decreased by 1.

Keys of our algorithm

- Our implementation initially compute the degree anonymization algorithm.
- Create a new graph with mapping between name of label and number (logically for anonymization).
- Use the same structure of *ConstructGraph* with few difference:

  In particular, when all the values of vector 'a' are zero, we control if the number of edges added in a graph are equal to half of the sum of values of 'a'.

  If this condition is satisfied, it returns the new graph, otherwise it returns null.

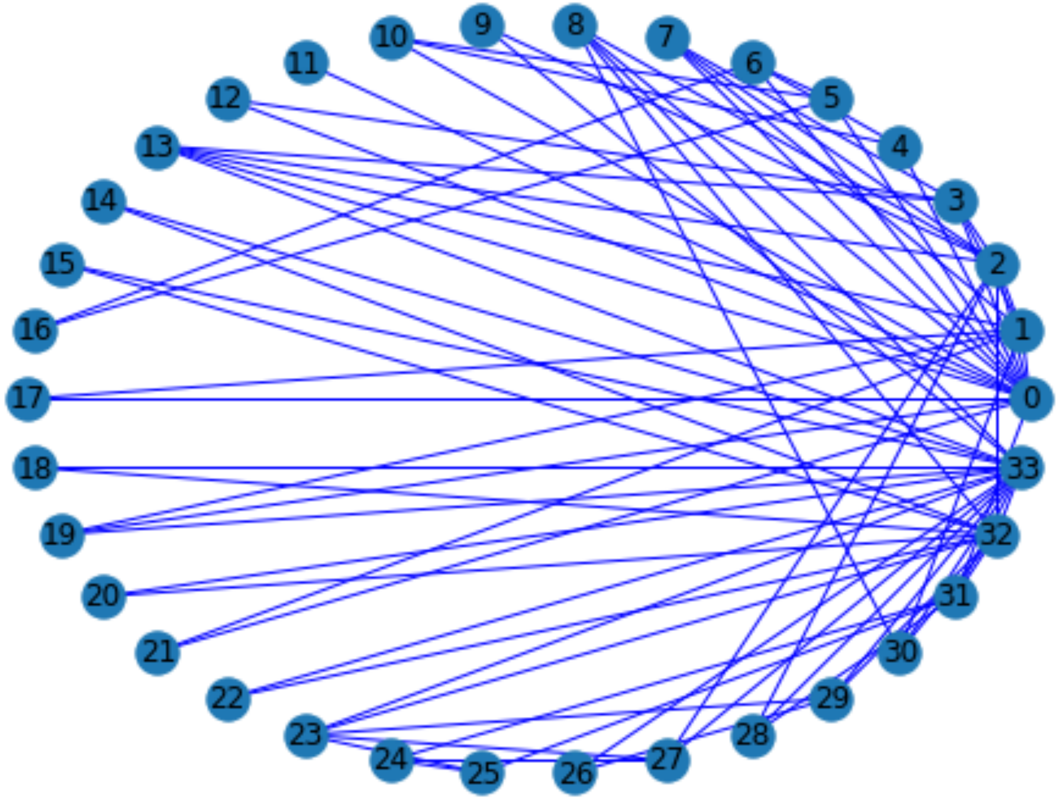
The real difference between ConstructGraph and Supergraph are:

- *ConstructGraph* is an oracle: if it returns null means that don't exist a graph for that degree anonymization;
- When the *Supergraph* returns null, it doesn't mean that do not exist a supergraph, but our implementation is not allow to find it.

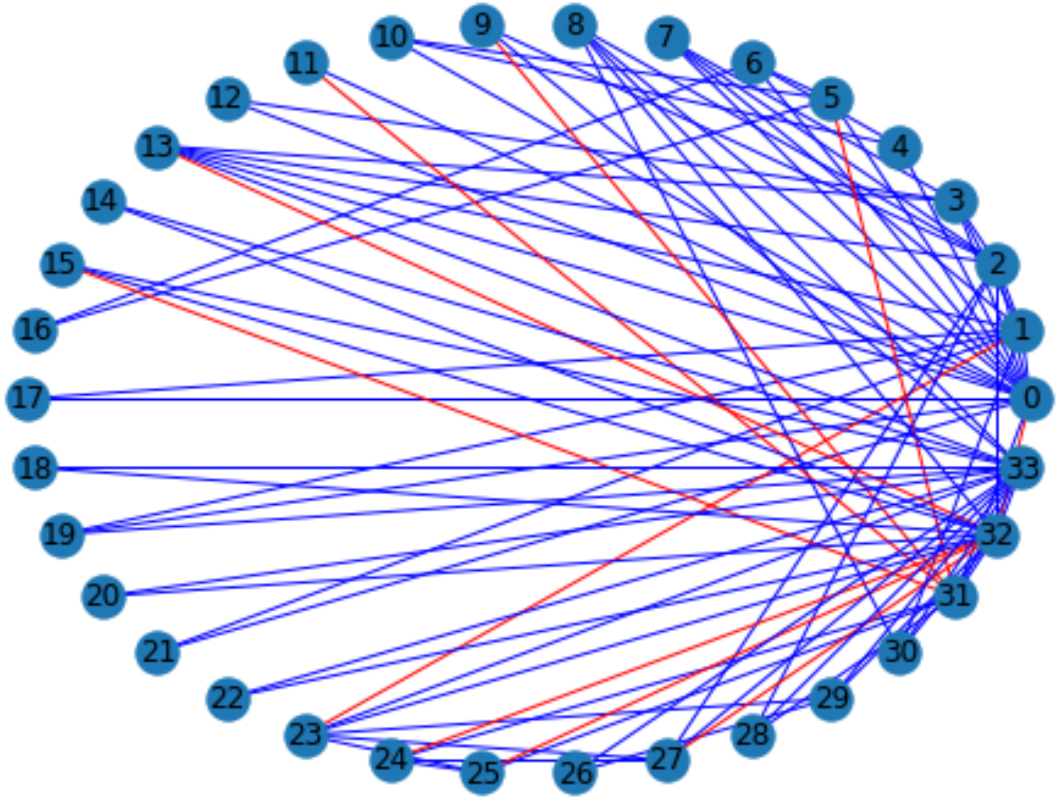
  With other approach (such as *probing* scheme) it's possible to force our supergraph algorithm with a little extra cost in order to find a supergraph.

Images of karate_club_graph

Starting graph



Supergraph

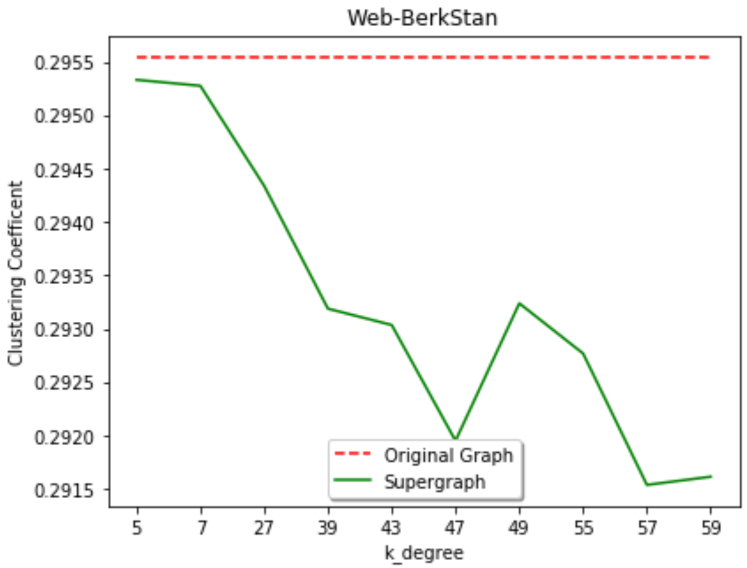


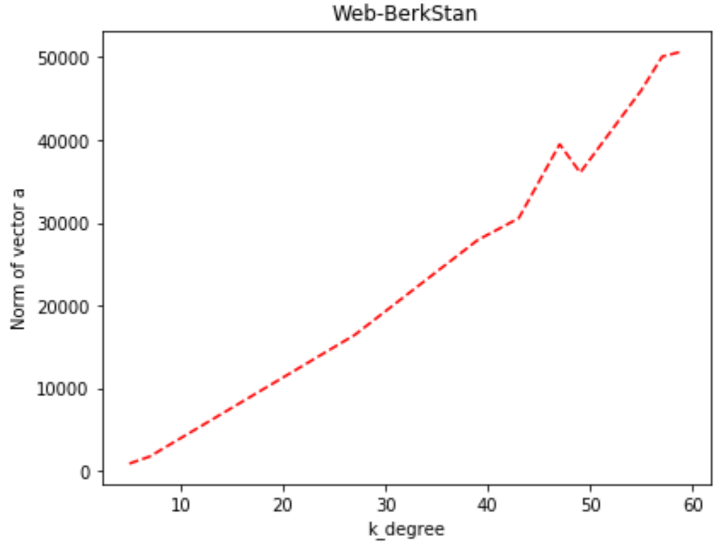
METRICS IN ORDER TO EVALUE THE SUPERGRAPH ALGORITHM

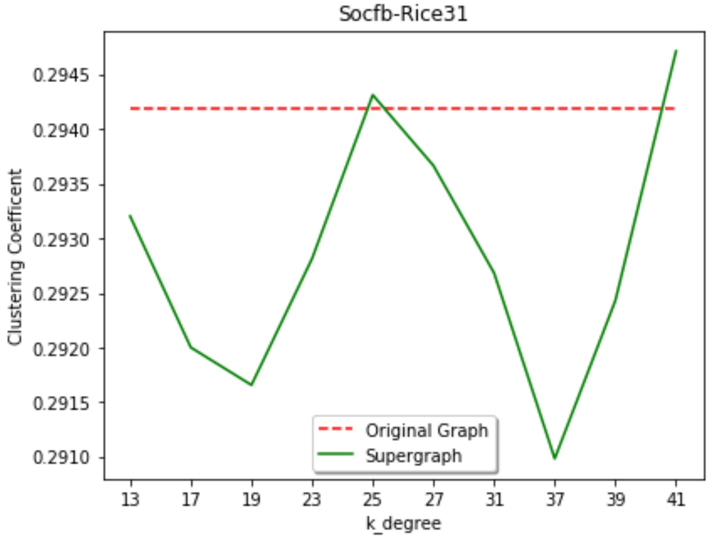
- Anonymization cost $L_1(d_A - d)$: This is the L_1 norm of the vector of differences between the k-anonymous degree sequence obtained using supergraph and the degree sequence of the original graph. The smaller the value of $L_1(d_A - d)$, the better is the qualitative performance of the algorithm.
- Clustering Coefficient (CC): We additionally compare the clustering coefficients of the anonymized graphs with the clustering coefficients of the original graphs. In graph theory, a clustering coefficient is a measure of the degree to which nodes in a graph tend to cluster together. Evidence suggests that in most real-world networks, and in particular social networks, nodes tend to create groups strongly united and characterized by a relatively high density of connections.
- In our testing we use an algorithm (inside Networkx library) in order to compute the average clustring coefficent;
- Formula is:

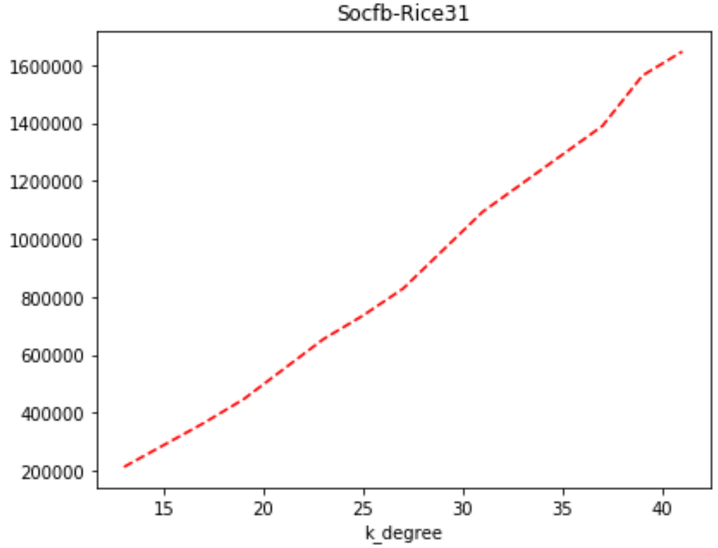
$$C = \frac{1}{n} \sum_{v \in G} c_v,$$

where c_v is clustering coefficent of each node and n is the numer of nodes.


Web-BerkStan
Clustering Coefficent
0.2955
0.2950
0.2945
0.2940
0.2935
0.2930
0.2925
0.2920
0.2915
Original Graph
Supergraph
5 7 27 39 43 47 49 55 57 59
k_degree


Web-BerkStan
Norm of vector a
50000
40000
30000
20000
10000
0
10 20 30 40 50 60
k_degree


Socfb-Rice31
Clustering Coefficent
0.2945
0.2940
0.2935
0.2930
0.2925
0.2920
0.2915
0.2910
Original Graph
Supergraph
13 17 19 23 25 27 31 37 39 41
k_degree


Socfb-Rice31
1600000
1400000
1200000
1000000
800000
600000
400000
200000
15 20 25 30 35 40
k_degree

- The vector norm grows linearly to the value of k-degree.
- In fact, the performances of algorithm decrease when k-degree increases.
- The graph of the average cluster coefficient does not define a precise analysis. However, we note that the original values (real datasets that include social networks) have almost always greater value than the anonymized ones (as said in the definition of clustering coefficient), because we treat them as graphs.

RESULTS

- We tested the algorithms mainly on two real datasets: web-BerkStan and socfb-Rice312. The *supergraph* is not always obtained (as is the construct graph). It could happen because there are nodes that can connect only with certain other nodes of the vector a (a very small range). If the latter are connected with nodes with a greater connection availability, the algorithm does not add the number of edges necessary because the former will have no nodes to connect to. A solution for always obtain a supergraph is the one adopted by the probing scheme which attempts by choosing, in a random way, some links until it finds the solution.
- In a README file we say how we have tested datasets

CODE

- The code is written in Python and published on GitHub (https://github.com/enz93/k-degree).
  All the code is written inside PyCharm. Read the file README inside the folder and you follow the instruction for execute the programm

REFERENCE

- Kun Liu and Evimaria Terzi. “Towards identity anonymization on graphs”. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. ACM. 2008, pp. 93–106.