# Centrality and Components

## CSL7390: Social Network Analysis: Programming Assignment 2

### Dataset

Zachary Karate Club dataset was used for understanding the algorithms. The successfully tested algorithms were run on Ego-Facebook dataset.

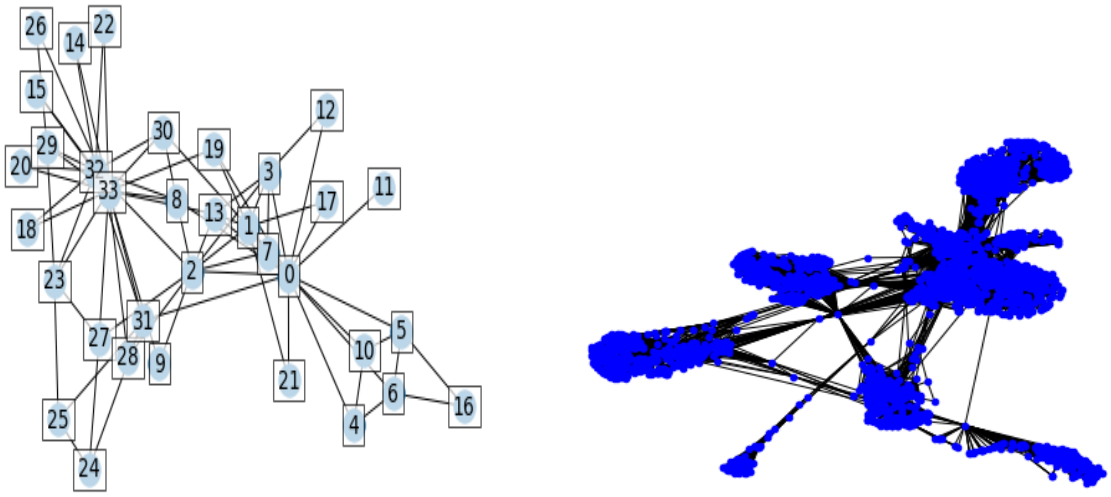


Figure 1: Left - Zachary Karate Club dataset, Right - Ego Facebook dataset

### Betweenness Centrality

This measure is the proportion of the number of shortest paths passing through a node. It is given as:

$$c_B(v) = \frac{Number\ of\ shortest\ paths\ passing\ through\ v}{Total\ number\ of\ shortest\ paths}$$

The nodes having higher values of this centrality measure may lead to disconnected components after their removal.

Algorithm

**Algorithm 1:** Betweenness centrality in unweighted graphs

$C_B[v] \leftarrow 0,\ v \in V;$
**for** $s \in V$ **do**
  $S \leftarrow$ empty stack;
  $P[w] \leftarrow$ empty list, $w \in V;$
  $\sigma[t] \leftarrow 0,\ t \in V;\quad \sigma[s] \leftarrow 1;$
  $d[t] \leftarrow -1,\ t \in V;\quad d[s] \leftarrow 0;$
  $Q \leftarrow$ empty queue;
  enqueue $s \rightarrow Q;$
  **while** $Q$ *not empty* **do**
    dequeue $v \leftarrow Q;$
    push $v \rightarrow S;$
    **foreach** *neighbor* $w$ *of* $v$ **do**
      *// $w$ found for the first time?*
      **if** $d[w] < 0$ **then**
        enqueue $w \rightarrow Q;$
        $d[w] \leftarrow d[v] + 1;$
      **end**
      *// shortest path to $w$ via $v$?*
      **if** $d[w] = d[v] + 1$ **then**
        $\sigma[w] \leftarrow \sigma[w] + \sigma[v];$
        append $v \rightarrow P[w];$
      **end**
    **end**
  **end**
  $\delta[v] \leftarrow 0,\ v \in V;$
  *// $S$ returns vertices in order of non-increasing distance from $s$*
  **while** $S$ *not empty* **do**
    pop $w \leftarrow S;$
    **for** $v \in P[w]$ **do** $\delta[v] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]} \cdot (1 + \delta[w]);$
    **if** $w \neq s$ **then** $C_B[w] \leftarrow C_B[w] + \delta[w];$
  **end**
**end**

Betweenness Centrality Algorithm[1]

In the above algorithm, the array d holds the shortest path of nodes from the node s, $\sigma$ arrays are used to find the ratio and P[v] contains the list of nodes lying on the shortest path from s to v. The algorithm is divided into two stages: *finding single source shortest paths* (till end of while loop) and *accumulation of dependencies*. The speciality of this algorithm is that it enables us to find betweenness centrality using few nodes (also known as pivots), instead of running over all nodes for shortest paths. In that case, it samples *k* number of random nodes from the graph and loops only over the sampled nodes yet finding dependencies of the remaining nodes. It works well when the graph has less components. However, it can also run over the complete set of nodes.

## Readings

The following are the betweenness centrality measures for the input set of nodes from Zachary karate club dataset (for easy illustrations). These can be compared with the dataset graph above. For instance, the node *0* has a high value that indicates the graph is susceptible to be disconnected on the removal of *0*. The nodes with zero values indicate that they are having degree 1.

```
================== Betweenness Centrality ====================
0 : 0.4825712481962482
2 : 0.13851937029020361
3 : 0.00789592352092352
4 : 0.0
5 : 0.01341540404040404
9 : 0.000741041366041366
11 : 0.0
12 : 0.0
13 : 0.029173180214846876
15 : 0.0
16 : 0.0
17 : 0.0
18 : 0.0
20 : 0.0
24 : 0.002683080808080808
25 : 0.012550855780022447
26 : 0.0
30 : 0.012648809523809524
31 : 0.08150177368927368
32 : 0.1660443722943723
```

The following are the betweenness centrality measures for Ego Facebook dataset. Randomly 100 nodes were chosen as input.

```
======================Betweenness Centrality==========================
32 : 0.0
76 : 0.0
137 : 0.0
```

165 : 0.0
177 : 0.0
194 : 0.0
239 : 0.0
262 : 0.0
311 : 0.0
327 : 0.0
340 : 0.0
459 : 5.530581227165313e-06
486 : 0.0
496 : 0.0
502 : 0.0
555 : 0.0
625 : 0.0
627 : 0.0
667 : 0.0
702 : 0.0
796 : 0.0
806 : 0.0
827 : 0.0
898 : 0.0
929 : 0.0
987 : 0.0
1009 : 0.0
1042 : 0.0
1065 : 0.0
1111 : 0.0
1167 : 0.0
1170 : 0.0
1193 : 0.006450082988420187
1225 : 0.0
1266 : 0.0
1270 : 0.0
1458 : 7.790083643049946e-05
1494 : 8.468633950651566e-05
1601 : 0.0
1646 : 0.0
1654 : 0.0
1689 : 0.0
1698 : 0.0
1763 : 0.0
1787 : 0.0
1896 : 0.0
1908 : 0.0
1921 : 0.0
1948 : 7.772743346532168e-05
1949 : 0.0

1987 : 0.0
1998 : 5.0772638913336884e-05
2004 : 5.307899096401067e-05
2073 : 0.0
2157 : 2.3570394556001287e-05
2201 : 0.0
2245 : 0.0
2258 : 0.0
2284 : 6.717852837199537e-05
2327 : 0.00015477889778879055
2451 : 9.652588873166129e-05
2467 : 0.0
2493 : 0.0
2524 : 0.0
2555 : 9.515903054661126e-05
2643 : 4.95446004601765e-05
2645 : 0.0
2676 : 0.0
2681 : 0.0
2695 : 0.0
2709 : 0.0
2712 : 0.0
2741 : 0.0
2757 : 0.0
2765 : 0.0
2852 : 0.0
2859 : 0.0
2906 : 0.0
3054 : 0.0
3064 : 0.0
3080 : 0.0
3203 : 0.0
3231 : 0.0
3254 : 0.0
3286 : 0.0
3294 : 0.0
3306 : 0.0
3410 : 0.0
3412 : 0.0
3452 : 0.0
3474 : 0.0
3501 : 0.0003602543516421199
3502 : 0.0
3536 : 0.0
3621 : 0.0
3698 : 0.0
3727 : 0.0

```
3750 : 0.0002783878731706841
3854 : 0.0
4017 : 0.0
```

## Closeness Centrality

Closeness centrality for a node is the sum of geodesic distances (shortest paths) between the node and other nodes. It is an inverse measure. It is given as follows:

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

Here *n* is the number of nodes reachable from *u. d(v, u)* is the shortest distance between *v* and *u*.

In case of weighted graphs, weights are considered instead.

### Algorithm

Input: Graph G, u : Graph Node
If u is None, then closeness centrality of all nodes is calculated. Otherwise, the algorithm can be used to find the closeness centrality measure of a single node by providing it in the parameter *u.* In this case, the set *V* is replaced with a list of the single node *u.*

For finding closeness measures, we again need single source shortest paths of the desired nodes. We use the stage one of *Algorithm 1 (shortest path finding)* with an additional variable *reachable* that counts the number of nodes reachable from *s*.

For the node *s,* we have the *d* array having the shortest paths from *s* to all other nodes. We also have the *reachable* count. The sum over all values in *d* is calculated and applied in the above formula.

### Readings

The following values of nodes are their closeness centrality measures from Zachary karate club dataset (for easy illustrations). The smaller the measure, the closer it is to other nodes.

```
====================== Closeness Centrality ====================
0 : 0.5689655172413793
1 : 0.4852941176470588
2 : 0.559322033898305
6 : 0.38372093023255816
8 : 0.515625
10 : 0.3793103448275862
12 : 0.3707865168539326
15 : 0.3707865168539326
```

```
16 : 0.28448275862068967
17 : 0.375
19 : 0.5
20 : 0.3707865168539326
22 : 0.3707865168539326
23 : 0.39285714285714285
24 : 0.375
25 : 0.375
28 : 0.4520547945205479
30 : 0.4583333333333333
31 : 0.5409836065573771
32 : 0.515625
```

The following are the closeness centrality measures for Ego Facebook dataset. Randomly 20 nodes were chosen as input.

```
================ Closeness Centrality ===================
198 : 0.2854920814479638
398 : 0.2729853975121687
487 : 0.27396702625686953
698 : 0.27118871725990595
785 : 0.17827027504304446
1278 : 0.3163337250293772
1375 : 0.31675556950109823
1498 : 0.3150749063670412
1743 : 0.3162098668754894
2388 : 0.259879006307118
2431 : 0.25986228200012873
2886 : 0.28263456288933997
3214 : 0.28368694674722494
3347 : 0.28394627663314814
3372 : 0.2825752274317705
3487 : 0.2600463678516229
3561 : 0.27762117566173944
3567 : 0.23940238335210767
3893 : 0.23928888888888888
3957 : 0.2394591709660203
```

## K-Core

A k-core in a graph is a group of nodes all having at least k degree. The algorithm for finding k-cores in a graph is straightforward.

### Algorithm

1. Remove all nodes with degree < k
2. Recalculate degrees
3. Repeat 1 and 2 till no more nodes are removed.

## Readings

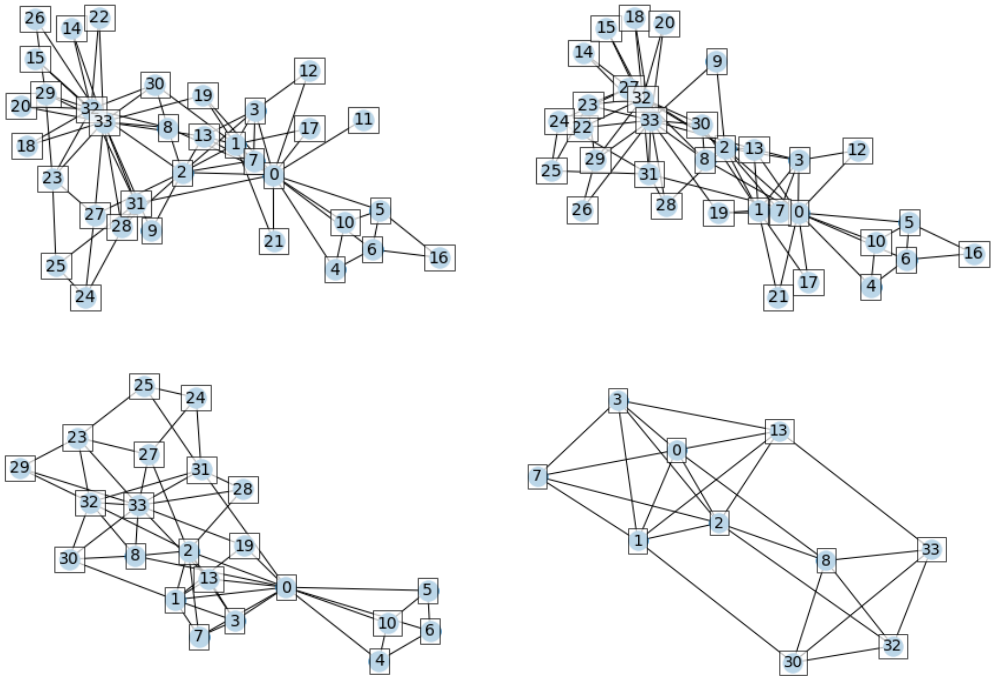

Figure 2: K-cores in Zakaray Karate Club network for k = 1 to 4 (top-left to bottom-right)

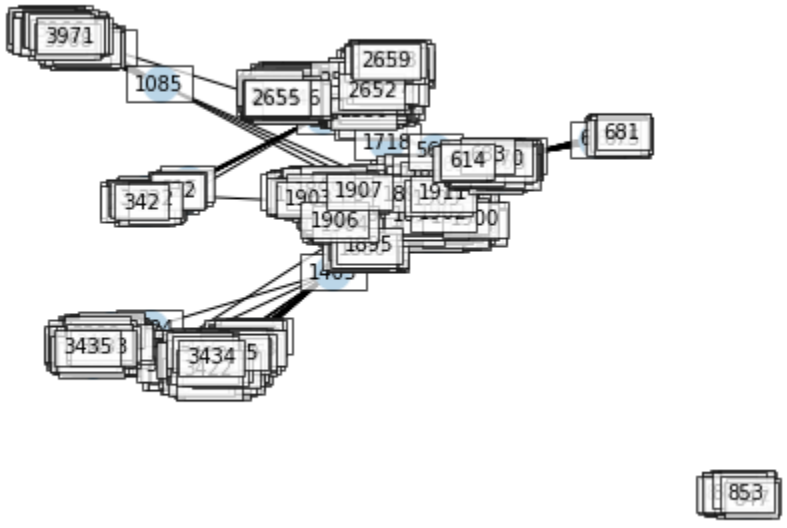


Figure 3: K-cores in Ego-Facebook network for random values of k (Here k = 19)

## Cliques

The computation of the largest clique in the Ego-facebook dataset did not stop for any available function in networkx. The minimum size of the clique is two (Dyads) and the number is equal to the number of edges in the network i.e. `88234`.

## References

[1] Ulrik Brandes: A Faster Algorithm for Betweenness Centrality. Journal of Mathematical Sociology 25(2):163-177, 2001. https://doi.org/10.1080/0022250X.2001.9990249