Ben-Gurion University of the Negev
The Department of Computer Science

January 2019

Privacy Preserving Data Mining

An Information-Flow Control Model for Online Social Networks Based on User-Attribute Credibility and Connection-Strength Factors

Based on an article by Prof. Ehud Gudes and Mr. Nadav Voloch

Academic advisors
Prof. Ehud Gudes and Mr. Nadav Voloch

Michal Menachem
Eliran Kdoshim

Introduction

The algorithm we have implemented runs on an Online Social Network (OSN) represented by a graph, with a source node and a target node.
Let us say that Alice (our source node) has posted something on some OSN (e.g. Facebook), we would like to be able to assess Alice's willingness to let some other OSN user, say Eve (our target node), see that post.
The algorithm will determine whether Eve is an acquaintance or an adversary of Alice using both the OSN (graph) properties and social metrics based on previous research in that field.

The Algorithm: High Level View

Let N be a directed graph representing the information-flow in some OSN, where each node represents a user, and there exists a directed edge from one user to another, say from Alice to Bob, iff a post by Alice will show up on Bob's feed (e.g. on Facebook, when Alice and Bob are friends, as opposed to Instagram, where it is enough for Bob to follow Alice).

Next, let us define a score c (between 0 to 1) for each **node**, which will represent the node's (user's) credibility. The factors which are taken into account when computing the score are:

Variable	Attribute	Rationale
TF	Total Number of Friends	Fake profiles tend to have a small number of friends.
AUA	Age of User Account	Fake profiles and spammers' profiles are usually deleted due to OSN security policies.
FFR	Followers/Followees Ratio	Fake profiles usually follow a lot more users than they are being followed themselves.

Similarly, we shall define a score p (between 0 to 1) for each **edge**, which will represent the credibility of a connection between two users. The factors which are taken into account when computing the score are:

Variable	Attribute	Rationale
MF	Mutual Friends	Fake profiles, social-bots and adversaries don't usually have a big number of mutual friends with other users, if any.
FD	Friendship Duration	Similarly to the rationale of the AUA attribute, it is unlikely for a user to be friends with a fake user for a long period of time.
OIR	Outflow/Inflow Ratio	Fake profiles and spammers usually have a lot more inflow actions (e.g. advertisement) than outflow actions to them.
RA	Resemblance Attributes	A large number of common features between two users can strengthen the credibility of their friendship.

All of the attributes above (both the nodes' and edges') are derived from previous research.

Now, using these scores, we shall define a score for a **path** from the source node to the target node in N.
For $1 \leq i \leq n$, whereas n is the number of paths from source to target, $PATH_i^{src \to trgt}$ is defined as the set of all the nodes and edges in that path (including source and target).
The score of a path will define the Total Sharing Probability (TSP) value, which will indicate the willingness of the source node to share information with the target node. Let k be the number of edges in

$PATH_i^{src \to trgt} = v_0 e_1 v_1 \dots e_k v_k$, where e_i is the edge from v_{i-1} to v_i so the TSP of that path is defined as:

$$TSP\left(PATH_i^{src \to trgt}\right) = \prod_{i=1}^{k} c_{v_i} p_{e_i}$$

We shall note that $c_{v_{src}}$ is omitted from the calculation, since the source obviously does not need to be checked for its credibility.

Now, we must set some threshold numeric value of Minimum Sharing Probability (MSP) for the TSP of a path for it to be considered as safe. The article we have based on had chosen the value 0.5, meaning having $TSP \geq 0.5$ marks the path as safe, and the target node as an acquaintance. This value is of course debatable and flexible, and demands further research.

A pseudo-code of the algorithm:

isAnAcquaintanceNotAnAdversary (Graph G, Node v_{src}, Node v_{trgt})

1. $MSP \leftarrow 0.5$
2. $\{AllPaths^{src \to trgt}\} \leftarrow AllPathsSrcToTarget\left(G, v_{src}, v_{trgt}\right)$
3. for $1 \leq i \leq n$:
    3.1. if $TSP\left(PATH_i^{src \to trgt}\right) \geq MSP$
        3.1.1. return $true$
4. return $false$

Our Algorithm: Detailed Description

We have described the variables used in this model, now, we shall describe exactly how we use the attributes to define a node's or an edge's score.
We have said that c is the credibility value of a user (node) and p is the credibility value of a connection (edge). In order to define their values, we shall define the credibility values of each of the nodes' and the edges' attributes, which will be marked as $c_{attribute}$ or $p_{attribute}$ respectively (e.g. c_{TF} is the credibility value for the Total Friends user attribute and p_{MF} is the credibility value for the Mutual Friends connection attribute). It is worth noting that the following assignments are partly debatable. Some are based on previous researches and some on given estimations for the purpose of this model, making them flexible.

The values are:

$$c_{TF} = \begin{cases} \dfrac{TF}{AvgTF} & (TF < AvgTF) \\ 1 & (TF \geq AvgTF) \end{cases}$$

Where $AvgTF$ is the average amount of total friends among all OSN users. A profile with more than the average amount of friends is with a high probability of being genuine.

$$c_{AUA} = \begin{cases} \dfrac{AUA}{365} & (AUA < 365) \\ 1 & (AUA \geq 365) \end{cases}$$

The c_{AUA} is calculated in days, and relies on the fact that spammers' profiles are usually taken down due to OSN security policies.
That makes a user who has been active for over a year genuine with a high probability.

$$c_{FFR} = \begin{cases} FFR & (FFR < 1) \\ 1 & (FFR \geq 1) \end{cases}$$

As stated in the rationale of the FFR attribute, fake profiles usually follow a lot more users than are being followed themselves. Therefore, a followers/followees ratio ≥ 1 will make the user profile genuine with a high probability.

$$p_{MF} = \begin{cases} \dfrac{MF}{\max\{u_{TF}, v_{TF}\} \cdot 0.1} & (MF < \max\{u_{TF}, v_{TF}\} \cdot 0.1) \\ 1 & (MF \geq \max\{u_{TF}, v_{TF}\} \cdot 0.1) \end{cases}$$

Where (u, v) is the edge. The p_{MF} value is based on the rationale of the MF connection attribute, that fake profiles tend to have a small number of mutual friends with other users, if any.
u_{TF} and v_{TF} are the TF user attribute values of the nodes u and v respectively. We rely on the estimation that if the number of mutual friends of these two nodes (users) is at least a tenth of the maximum number of total friends between that of the two, the connection between them is considered genuine.

$$p_{FD} = \begin{cases} \frac{FD}{365} & (FD < 365) \\ 1 & (FD \geq 365) \end{cases}$$

The p_{FD} value is calculated in days, and is based on the rationale of the FD connection attribute, that it is unlikely for a user to be friends with a fake user or a spammer for a long period of time, based on the rationale of the AUA user attribute.

$$p_{OIR} = \begin{cases} OIR & (OIR < 1) \\ 1 & (OIR \geq 1) \end{cases}$$

The p_{OIR} value relies on the rationale behind the OIR connection attribute, that fake profiles and spammers usually have a lot more inflow actions than outflow actions to them. Therefore, a connection where the OIR is at least 1 is with a high probability genuine, since both users interact with each other and therefore are likely to be actual friends.

Now, in order to talk about the $\boldsymbol{p_{RA}}$ value, we must first discuss the resemblance attributes which are taken into account in the calculation. These resemblance attributes were chosen based on previous works, and are: hometown, current country, current city, home country, gender, language and religion. Each of these resemblance attributes can get either a non-null or a null value.
Let (u, v) be the edge we wish to set its p_{RA} value. Let us denote the following factors:
- We define $\boldsymbol{TA_u}$ to be the total number of non-null resemblance attributes (from the list of 7 attributes above) of u.
- We define $\boldsymbol{TRA_{u,v}}$ to be the total number of non-null resemblance attributes of u and v that are the equal (e.g. they both currently live in Israel).

Now, we are ready to define p_{RA}:

$$p_{RA} = \frac{TRA_{u,v}}{TA_u}$$

Firstly, it is important to state that this value is between 0 to 1:
- At least 0 since both TA_u and $TRA_{u,v}$ are defined to be positive numbers between 0 to 7.
- At most 1 since the maximal number of common non-null resemblance

attributes between u and v could be the total number of non-null resemblance attributes of u at most.

Secondly, this assignment to the p_{RA} value portrays the rationale of the RA attribute very well, since the more similar two users are (the more resemblance attributes they have in common), the more likely they are to be actual friends.

Now, we can define the c and p values for each node and edge respectively, by averaging the different factors above:

$$c = \frac{c_{TF} + c_{AUA} + c_{FFR}}{3}$$

$$p = \frac{p_{MF} + p_{FD} + p_{OIR} + p_{RA}}{4}$$

It is worth noting that in our model we assumed that the significance of each of the factors was the same, but if that is not the case, each factor can be multiplied by its weight (importance) in the calculation.

Additionally, we would like to present an **optimization** for the algorithm suggested on page 4.

A pseudo-code of the optimized algorithm:

isAnAcquaintanceNotAnAdversaryOptimized (Graph $G = (V, E)$, Node v_{src}, Node v_{trgt})

1. $MSP \leftarrow 0.5$
2. for $(u, v) \in E$:
    2.1. $w((u, v)) = \ln(c_v * p_{u,v})$
3. $heaviestPath \leftarrow BellmanFord(G, v_{src}, v_{trgt}, w)$
4. if $TSP(heaviestPath) \geq MSP$
    4.1. return $true$
5. Return $false$

We shall now explain the correctness of this optimized algorithm:

- Let us note that **isAnAcquaintanceNotAnAdversary** (G, v_{src}, v_{trgt}) returns $true \Leftrightarrow$ there exists a path from v_{src} to v_{trgt} whose

TSP is at least MSP ⇔ the maximal TSP of a path from v_{src} to v_{trgt} is at least MSP.

- Claim: let there be $heaviestPath$ as described in the pseudo-code above. $TSP(heaviestPath) \geq TSP(path)$ for every path from v_{src} to v_{trgt}.
  Proof: Let $p = v_0 e_1 v_1 \dots e_m v_m$ be some path from v_{src} to v_{trgt}.
  From the way $heaviestPath = v'_0 e'_1 v'_1 \dots e'_k v'_k$ is defined, the following holds:

$$\sum_{i=1}^{k} w(e'_i) \geq \sum_{i=1}^{m} w(e_i) \quad \Leftrightarrow$$

$$\Leftrightarrow \quad \sum_{i=1}^{k} \ln\left(p_{e'_i} \cdot c_{v'_i}\right) \geq \sum_{i=1}^{m} \ln\left(p_{e_i} \cdot c_{v_i}\right) \quad \Leftrightarrow$$

$$\Leftrightarrow \quad \ln\left(\prod_{i=1}^{k} p_{e'_i} \cdot c_{v'_i}\right) \geq \ln\left(\prod_{i=1}^{m} p_{e_i} \cdot c_{v_i}\right) \quad \Leftrightarrow$$

$$\Leftrightarrow \prod_{i=1}^{k} p_{e'_i} \cdot c_{v'_i} \geq \prod_{i=1}^{m} p_{e_i} \cdot c_{v_i} \quad \Leftrightarrow$$

$$\Leftrightarrow \quad TSP(heaviestPath) \geq TSP(p)$$

From combining the two points above we can conclude that **isAnAcquaintanceNotAnAdversaryOptimized** returns $true$ iff **isAnAcquaintanceNotAnAdversary** returns $true$.

Original algorithm runtime:

Reminder: n is the number of paths from the source node to the target node in N.
Worst case runtime: when $n = O(V!)$ – we get a runtime of $\boldsymbol{O(E \cdot V!)}$ since calculating the TSP of a given path takes $O(E)$.

Optimized algorithm runtime:

The for-loop in step 2 - $O(E)$.
Running Bellman-Ford - $O(V \cdot E)$.
Calculating the TSP of the heaviest path - $O(E)$.
A total runtime of $\boldsymbol{O(V \cdot E)}$.

Software Description

Our software consists of one file. Throughout the code we have used a Python library called *NetworkX* for performing graph operations.
The most important methods are:

- main()
  - build_undirected_graph()
    - set_scores()
  - build_directed_graph()
  - is_acquaintance() \ is_acquaintance_optimized()
    - compute_tsp(path)

It should be noted that the graph g that represents the OSN itself is a global variable that is accessed and modified by all the methods above.

Let us elaborate on each one:

build_undirected_graph() – This method generates a random undirected graph g that represents an OSN – the nodes represent the users and the edges between two nodes (users) signifies a friendship between them in the OSN. The number of nodes and edges (i.e. number of users and friendships) is received as input from the user. We then call the method *set_scores()* which assigns a score c and a score p for every node and edge respectively.

set_scores() – This method assigns scores to the nodes and the edges in g. These scores are partly derived from the structure of g (the OSN) and partly randomly generated. We will now elaborate on the way these scores were derived\generated:

First, for each node v:

- $v[TF] = \deg(v)\ in\ g$
- $v[AUA] = random\ int\ in\ [0, 365 \cdot 12]$ (since Facebook has been around for about 12 years).
- $v[FFR]$ – For each user, we randomly chose the users he follows. The FFR value is derived directly from that.
- Additionally, for the sake of computing the RA attribute of each edge, we randomly generated the resemblance attributes.
  Firstly, every resemblance attribute is either null or non-null uniformly. Therefore, we randomly picked a number in [0, 1] – if it was smaller than 0.5 we let the resemblance attribute be null,

otherwise we performed the following:
- $v[hometown] = random\ int\ in\ [1, 200]$.
- $v[current\_country] = random\ int\ in\ [1, 15]$.
- $v[current\_city] = random\ int\ in\ [1, 200]$.
- $v[home\_country] = random\ int\ in\ [1, 15]$.
- $v[gender] = random\ int\ in\ [1, 2]$.
- $v[language] = random\ int\ in\ [1, 50]$.
- $v[religion] = random\ int\ in\ [1, 7]$.

- $v[c] = \frac{v[FD]+v[AUA]+v[FFR]}{3}$

Secondly, for each edge (u, v):

- $(u, v)[MF] = |neighbors(u) \cap neighbors(v)|$
- $(u, v)[FD] = random\ int\ in\ [0, \min(u[AUA], v[AUA])]$
- $(u, v)[OIR] = random\ float\ in\ [0, 3]$
- $(u, v)[RA] = \frac{TRA((u,v))}{TA(u)}$ as TRA and TA defined above, with resemblance attributes generated in the previous for-loop.
- $(u, v)[p] = \frac{(u,v)[MF]+(u,v)[FD]+(u,v)[OIR]+(u,v)[RA]}{4}$

build_directed_graph() – In this method, we morph our randomly generated undirected graph into a directed acyclic graph which represents the information flow from the source node received as user input. We turn the graph to an acyclic one since paths from source to target that contain loops have a TSP that is not larger than the TSP of that same path without the loops, so we can ignore those loops, since we want the TSP to be large. In order to turn the graph into a DAG, for every undirected edge we will keep only the edges going outwards from the source node and will set the direction of the edge accordingly (in the opposite direction of the source). We shall note that the DAG only contains the connectivity component of the source node, since any nodes out of it are not relevant.

is_acquaintance() – In this method we implement the algorithm in its original form (as it was described in the article).
Firstly, we check whether the target node is in the source node's connectivity component, and if it is not, we can determine that there is no

path from source to target, and return $false$. Otherwise, we run an algorithm from the *NetworkX* library called *all_simple_paths* which receives as input a graph, a source node and a target node and returns all simple paths from source to target. We then iterate over all those paths and compute their TSP using *compute_tsp(path)* until we reach a path with TSP $\geq MSP$, and then return $true$. Otherwise, return $false$.

is_acquaintance_optimized() - In this method we implement the algorithm in its optimized form. Firstly, we check whether the target node is in the source node's connectivity component, and if it is not, we can determine that there is no path from source to target, and return $false$. Otherwise, for every edge (u, v) in the DAG we define an attribute *weight* to be so: $\ln\left(\frac{1}{(u,v)[p]*v[c]}\right)$. Using this *weight* attribute we then call *Bellman-Ford* to find the lightest path from source to target. Let us note that the lightest path from source to target using the above *weight* is the heaviest path had we used the *weight*: $\ln((u, v)[p] * v[c])$, which we had proved in the previous section to have the largest TSP of a path from source to target. It is worth noting that we have used *Bellman-Ford* to find the **lightest** path from source to target as opposed to the **heaviest** since *NetworkX* has only implemented it so. We then check whether the path's TSP is $\geq MSP$ – if so, we return $true$. Otherwise, $false$.

compute_tsp(path) – In this method, we compute the TSP of the path received as input. *NetworkX* returns paths in the form of a list of nodes. Since we want to iterate through the edges, we created a list of the path's edges using the Python Standard Library method *zip(path, path[1:])*, which receives two lists and returns a list of pairs, pairing elements of the same index in the lists, resulting in a list of all the edges.
We then iterate over the edges and compute the TSP.

User Interaction Description

Our user interaction includes:

(1) When first running the program, the user is requested to insert the following parameters as input:
- Amount of users in the OSN.
- Requested total amount of friendships between users in the OSN.
- The requested MSP.

(2) After the user inserts the above input, an undirected graph which represents the OSN appears on the user's screen. Each of the nodes has an identifying number on it.

(3) After the user closes the graph's window, he is requested to insert the following parameters as input:
- The requested source node (its number).
- The requested target node (its number).

(4) Later, a DAG which contains the connectivity component of the user node appears on the screen, representing the information flow from the source user. If the target node was found to be an acquaintance of the source node, the nodes of the path from the source to the target which the algorithm found to be of TSP $\geq MSP$ (we shall call that path *ansPath* in (5)) are colored in blue (when the rest of the nodes are red).
Also, the scores of the nodes and the edges appear on them respectively.

(5) In addition, the following information is printed to the user's terminal, both for the original and the optimized algorithms:
- The TSP of *ansPath*.
- Whether the target node is an acquaintance or an adversary.
- The scores of the nodes and edges of the DAG.

Printouts of Test Results

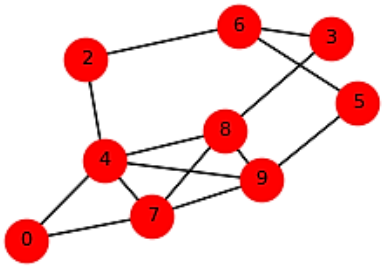
We shall present printouts of 4 different test runs.

Test 1:

(1)

```
Insert the requested amount of users: 10
Insert the requested amount of friendships: 14
Insert the requested MSP: 0.5
```

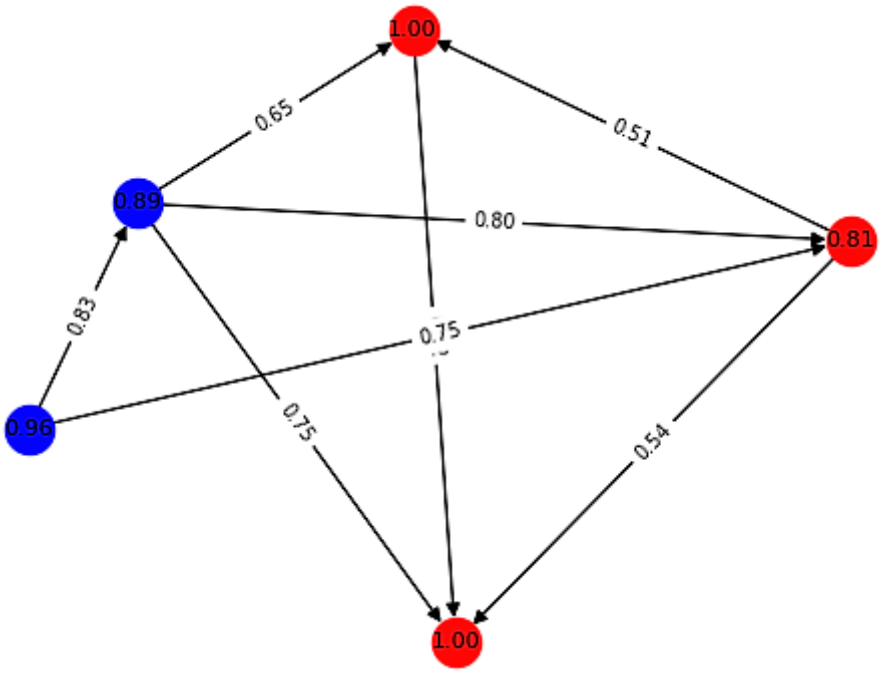
(2)



(3) + (5)

```
Insert the requested source node: 0
Insert the requested target node: 4
TSP received is (regular algo): 0.7407407407407407
Is acquaintance (regular algo): True
TSP received is (optimized algo): 0.7407407407407407
Is acquaintance (optimized algo): True
Node number: 0  c: 0.9583333333333334
Node number: 4  c: 0.8888888888888888
Node number: 7  c: 0.8095238095238094
Node number: 8  c: 1.0
Node number: 9  c: 1.0
Edge between 0 and 4    p: 0.8333333333333334
Edge between 0 and 7    p: 0.75
Edge between 4 and 8    p: 0.6453789159300103
Edge between 4 and 9    p: 0.75
Edge between 4 and 7    p: 0.8
Edge between 7 and 9    p: 0.5395551155858551
Edge between 7 and 8    p: 0.510703402720946
Edge between 8 and 9    p: 0.618871976998055
```

(4)

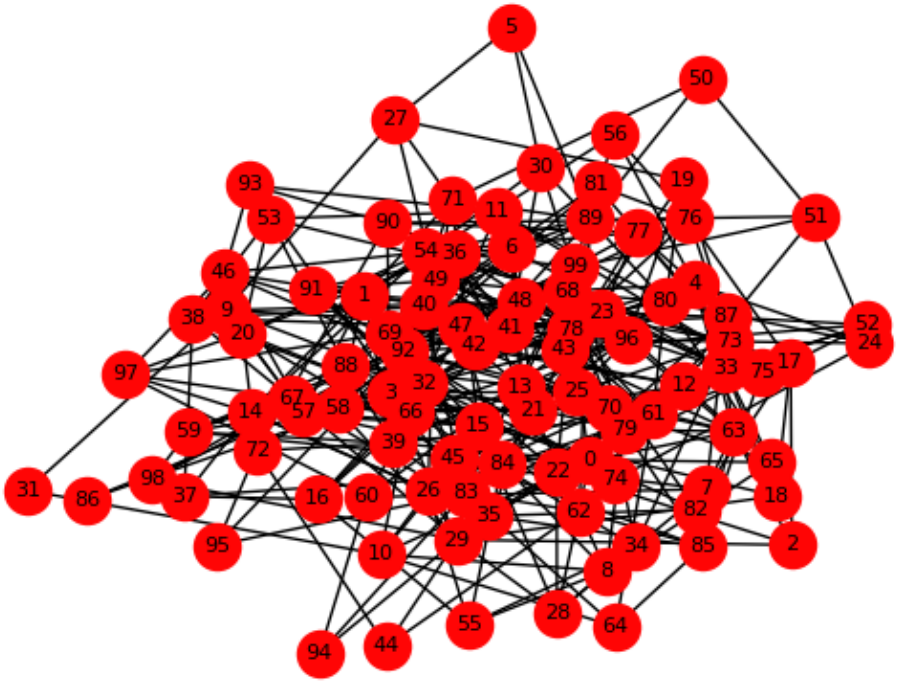


Test 2:

(1)

```
Insert the requested amount of users: 100
Insert the requested amount of friendships: 360
Insert the requested MSP: 0.3
```

(2)

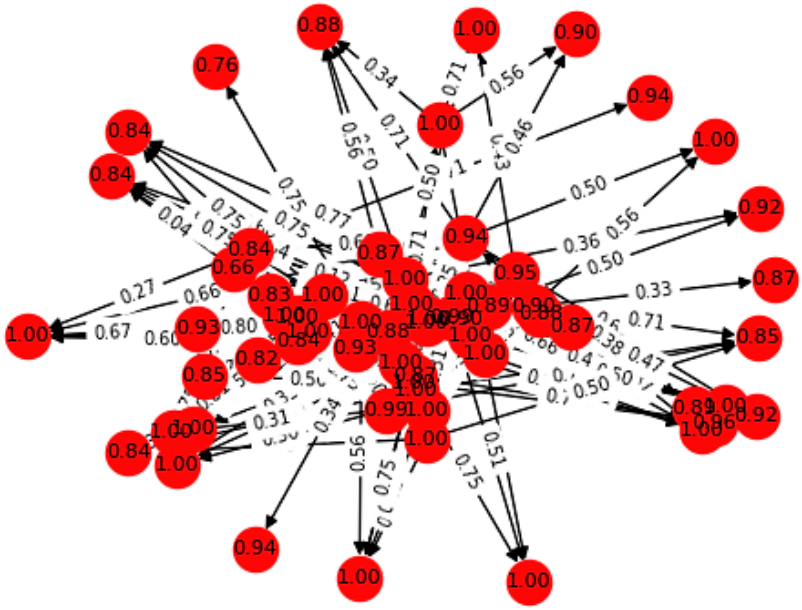


(3) + (5) Only partial printouts of the final output, since the graph is large and therefore so is the output regarding the nodes' and edges' scores.

```
Insert the requested source node: 12
Insert the requested target node: 80
TSP received is (regular algo): 0
Is acquaintance (regular algo): False
TSP received is (optimized algo): 0.17813979473476882
Is acquaintance (optimized algo): False
Node number: 0  c: 1.0
Node number: 1  c: 1.0
Node number: 2  c: 0.8675213675213675
Node number: 3  c: 0.8401360544217686
Node number: 4  c: 0.9154929577464789
Node number: 5  c: 0.9383838383838383
Node number: 6  c: 1.0
Node number: 7  c: 0.8241758241758242
Node number: 8  c: 1.0
Node number: 9  c: 0.9047619047619048
Node number: 10 c: 0.8409090909090908
Node number: 11 c: 1.0
Node number: 12 c: 0.9322033898305085
Node number: 23 c: 1.0
Node number: 25 c: 1.0
Node number: 33 c: 0.839662447257384
Node number: 36 c: 0.8659420289855072
Node number: 41 c: 1.0
Node number: 42 c: 1.0
Node number: 45 c: 1.0
Node number: 46 c: 0.9500000000000001
Node number: 48 c: 1.0
Node number: 49 c: 1.0
Node number: 52 c: 0.8328174108996027
Node number: 54 c: 0.8721461187214613
Node number: 57 c: 1.0
```

```
Edge between 12 and 23  p: 0.4261637231114473
Edge between 12 and 42  p: 0.5
Edge between 12 and 45  p: 0.75
Edge between 12 and 79  p: 0.5
Edge between 23 and 1   p: 0.35890410958904106
Edge between 23 and 3   p: 0.75
Edge between 23 and 84  p: 0.5
Edge between 23 and 33  p: 0.5625
Edge between 23 and 25  p: 0.6127440319167455
Edge between 23 and 77  p: 0.2528231081607571
Edge between 23 and 36  p: 0.5
Edge between 25 and 96  p: 0.75
Edge between 25 and 83  p: 0.75
Edge between 25 and 3   p: 0.75
Edge between 25 and 88  p: 0.5
Edge between 25 and 73  p: 0.5
Edge between 25 and 92  p: 0.65625
Edge between 25 and 63  p: 0.5535388127853881
Edge between 33 and 0   p: 0.8
Edge between 33 and 65  p: 0.5735640845298584
Edge between 33 and 68  p: 0.8125
Edge between 33 and 6   p: 0.6006849315068493
Edge between 33 and 48  p: 0.5664383561643835
Edge between 33 and 79  p: 0.75
Edge between 33 and 63  p: 0.7583193813346696
Edge between 36 and 99  p: 0.5552350317104103
Edge between 36 and 6   p: 0.663013698630137
Edge between 36 and 41  p: 0.75
Edge between 36 and 90  p: 0.6614861926160648
Edge between 36 and 92  p: 0.5363869863013698
Edge between 36 and 78  p: 0.7083333333333333
Edge between 41 and 49  p: 0.5431115710945741
Edge between 41 and 90  p: 0.6209340036083403
Edge between 41 and 61  p: 0.5
Edge between 41 and 63  p: 0.55
Edge between 42 and 4   p: 0.4558117091116288
Edge between 42 and 70  p: 0.2904109589041096
Edge between 42 and 89  p: 0.11635856273180997
Edge between 42 and 46  p: 0.5
Edge between 45 and 65  p: 0.8159994919562239
Edge between 45 and 85  p: 0.5
Edge between 45 and 1   p: 0.26165190171110314
Edge between 45 and 10  p: 0.5
Edge between 45 and 59  p: 0.5
```

(4)

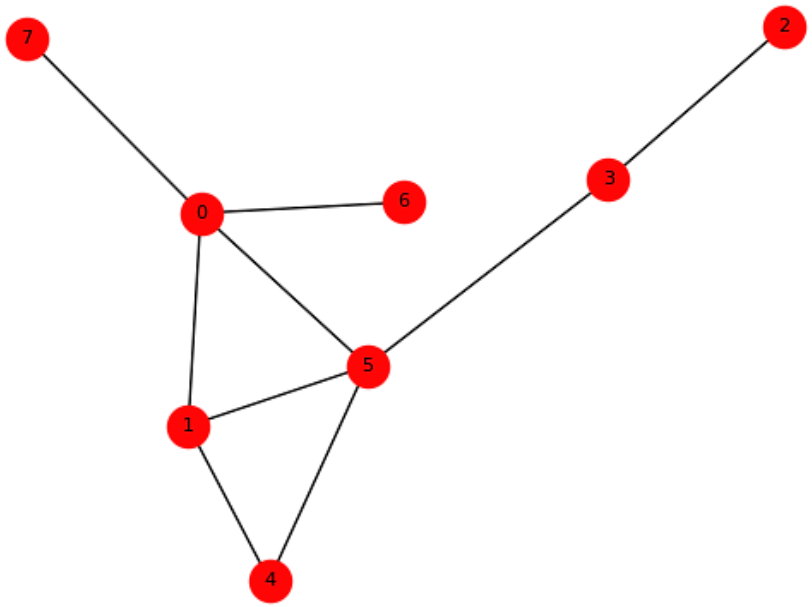


Test 3:

(1)

```
Insert the requested amount of users: 8
Insert the requested amount of friendships: 9
Insert the requested MSP: 0.5
```

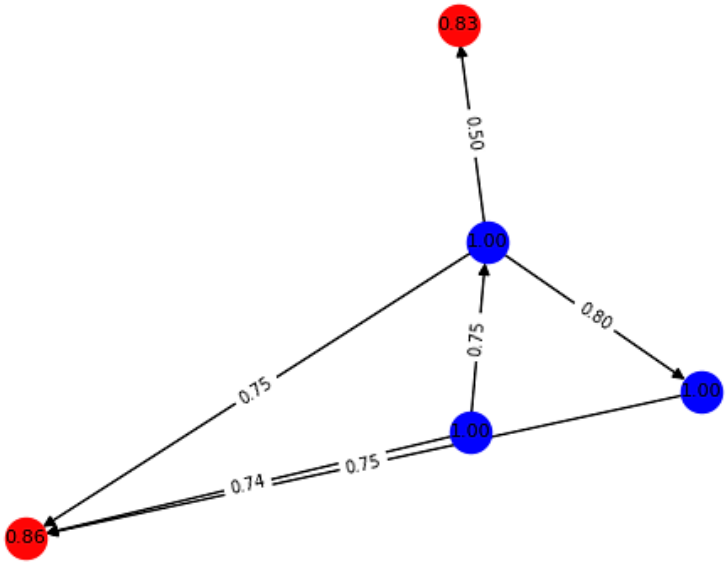
(2)



(3) + (5)

```
Insert the requested source node: 4
Insert the requested target node: 0
TSP received is (regular algo): 0.6000000000000001
Is acquaintance (regular algo): True
TSP received is (optimized algo): 0.6000000000000001
Is acquaintance (optimized algo): True
Node number: 0  c: 1.0
Node number: 1  c: 0.8571428571428571
Node number: 3  c: 0.8333333333333334
Node number: 4  c: 1.0
Node number: 5  c: 1.0
Edge between 0 and 1    p: 0.75
Edge between 4 and 1    p: 0.7356164383561644
Edge between 4 and 5    p: 0.75
Edge between 5 and 0    p: 0.8
Edge between 5 and 1    p: 0.75
Edge between 5 and 3    p: 0.5
```

(4)

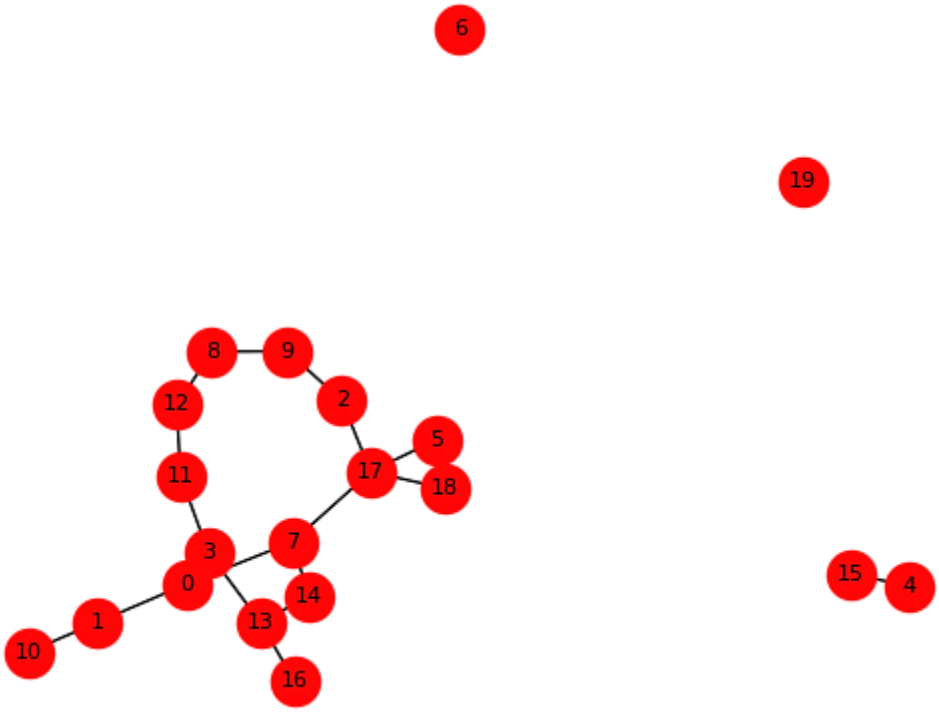


Test 4:

(1)

```
Insert the requested amount of users: 20
Insert the requested amount of friendships: 18
Insert the requested MSP: 0.45
```

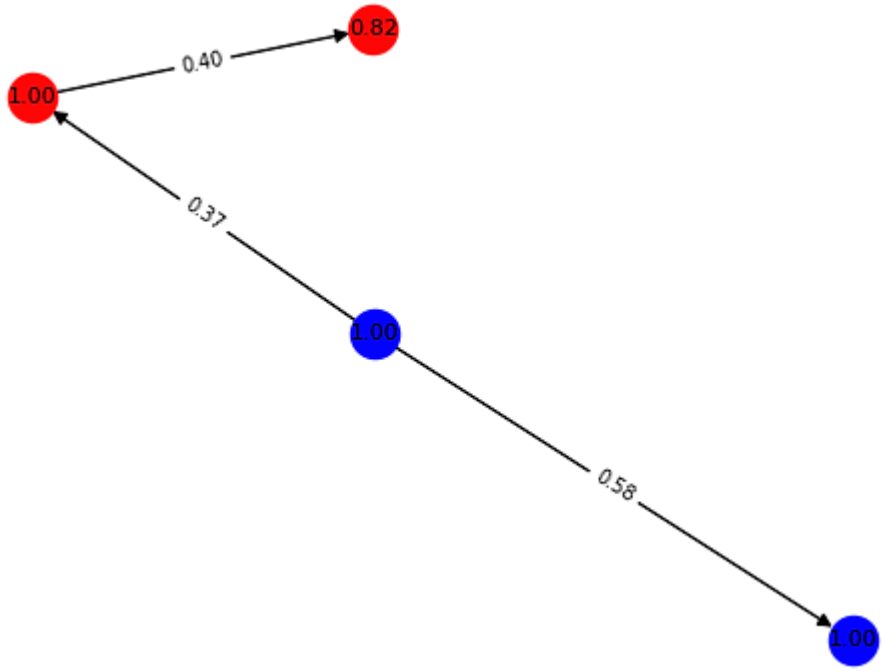
(2)



(3) + (5)

```
Insert the requested source node: 8
Insert the requested target node: 12
TSP received is (regular algo): 0.5833333333333334
Is acquaintance (regular algo): True
TSP received is (optimized algo): 0.5833333333333334
Is acquaintance (optimized algo): True
Node number: 8  c: 1.0
Node number: 9  c: 1.0
Node number: 2  c: 0.8222222222222223
Node number: 12 c: 1.0
Edge between 8 and 9    p: 0.3698630136986301
Edge between 8 and 12   p: 0.5833333333333334
Edge between 9 and 2    p: 0.4013698630136986
```

(4)



Conclusion

In this project we have implemented a Flow-Control model for an OSN, which takes into account user-credibility and connection-credibility attributes in order to assess a user's willingness to share information with some other user. In addition to the implementation presented in the article we have based on, we have presented an alternative implementation that uses *Bellman-Ford*'s algorithm with a better runtime.
Further work can be done regarding this model, perhaps one which supplies a solution for a similar, yet quite different problem: how can we assess a user's willingness to share information with another user which is not necessarily a spammer or a fake profile, but just a genuine user with whom the user might not want to share information with.

Bibliography

Gudes Ehud and Nadav Voloch - "An Information-Flow Control Model for Online Social Networks Based on User-Attribute Credibility and ConnectionStrength Factors". International Symposium on Cyber Security Cryptography and Machine Learning. Springer, Cham, 2018.