Improve Fuzzy Clustering and Force Directed Algorithms for Visualization of Personal Social Networks

Yao Yujian

Abstract

To properly visualize one's social network which typically contains outliers and sometimes parse communities, we improve a community detection algorithm called fuzzyclust and implement a better force-directed-based visualization algorithm that takes into consideration the community structure. We build an online tool based on these algorithms and present the algorithms in this article.

1 Introduction

The advent of social networking sites such as Facebook has allowed people to digitalize their real-life social network, making the process of visualizing one's social network much simpler as the data gathering process is straight- forward and efficient. This makes it possible to easily analyze one's social network structure to find out gather interesting insights such as detecting communities and outliers. To capitalize this, we build an online social network analysis tool that shows users overlapping communities and outliers in their personal social network.

2 Problem Statement

This project aims to build a tool to visualized one's personal social network by showing communities and outliers. A personal social network is a social network formed by only one's friends. Two friends are considered connected if they are also friends of each other.

Communities are a group of people who have denser connections among each other. Outliers are people in the social network that does not belong to any communities.

We call the network *personal* social network because it's a social network with imperfect information due to one's limitation in interacting with others. For example, a few of a person's friends could have belonged to some obvious community, but that person does not know many from that community, so such a community cannot be trivially discovered given only the person's social network data.

3 Algorithms

The visualization consists of two steps: community detection and positioning of vertexes based on the graph structure for better representation.

3.1 Community Detection

We utilized the fuzzyclust (Nepusz et al. (2008)) algorithm, with some improvements specific to personal social networks.

3.1.1 Fuzzyclust Algorithm

The fuzzyclust algorithm works by optimizing a function of posterior similarities of vertex, defined by the similarity of the communities they belong to, based on the prior assumption of their similarity.

Formally, let c be the number of communities and N be the number of vertexes, define a *partition matrix* U such that for each entry u_{ij}:

$$u_{ij} \in [0,1] \text{ for all } 1 \leq i \leq c, 1 \leq k \leq N$$

$$\sum_{i=1}^{c} u_{ik} = 1 \text{ for all } 1 \leq k \leq N$$

$$0 < \sum_{k=1}^{N} u_{ik} < N \text{ for all } 1 \leq i \leq c$$

Intuitively, u_{ij} refers to the *degree* that Vertex j belongs to community i, such that $u_{ij} = 0$ means that j does not belong to i at all and $u_{ij} = 1$ means that j belongs to and only belongs to i.

With the *partition matrix*, define the similarity of two vertex i and j as:

$$s_{ij} = \sum_{k=1}^{c} u_{ki} u_{kj}$$

Also define a prior similarity assumption:

$$\tilde{s}_{ij} = \begin{cases} 1, & \text{if i and j are adjacent} \\ 0, & \text{otherwise} \end{cases}$$

Then we have a goal function:

$$D_G(U) = \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} (\tilde{s}_{ij} - s_{ij})^2$$

where w_{ij} is the weight between two vertexes that can be varied to tweaked the algorithm. The simplest case is where $w_{ij} = 1$ for all i, j.

By minimizing $D_G(U)$ with regard to U, then, we can find the optimal partition matrix that is indicative of the communities each vertex belongs to.

We can then utilize a iterative gradient-based method to optimize this function and obtain a good partition matrix.

Firstly, it's shown that the gradient of the goal function is:

$$\frac{\partial \tilde{D_G}}{\partial u_{kl}} = 2 \sum_{i=1}^{N} (e_{il} + e_{li}) \left(\frac{1}{c} - u_{ki} \right)$$

where $e_{ij} = w_{ij}(\tilde{s}_{ij} - s_{ij})$.

So the following iterative algorithm can be employed:

1. Initialize a randomized partition $U^{(0)}$ and let $t = 0$
2. Compute $\dfrac{\partial \tilde{D_G}}{\partial u_{kl}}$ for all k, l

3. If $\max_{k,l} \frac{\partial \tilde{D_G}}{\partial u_{kl}} < \varepsilon$, stop the iteration and return $U^{(t)}$ as the solution
4. Otherwise, calculate the next partition in the iteration with the following equation:

$$u_{ij}^{(t+1)} = u_{ij}^{(t)} + \alpha^{(t)} \frac{\partial \tilde{D_G}}{\partial u_{kl}}$$

   where $\alpha^{(t)}$ is the step size constant chosen appropriately.
5. Increase t by 1 and continue from step 2

The above algorithm works if the number of community c is known. In the case that c is not known, we can increment c from 2 and compute the optimal U, then the modularity given U, until the modularity does not increase.

3.1.2 Improvement

One problem with fuzzyclust is that pairs of non-adjacent vertexes are 'treated equally' because their prior similarities are all 0. Although this can be mitigated with the graph structure - since the vertexes may be connected to some other vertexes which will dictate a prior similarity of 1 with them - it frequently lead to situations where disjoint, sparse vertexes are put into the same community. This can occur quite frequently in personal social networks, where there can exist some sparse community due to lack of iteration with that particular group of people.

To prevent this, we make use of the weight parameter w_{ij} to penalize heavily when disconnected vertexes are joined into the same community. Moreover, to avoid over-penalizing the goal function due to outlier vertexes, we create one *residual* community to host outliers.

We also normalize the graph before running fuzzyclust by decomposing the graph into connected components, such that if the size of connected component is less than 10, we assume the whole component to be one community, otherwise run fuzzyclust on the component to obtain communities within the component.

Formally, let the distance between two vertexes be d_{ij}, and let the maximum distance in the graph be d_{max}, we define w_{ij} as:

$$w_{ij} = \begin{cases} f(d_{max}+1)/(degree_i + degree_j - 2), & \text{if } d_{ij} = 1, \\ f(d_{ij}), & \text{if } d_{ij} > 1 \end{cases}$$

where $f(x)$ is an increasing function. In this project, we pick $f(x) = 2^x$.

We also modify the similarity to:

$$s_{ij} = \sum_{k=1}^{c} m_k^2 u_{ki} u_{kj}$$

where m_k is a depress factor such that:

$$m_k = \begin{cases} f(d_{max}^2) & \text{if } k = 1 \\ 1, & \text{if } k > 1 \end{cases}$$

Correspondingly, the gradient is now computed as:

$$\frac{\partial \tilde{D_G}}{\partial u_{kl}} = 2 \sum_{i=1}^{N} (e_{il} + e_{li}) \left(\frac{1}{c} - m_i u_{ki} \right)$$

This makes the first community the *residual* community, which should be regarded as a haven for outliers and should be discarded in the visualization process.

The definition of w makes sure that vertexes with less neighbors will likely be in the community as their neighbors, and vertexes far away will be very unlikely to be in the same communities. However, it still allows vertexes close to each other to be in the same community, even if they are not connected, if there are enough intermediate connections between them.

The addition the residual community also makes the algorithm tend to move outliers into the first communities since two vertexes will not incur a high similarity even if they are both added to the first community. On the other hand, connected vertexes are unlikely to be moved into the first community since that will result in a very small similarity, making the difference between the prior similarity of 1 and the posterior similarity large.

One other simple improvement is to add some heuristic in finding the number of the communities. We can take a guess of c by assuming $c_{min} = N/k$, where k can be some reasonable number, then start from c_{min} instead of 2.

3.2 Positioning Nodes

To show the communities more clearly, we implement a force directed-based algorithm to position the vertexes on a 2D plan while taking into consideration the communities they belong to.

In particular, given the graph and the community structure computed from Section 3.1, we construct a physics system with each vertex and community:

1. Each pair of vertexes i and j repel each other with a force F_r/d_{ij}
2. Each pair of vertexes i and j that are connected by an edge attracts each other with a force $F_a d_{ij}$
3. Each community also has a center, and two communities i and j repel each other with a force F_{rc}/d_{ij}
4. Each community center c attracts every vertex v in the community with force $F_{ac}d_{cv}$
5. A center point in the plan which always attracts vertexes towards it.

With this system, we can run the positioning algorithm as such:

1. Randomly position all vertexes and community centers
2. Initialize velocity of each node to be 0
3. Run for k iterations
    1. Apply forces according to the rules above
    2. Sum up all forces each node experiences
    3. Compute acceleration by assuming masses to be 1
    4. Compute new velocity of each node according to the acceleration
    5. Compute new positions of each node according to the velocity

The output will be the positions of each vertex. We ignore the community center in the output.

This way, vertexes in the same community will be even closer to each other, while communities will be many overlaps will also be closer. On the other hand, vertexes not in the same communities will be farther apart, and communities without much overlap will also be further apart, making the community structures more apparent. Lastly, outliers in the graph will be even further apart compared to those in the community since they do not experience attraction from the community center.

4 Complexity Analysis

The original fuzzyclust algorithm is claimed to run in $O(chN^2)$ time, where h is the number of steps for the iterative gradient decent algorithm to converge. Since we do not modify the gradient decent component, the run time should still be the same. However, since we are unsure of c before running the algorithm, we need to increment c from some small value. So the overall runtime is:

$$\sum_{k=c_{min}}^{c} O(kh_kN^2) = O(c^2h_{max}N^2)$$

Moreover, we also need to compute the shortest distance between every vertexes. This can be done by running depth-first search from every vertex, resulting in $O(NE + N^2)$ run time, where E is the number of edges. So the total time complexity is $O(c^2h_{max}N^2 + NE + N^2)$. Assuming that $E/N < c^2h_{max}$, which is likely in a social network graph which is usually sparse, the final time complexity will be $O(c^2h_{max}N^2)$.

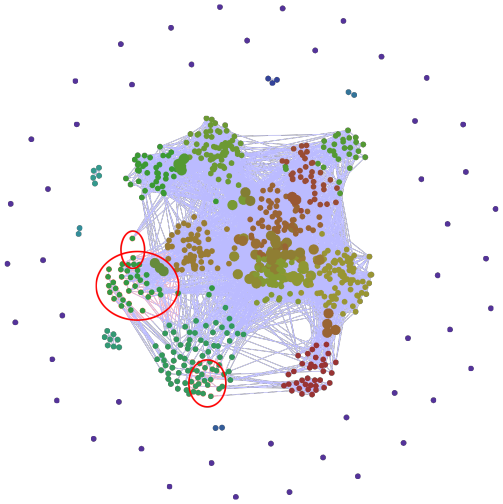
5 Results

We have implement a web interface for the above algorithms at `http://pals.yjyao.com`.

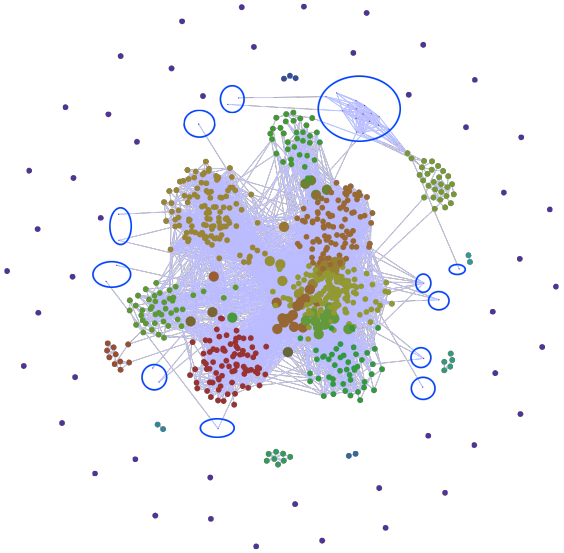
The following images show the final visualization of some real-life personal social networks from Facebook with both the original and improved version of fuzzyclust. Note that vertexes that are not supposed to be in the same communities are circled in red, and outliers circled in blue.

As seen, there's less tendency for sparse vertexes to be grouped into the same community with the new algorithm. Moreover, we can see that the new algorithm also help to identify outliers, which is not possible in the original algorithm.

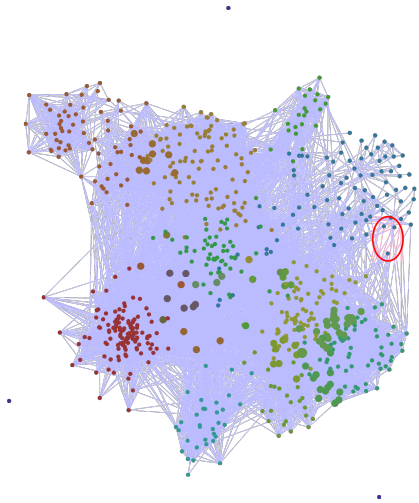
Interestingly, not only did it identify outliers who have only one or very few connections into some dense community, but also 'outlier bridges' - people who connect two or more otherwise very disconnected, but large, communities. This is because if the outlier is to be grouped into at least one of the communities, the dissimilarity will be large because most of them have a prior similarity of 0 and quite a large weight. This means that in the optimal solution, such vertex cannot be grouped into any one of these communities. Intuitively, this is sensible, because if we have a few large, mostly disconnected communities which



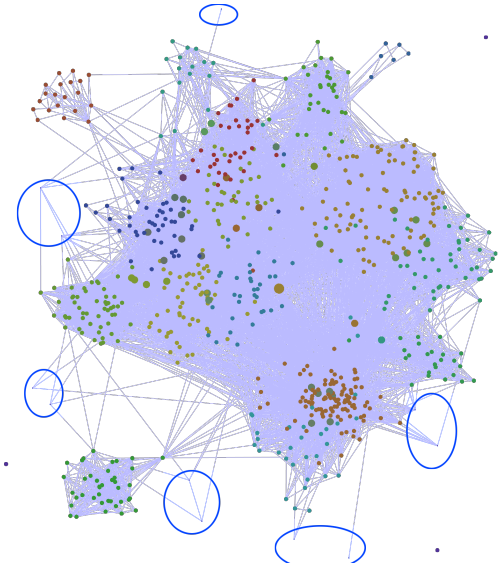
(a) Original Fuzzyclust result



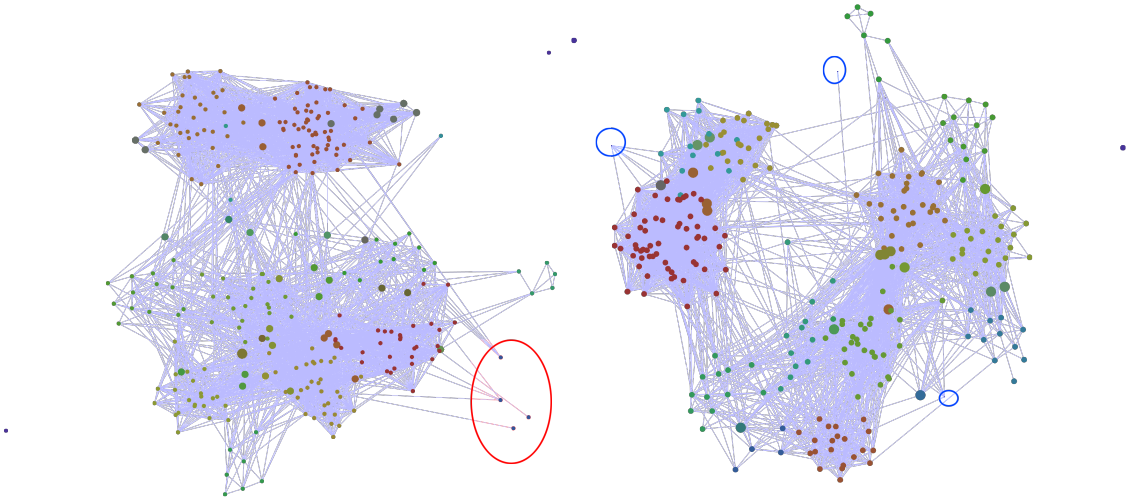
(b) Improved Fuzzyclust result



(c) Original Fuzzyclust result

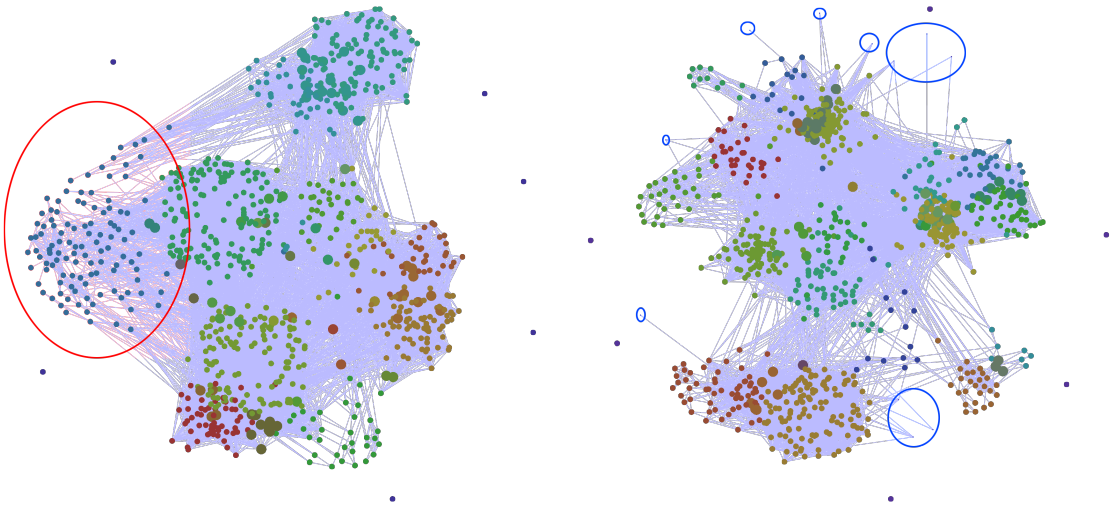


(d) Improved Fuzzyclust result



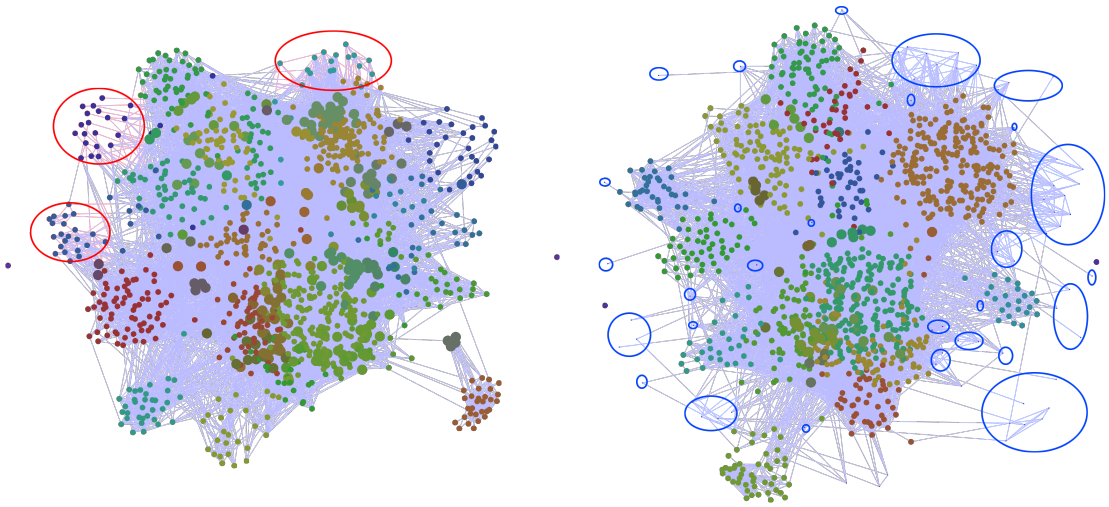
(e) Original Fuzzyclust result

(f) Improved Fuzzyclust result



(g) Original Fuzzyclust result

(h) Improved Fuzzyclust result



(i) Original Fuzzyclust result

(j) Improved Fuzzyclust result

happen to be connected by a person, this person is unlikely to belong to any other them, for otherwise he should have known a lot more people from all these communities. It is possible that this person, along with his connections both in and out of this social network, should form perhaps another community. However, due to the limitation of personal social network, this cannot be detected nor ascertained.

However, there may still be limitation to the new algorithm, as we can see that some small communities are also classified as outliers in large graph. Further tuning on the function $f(x)$ for weight and the depress factor m_1 may be needed.

6 Conclusion

In conclusion, we have improved the fuzzyclust algorithm to be able to detect outliers and prevent disconnected sparse vertexes to be grouped into communities. We have also implemented a force-directed-based graph visualization algorithm that displays communities better. Lastly, we have built an online visualization tool with these new algorithms.

7 Future work

We can further investigate the effect of $f(x)$ and m_1 on the output of the clustering algorithm so that the algorithm will correctly identify outliers.

8 Acknowledgment

Many thanks to my Supervisor Prof Lenong Hon-Wai, the students taking the same module and my friends who have volunteered their social network data. Without them, I could not have finished this project.

9 References

Nepusz, Tams, Andrea Petrczi, Lszl Ngyessy, and Flp Bazs. 2008. "Fuzzy Communities and the Concept of Bridgeness in Complex Networks." *Physical Review E* 77 (1): 016107.