Project Report

Topic 4: Fair Community Detection

Rak Anastasia Stamatis Konstantinos

1.0 Introduction:

In this report, we would like to explain our code, walk you through our thought process for its creation and clarify all non-trivial spots.

However, before going into the code itself, we would like to remind you what our goal with this project was. By choosing the Fair Community Detection topic, we wanted to examine how different community detection algorithms perform regarding fairness by running them on graphs whose nodes carry certain 'sensitive' attributes. After that, as we talked about during our Project Proposal, we would like to produce a method to process the communities obtained by these algorithms to achieve fairness where it's necessary.

So, on a final note before going into the code, we shall mention that the datasets that we worked on are the Twitch Gamers Social Network from SNAP and two synthetic datasets generated using the stochastic block model.

2.0 The Code:

Our code begins with a cell that includes mainly module imports and a filter to ignore warnings. The latter was used to avoid overwhelming amounts of warnings. As for the module imports, the most notable ones are those related to the *dask* module, which are necessary for us to be able to run parallel computations on graphs as big as the one that is inferred from the Twitch Gamers Social Network dataset (trying to perform the K-means clustering and Spectral Clustering algorithms on this graph without using parallel computation was impossible due to memory limitations).

Right after that, we import the *Client* method from the *dask.distributed* module, and through the only other command in the cell we get four 'workers' which are going to be used to attain communities via the K-means algorithm. When running this cell, one also gets as output a link which leads to a page which shows details about the computations performed by each core (e.g. memory, usage, computations completed etc.).

In the following cell, we form two data frames from the csv files that you can download if you follow the link to the Twitch Gamers dataset. *large_twitch_edges.csv* contains lines of pairs of numbers. These numbers are the IDs of the Twitch users and when two numbers are paired in the same line this indicates that the corresponding users are mutual followers of each other. *large_twitch_features.csv* includes data on the Twitch users, such as their views, their language, and others. However, the only one of these attributes that we are interested in when considering the performance of clustering algorithms in respect to fairness is the users' affiliation with Twitch, which is represented by a binary digit. To have access to the contents of these data frames, we turn them into NumPy arrays in the following lines of code.

In the next cell, the graph that we have claimed can be inferred from *large_twitch_edges.csv* is finally defined formally as a NetworkX graph.

2.1 Functions:

Then we define a few useful functions for the process of our algorithm.

2.1.1 Clustering():

In this function we initially request the type of graph that is using this function. If it is a synthetic graph (*type_of_graph == 0*), then it goes through kmeans modeling from the '*sklearn'* library. We did not find it necessary to perform other types of clustering on our synthetic graphs, as we were defining the number of clusters on creation. If the type of graph is not synthetic, then it doesn't go through clustering, as it has been performed before calling this function. What this function returns for all types of clustered graphs (apart from propagation clustering) is a list of the clusters with the appended nodes of the graph (*dict_*), that we use for further computations, including modularity, which is performed and printed within *clustering().*

2.1.2 Node_affiliation():

In this function, we assign a binary key to each node in regard to its affiliation with twitch. Again, there is a distinction between whether the graph is synthetic or not. If it's the Twitch Graph, then the affiliation can be found in the dataset turned array 'large_twitch_features', otherwise the affiliation is applied randomly. Since the nodes of graphs generated using an SBM do not have attributes that can be used to judge the fairness performance of any clustering algorithm that is going to be ran on the graph, we must produce such attributes ourselves. In the following cell, for each of the two synthetic graphs we perform the following process: For each of the graph's nodes, we generate a number between 0 and 999. If the number is smaller than 500, the node gets the attribute '0', otherwise it gets the attribute '1'. Thus, we aim to get approximately half of the graphs' nodes to have attribute '0' and the rest to have attribute '1'. Some cells below, a calculation on how many nodes with attribute '1' and '0' exist in each synthetic graph is performed (expectedly, not exactly half of the nodes possess attribute '0' or '1'. One is slightly more prevalent).

2.1.3 Graph_fairness():

Here, we calculate the fairness of all the graph for the affiliated users. This is useful for comparison, as in many cases a 50-50% split would not be fair because there might have been fewer affiliated users in a whole. Regardless, in '*Twitch Graph'* the separation is very close to that 50-50% split, specifically, 48.5% of users are affiliated.

2.1.4 Find_cluster_fairness():

Similar logic as above, only for each cluster of the graph. This function returns an array *num_of_aff,* which contains the affiliated nodes, and prints the fairness of each cluster.

2.1.5 Improve_fairness():

In our final function, we perform an improvement of the cluster fairness. This is the core of our project. The logic is to compare the fairness of each cluster, to that of the overall graph. Then, for the clusters that are deemed unfair, one or more nodes are reassigned to a different cluster.

Specifically, a loop compares two clusters each time. If one cluster has fairness smaller by that of the graph by at least an assigned difference('*divergence*' in this function) and the other cluster is over the fairness by this '*divergence*', then a non-affiliated node from the first cluster is removed and added to the second. Then, the *find_cluster_fairness()* function is called within this function to measure the cluster fairness after this edit and the loop repeats itself until there are no clusters that can couple in the original condition.

Notably, we consider the community quality of a network to be important, thus before reassigning a node, we apply a condition that this node has connection to the other cluster. If the node does not connect, then no edit happens to it, and we move on to another non-affiliated node. Unfortunately, this condition was only able to be applied to our synthetic graphs. The reason is, that, due to the Twitch Graph size, especially the number of edges between nodes, which exceeds 6 million, the *networkX* function for checking connectivity (*is_connected(G)*) was taking too long to yield results. After many hours, the program was not able to check connectivity, even for the first pair of clusters. We couldn't find the source code of this function to edit it with dask arrays, or any other easily applicable connectivity algorithms. Also, considering the modularity of this graph, which was low for all three clustering methods used (see 'Results'), we decided to run the *improve_fairness()* algorithm without caring about the connectivity of the new clusters, as community structure was weak to begin with.

2.2 Graph Preparations:

As all the functions were defined, next we produced our graphs and clusters. To generate our own synthetic data, we used the *Stochastic Block Model* (SBM). Since randomness is involved in the generation of such graphs, we choose a seed for the random functions in the beginning of the related cell in order to get the same results every time we run the code. The one of the two graphs is generated with an assortative planted partition model, which means that the matrix *P1* that contains the probabilities of an edge existing between two communities features a constant *p* on every diagonal element and a constant *q* in every non-diagonal element such that $p>q$. The reason we choose $p>q$ is that we ideally want a higher concentration of edges between nodes of the same community. For the other graph we generate random probabilities of an edge forming between nodes of the same community in the range of 0.1 to 0.5 and probabilities of an edge forming between nodes of different communities in the range of 0.0 and 0.1. This version of the SBM is more generalized than the planted partition one since it does not feature just a constant probability on all elements on the main diagonal and another constant probability on all elements off the main diagonal. For both graphs we choose to have a total of 1000 nodes split into communities of size indicated by the *sizes* array.

To perform calculations to the '*Twitch Graph'*, as mentioned, *dask* was used. Using the *MinMaxScaler*, we scaled the array containing the edges. Otherwise, the calculation done later were other taking too long or returning 'NaN-Infinity' errors. Then, we split the scaled array into chunks that are going to be handled by the different workers that were initialized earlier. These chunks are kept into *dask_array*. Through the next line, by using the *persist()* method, we perform some computations on this array which are saved in memory for later use.

The next step is to run the K-means clustering algorithm on the Twitch graph using parallel computations, through the *dask-ml* library. It makes use of the *daskar* variable, which maintains the same information as the one that is contained in *edges_array* and in '*Twitch_Graph'*. It is the dask array with the *persist()* function applied. After obtaining the labels of the clusters produced by the K-means algorithm, we apply the *dask.compute()* function to the labels so we can use it as a numparray in our defined functions above. The output produces many *'BokehUserWarning'*, which, while being overwhelming and unfortunately, seemingly impossible to silence, are of no real consequence to our results.

Similarly, using the *dsk-ml* library we performed Spectral Clustering and applied the compute() function. Again, best ignore the *'BokehUserWarnings'*.

Lastly, we run the third of the three community detection algorithms that we set out to evaluate on their fairness performance, and this algorithm is '*Label Propagation'*. For this task we didn't use parallel computation since this cell runs in less than 30 minutes, and produces 79

clusters and modularity equal to 0.03567630678997064, which proves that there is not much in the way of community structure in our graph.

3.0 Results:

For each type of graph or type of clustering, we assigned a single cell for all the calculations. We have 5 cells, specifically, 3 for the Twitch Graph(kmeans, spectral clustering and label propagation) and 2 for our synthetic graphs (synthetic_2 and synthetic_2, both with kmeans). On each cell, we called we functions that we defined earlier. The only exception being, that for label propagation there was no need to call for the clustering function, as it produced the modularity and the labels list.

3.0.1 Twitch Graph- kmeans clustering:

```
Graph with 168114 nodes and 6797557 edges
The modularity is: 0.0158286905068509
The length of cluster   0    is 40513
The length of cluster   1    is 44646
The length of cluster   2    is 41170
The length of cluster   3    is 41785
48.50637067704058 % of the users are affiliated
Cluster  0  has  47.8685853923432 % affiliated streamers
Cluster  1  has  49.453478475115354 % affiliated streamers
Cluster  2  has  47.61233908185572 % affiliated streamers
Cluster  3  has  48.99365801124806 % affiliated streamers
Clusters have an average of  48.48201524014058 % affiliated streamers
```

Figure 1

As seen in '*figure 1*', we have a small modularity, which is why we believe the community structure is weak. We experimented with various numbers of clusters, but the modularity was even smaller. We stuck with 4 clusters, as it has higher modularity than any more than that and it gave us some room to work on fairness.

Comparing the graph fairness and the fairness of each cluster, we can see that, even before editing, the network appears to be fair in regards to affiliation. Despite that, we used a '*divergence*' of 0.8 and called the improve_fairness() function. The algorithm made changes only between cluster 1 and cluster 2. Then we called the find_cluster_fairness() function again, to see how it improved. The results are shown in '*figure 2*'. For smaller '*divergence*' there would be more edits, and beter fairness, but in the cost of calculation time.

```
Cluster  0  has  47.8685853923432 % affiliated streamers
Cluster  1  has  49.43178012458416 % affiliated streamers
Cluster  2  has  47.70735981308411 % affiliated streamers
Cluster  3  has  48.99365801124806 % affiliated streamers
Clusters have an average of  48.50034583531488 % affiliated streamers
```

Figure 2

Similarly, we worked on the other cells.

3.0.2 Label Propagation:

```
48.50637067704058 % of the users are affiliated
Cluster  0  has  48.853464592013296 % affiliated streamers
Cluster  1  has  35.69767441860465 % affiliated streamers
Cluster  2  has  45.08431434934776 % affiliated streamers
Cluster  3  has  59.375 % affiliated streamers
Cluster  4  has  0.0 % affiliated streamers
Cluster  5  has  39.02053712480253 % affiliated streamers
Cluster  6  has  41.202672605790646 % affiliated streamers
Cluster  7  has  0.0 % affiliated streamers
Cluster  8  has  0.0 % affiliated streamers
Cluster  9  has  0.0 % affiliated streamers
Cluster  10  has  0.0 % affiliated streamers
Cluster  11  has  0.0 % affiliated streamers
Cluster  12  has  0.0 % affiliated streamers
Cluster  13  has  0.0 % affiliated streamers
Cluster  14  has  0.0 % affiliated streamers
Cluster  15  has  0.0 % affiliated streamers
Cluster  16  has  50.0 % affiliated streamers
Cluster  17  has  0.0 % affiliated streamers
Cluster  18  has  0.0 % affiliated streamers
Cluster  19  has  50.0 % affiliated streamers
Cluster  20  has  0.0 % affiliated streamers
Cluster  21  has  0.0 % affiliated streamers
Cluster  22  has  0.0 % affiliated streamers
Cluster  23  has  33.333333333333336 % affiliated streamers
Cluster  24  has  66.66666666666667 % affiliated streamers
Cluster  25  has  0.0 % affiliated streamers
Cluster  26  has  0.0 % affiliated streamers
Cluster  27  has  0.0 % affiliated streamers
Cluster  28  has  0.0 % affiliated streamers
Cluster  29  has  0.0 % affiliated streamers
Cluster  30  has  33.333333333333336 % affiliated streamers
Cluster  31  has  0.0 % affiliated streamers
Cluster  32  has  0.0 % affiliated streamers
Cluster  33  has  100.0 % affiliated streamers
Cluster  34  has  50.0 % affiliated streamers
Cluster  35  has  0.0 % affiliated streamers
Cluster  36  has  0.0 % affiliated streamers
Cluster  37  has  100.0 % affiliated streamers
Cluster  38  has  50.0 % affiliated streamers
Cluster  39  has  0.0 % affiliated streamers
Cluster  40  has  0.0 % affiliated streamers
Cluster  41  has  0.0 % affiliated streamers
Cluster  42  has  50.0 % affiliated streamers
Cluster  43  has  0.0 % affiliated streamers
Cluster  44  has  0.0 % affiliated streamers
Cluster  45  has  0.0 % affiliated streamers
Cluster  46  has  0.0 % affiliated streamers
Cluster  47  has  0.0 % affiliated streamers
Cluster  48  has  50.0 % affiliated streamers
Cluster  49  has  0.0 % affiliated streamers
Cluster  50  has  0.0 % affiliated streamers
Cluster  51  has  100.0 % affiliated streamers
Cluster  52  has  0.0 % affiliated streamers
Cluster  53  has  0.0 % affiliated streamers
Cluster  54  has  0.0 % affiliated streamers
Cluster  55  has  0.0 % affiliated streamers
Cluster  56  has  50.0 % affiliated streamers
Cluster  57  has  0.0 % affiliated streamers
Cluster  58  has  0.0 % affiliated streamers
Cluster  59  has  0.0 % affiliated streamers
Cluster  60  has  0.0 % affiliated streamers
Cluster  61  has  0.0 % affiliated streamers
Cluster  62  has  0.0 % affiliated streamers
Cluster  63  has  0.0 % affiliated streamers
Cluster  64  has  0.0 % affiliated streamers
Cluster  65  has  0.0 % affiliated streamers
Cluster  66  has  0.0 % affiliated streamers
Cluster  67  has  50.0 % affiliated streamers
Cluster  68  has  0.0 % affiliated streamers
Cluster  69  has  0.0 % affiliated streamers
Cluster  70  has  0.0 % affiliated streamers
Cluster  71  has  50.0 % affiliated streamers
Cluster  72  has  0.0 % affiliated streamers
Cluster  73  has  50.0 % affiliated streamers
Cluster  74  has  0.0 % affiliated streamers
Cluster  75  has  0.0 % affiliated streamers
Cluster  76  has  0.0 % affiliated streamers
Cluster  77  has  0.0 % affiliated streamers
Cluster  78  has  0.0 % affiliated streamers
Clusters have an average of  15.222367043340409 % affiliated streamers
```

Figure 3

```
Cluster  0  has  48.84614183555569 % affiliated streamers
Cluster  1  has  38.3270911360799 % affiliated streamers
Cluster  2  has  45.08431434934776 % affiliated streamers
Cluster  3  has  48.78378378378378 % affiliated streamers
Cluster  4  has  0.0 % affiliated streamers
Cluster  5  has  39.02053712480253 % affiliated streamers
Cluster  6  has  41.202672605790646 % affiliated streamers
Cluster  7  has  0.0 % affiliated streamers
Cluster  8  has  0.0 % affiliated streamers
Cluster  9  has  0.0 % affiliated streamers
Cluster  10  has  0.0 % affiliated streamers
Cluster  11  has  0.0 % affiliated streamers
Cluster  12  has  0.0 % affiliated streamers
Cluster  13  has  0.0 % affiliated streamers
Cluster  14  has  0.0 % affiliated streamers
Cluster  15  has  0.0 % affiliated streamers
Cluster  16  has  33.333333333333336 % affiliated streamers
Cluster  17  has  0.0 % affiliated streamers
Cluster  18  has  0.0 % affiliated streamers
Cluster  19  has  33.333333333333336 % affiliated streamers
Cluster  20  has  0.0 % affiliated streamers
Cluster  21  has  0.0 % affiliated streamers
Cluster  22  has  0.0 % affiliated streamers
Cluster  23  has  33.333333333333336 % affiliated streamers
Cluster  24  has  40.0 % affiliated streamers
Cluster  25  has  0.0 % affiliated streamers
Cluster  26  has  0.0 % affiliated streamers
Cluster  27  has  0.0 % affiliated streamers
Cluster  28  has  0.0 % affiliated streamers
Cluster  29  has  0.0 % affiliated streamers
Cluster  30  has  33.333333333333336 % affiliated streamers
Cluster  31  has  0.0 % affiliated streamers
Cluster  32  has  0.0 % affiliated streamers
Cluster  33  has  40.0 % affiliated streamers
Cluster  34  has  33.333333333333336 % affiliated streamers
Cluster  35  has  0.0 % affiliated streamers
Cluster  36  has  0.0 % affiliated streamers
Cluster  37  has  40.0 % affiliated streamers
Cluster  38  has  33.333333333333336 % affiliated streamers
Cluster  39  has  0.0 % affiliated streamers
Cluster  40  has  0.0 % affiliated streamers
Cluster  41  has  0.0 % affiliated streamers
Cluster  42  has  33.333333333333336 % affiliated streamers
Cluster  43  has  0.0 % affiliated streamers
Cluster  44  has  0.0 % affiliated streamers
Cluster  45  has  0.0 % affiliated streamers
Cluster  46  has  0.0 % affiliated streamers
Cluster  47  has  0.0 % affiliated streamers
Cluster  48  has  33.333333333333336 % affiliated streamers
Cluster  49  has  0.0 % affiliated streamers
Cluster  50  has  0.0 % affiliated streamers
Cluster  51  has  40.0 % affiliated streamers
Cluster  52  has  0.0 % affiliated streamers
Cluster  53  has  0.0 % affiliated streamers
Cluster  54  has  0.0 % affiliated streamers
Cluster  55  has  0.0 % affiliated streamers
Cluster  56  has  33.333333333333336 % affiliated streamers
Cluster  57  has  0.0 % affiliated streamers
Cluster  58  has  0.0 % affiliated streamers
Cluster  59  has  0.0 % affiliated streamers
Cluster  60  has  0.0 % affiliated streamers
Cluster  61  has  0.0 % affiliated streamers
Cluster  62  has  0.0 % affiliated streamers
Cluster  63  has  0.0 % affiliated streamers
Cluster  64  has  0.0 % affiliated streamers
Cluster  65  has  0.0 % affiliated streamers
Cluster  66  has  0.0 % affiliated streamers
Cluster  67  has  33.333333333333336 % affiliated streamers
Cluster  68  has  0.0 % affiliated streamers
Cluster  69  has  0.0 % affiliated streamers
Cluster  70  has  0.0 % affiliated streamers
Cluster  71  has  33.333333333333336 % affiliated streamers
Cluster  72  has  0.0 % affiliated streamers
Cluster  73  has  33.333333333333336 % affiliated streamers
Cluster  74  has  0.0 % affiliated streamers
Cluster  75  has  0.0 % affiliated streamers
Cluster  76  has  0.0 % affiliated streamers
Cluster  77  has  0.0 % affiliated streamers
Cluster  78  has  0.0 % affiliated streamers
Clusters have an average of  10.395753681460256 % affiliated streamers
```

Figure 4

In '*figure 3'* we see the results of the cluster fairness. The clusters produced in label propagation are unequal in size, with the first cluster having over 100000 nodes, while most of the others having only 2 (clusters with 0% fairness are either of size 2 or 3). We applied the algorithm for '*divergence*' being 0.34 so we could make use of the first cluster, since it is the one with most nodes. The new values for fairness are shown in '*figure 4*'.

3.0.3 Spectral Clustering:

The same process as in Kmeans Clustering, with '*divergence*' having the value of 0.9 ('*figure 5*' is original fairness and '*figure 6*' is after the application of the algorithm).

```
Graph with 168114 nodes and 6797557 edges
The modularity is: 0.01577789908121674
The length of cluster   0    is 44283
The length of cluster   1    is 42103
The length of cluster   2    is 39742
The length of cluster   3    is 41986
48.50637067704058 % of the users are affiliated
Cluster  0  has  49.44786938554299 % affiliated streamers
Cluster  1  has  48.99888368999834 % affiliated streamers
Cluster  2  has  47.57435458708671 % affiliated streamers
Cluster  3  has  47.90168151288525 % affiliated streamers
Clusters have an average of  48.480697293878315 % affiliated streamers
```

Figure 5

```
Cluster  0  has  49.41773865944482 % affiliated streamers
Cluster  1  has  48.99888368999834 % affiliated streamers
Cluster  2  has  47.606697721264005 % affiliated streamers
Cluster  3  has  47.90168151288525 % affiliated streamers
Clusters have an average of  48.4812503958981 % affiliated streamers
```

Figure 6

3.0.4 Synthetic Graphs:

What follows is that we run the K-means clustering algorithm on both graphs, and after labeling nodes by the cluster that they belong to, we form dictionaries that have the K-means cluster labels as keys and the nodes in each cluster as values exactly like we mentioned before.

We then measure the percentage of 'affiliated' nodes in the clusters generated by the K-means algorithm for both synthetic graphs, as well as the average percentage of affiliated nodes in all clusters, using the *graph_fairness()* and *find_cluster_fairness()* functions (We only named the randomly generated attribute of the synthetic graphs' nodes '*affiliation*' for it to match the attribute of the users of the *Twitch* dataset. This attribute could be named any other way).

'*Figure 7*' and *'figure 8'* are the outputs for *synthetic_1* and *synthetic_2*. In both cases we used a value of 0.5 for '*divergence*'.

```
Graph named 'stochastic_block_model' with 1000 nodes and 51898 edges
The modularity is: 0.49418921900539003
The length of cluster   0    is 300
The length of cluster   1    is 400
The length of cluster   2    is 100
The length of cluster   3    is 200
51.800000000000004 % of the users are affiliated
Cluster  0  has  48.666666666666664 % affiliated streamers            Cluster  0  has  51.40845070422535 % affiliated streamers
Cluster  1  has  54.0 % affiliated streamers                          Cluster  1  has  53.089244851258584 % affiliated streamers
Cluster  2  has  49.0 % affiliated streamers                          Cluster  2  has  51.578947368421055 % affiliated streamers
Cluster  3  has  53.5 % affiliated streamers                          Cluster  3  has  53.5 % affiliated streamers
Clusters have an average of  51.29166666666667 % affiliated streamers  Clusters have an average of  52.394160730976246 % affiliated streamers
```

Figure 7

```
Graph named 'stochastic_block_model' with 1000 nodes and 55593 edges
The modularity is: 0.27032389639999443
The length of cluster   0    is 103
The length of cluster   1    is 397
The length of cluster   2    is 300
The length of cluster   3    is 200
51.800000000000004 % of the users are affiliated
Cluster  0  has  50.48543689320388 % affiliated streamers            Cluster  0  has  51.48514851485149 % affiliated streamers
Cluster  1  has  53.65239294710327 % affiliated streamers            Cluster  1  has  53.24074074074074 % affiliated streamers
Cluster  2  has  48.666666666666664 % affiliated streamers           Cluster  2  has  51.40845070422535 % affiliated streamers
Cluster  3  has  53.5 % affiliated streamers                         Cluster  3  has  53.5 % affiliated streamers
Clusters have an average of  51.57612412674346 % affiliated streamers Clusters have an average of  52.4085849899544 % affiliated streamers
```

Figure 8

3.1 Final Comments on The Results:

Now, we need to address what it means for the processing of the clusters to be necessary in order to achieve fairness. It's of course statistically, and sometimes even numerically, impossible for all clusters to have the exact same percentage of affiliated nodes as the overall percentage in the whole graph. This does not mean that the clustering is unfair. We consider a particular clustering to be unfair if one or more clusters show a much lower or higher concentration of affiliated nodes than the overall percentage of affiliated nodes in the graph. The number by which the percentage of affiliated nodes in a certain cluster should be off from the overall percentage of affiliated nodes for the cluster to be considered 'unfair' is not standard and is linked to factors like what the actual overall percentage of affiliates is, what is the number of nodes within the cluster and the entire graph etc. This means that if a certain cluster of a graph with an equal number of affiliates and non-affiliates does not contain equally as many affiliates and non-affiliates, it is not automatically considered to be unfair; if the percentage of affiliates within the cluster is 'close enough' (with what 'enough' is being determined by us using criteria like the ones mentioned before) to 50%, then the cluster is treated as if its fair and no nodes that belong to it are subjected to cluster transfers.

In our code, we have implemented the previously discussed method for increasing fairness. Both synthetic graphs were clustered using the K-Means algorithm with 4 clusters, aiming to replicate the real number of communities with which the two SBM graphs were 'built'. A brief check verifies that K-Means perfectly recovers the 4 communities for both graphs.

Through the cell that counts and prints the number of affiliated nodes in both synthetic graphs we know that there are exactly 51,8% affiliated nodes in *synthetic_1* and in *synthetic_2*. By that point we have already printed the percentages of affiliated nodes of all clusters for both graphs, as well as the average percentage for these clusters. Before the process of fairness improvement begins, we can tell just by looking at the percentage of affiliated nodes of all four clusters that some node transfers to other clusters are going to be performed. Right after the ending of this process, the percentages of affiliated nodes for all clusters are printed again and we can clearly see that all percentages have gotten closer to 51,8%.

4.0 Appendix:

4.1 Data used to produce the Twitch Graph from https://snap.stanford.edu/data/twitch_gamers.html

4.2 Code is exported as an html file as well as the jupyter notebook file.

4.3 The outputs in the 'Results' section are exported in text documents.