Project 2: SNAP

Chalkiopoulos Georgios | $p3352124$

November 1, 2022

High performance system for analysis and manipulation of large networks

SNAP (Stanford Network Analysis Project) is a C++ based general purpose, high performance system for analysis and manipulation of large networks. SNAP scales to massive networks with hundreds of millions of nodes, and billions of edges. You can use it for a variety of graph related tasks, such as calculating structural properties, generating graphs and applying graph algorithms. In this homework we will be using Snap.py, a Python interface for SNAP.

1 Euler Paths and Circuits

The first part of this homework requires that we familiarize ourselves with graph creation and graph traversal with SNAP. We have to develop two Python functions that examine whether a given graph has:

- an Euler Path
- and Euler Circuit

An Euler path is a path that uses every edge of a graph exactly once. An Euler path starts and ends at different vertices. If a graph has an Euler path, then it must have exactly two vertices with odd degree, and it is these odd vertices that will form the beginning and end of the path.

An Euler circuit is a circuit that uses every edge of a graph exactly once. An Euler circuit starts and ends at the same vertex. If a graph has an Euler circuit, then all of its vertices must be of even degree. In both cases the graph should be connected.

In addition to implementing the above functions, we should complete the test-case that is provided with this homework. In particular, we should complete all four tests by filling in code that creates graphs that satisfy the tests' assertions:

- A graph that has an Euler path (but not an Euler circuit),
- A graph that does not have an Euler path,
- A graph that has an Euler circuit, and
- A graph that does not have an Euler circuit.

1.1 Implement functions that test if a given graph has an Euler Path and/or an Euler circuit

The code has been implemented in the `project2-1.py` file under the `has_euler_path` and `has_euler_circuit` functions. Both functions contain docstrings and comments, which explain step by step their functionality. In order to make the code cleaner, the `check.py` file was implemented, which contains the two required checks, apart from the connected graph check. The functions check and return:

- the odd vertices
- a boolean flag if the is at least one odd degree node or not

1.2 Implement functions that generate graphs which satisfy the unittest

The main unittest code is located under the file (`project2-1.py`), implemented under the class `TestEulerMethods`. We decided to have 1000 nodes, since it was the minimum requirement for the number of nodes (function `test_has_euler_circuit`). However, the code is working for any number of nodes over 1000, and one would have to change the number of nodes in the `TestEulerMethods` by changing the line 68 shown below:

```
67| class TestEulerMethods(unittest.TestCase):
68|     NODES = 1000
```

The code that generates the graphs, needed to successfully pass the unittest, is located under the `helper_classes.py` file. Specifically, the class `GenerateGraph` uses the `GenCircle` graph generator to create the required graphs for every case.

1.3 Running the unittest

Having written all the required code we executed the `project2-1.py` file, which resulted in all the tests succeding, as shown below:

```
Ran 4 tests in 0.024s

OK
```

2 Apply node centrality measures and community detection algorithms on generated graphs

For the second part of this homework, we will have to write a Python script `project2-2.py` that generates a graph of given size, reports some information on the graph, and compares the execution times of two community detection algorithms.

2.1 Generate the graphs and compare the two algorithms

For this part of the project the following files were created:

- `project2-2.py`
- `helper_classes.py`, where the `AlgorithmComparison` class was implemented
- `GLOBALS.py`, where global variables are defined.

The implementations uses the `GenSmallWorld` class to generate the required graph. We started from 50 nodes and gradually increased the number of nodes by a step of 20. In each step we calculated and saved into a pandas DataFrame the following metrics:

- The input values (Nodes, NodeOutDeg, RewireProb) to the GenSmallWorld generator. For the NodeOutDeg and RewireProb we used python's random library to randomize the values in each iteration
- The nodeID and degree of the node with the highest degree
- The nodeID and Scores of the nodes with the highest Hub and Authority Scores
- The execution time and the modularity scores for the Girvan-Newman community detection algorithm based on betweenness centrality and the Clauset-Newman-Moore community detection method

In order to test whether the algorithm is appropriate or not, the `exit_after` decorator was created, which uses python's threading library and stoped the execution of the community detection algorithm after 10 minutes. Moreover, the implementation stopped the algorithm if a memory error occurred. In either of those cases, the corresponding value in the DataFrame will be marked as `NaN`.

A small sample from the run may be found below. The results can also be found in the `project2-2_{algorithm}.csv` files and are presented in the appendix. Multiple runs, for various nodes have been executed in order to find the max number of nodes that the our system was able to reach (32GB RAM).

```
*******************************************************************
Nodes NodeOutDeg RewireProb highest_deg_ID highest_deg_degree
   50         14       0.25              3                 30
   70         15       0.18             33                 34
   90         18       0.01             28                 38
  110         12       0.93             65                 30
*******************************************************************
Nodes highest_hubscore_ID highest_hubscore_degree
   50                   3                0.162618
   70                  33                0.143466
   90                  28                0.110721
  110                  70                0.128485
*******************************************************************
Nodes highest_authscore_ID highest_authscore_degree
   50                    3                 0.162618
   70                   33                 0.143466
   90                   28                 0.110721
  110                   70                 0.128485
*******************************************************************
Nodes CommunityCNM_time_s CommunityCNM_modularity
   50            0.003997                0.153697
   70            0.000996                0.227496
   90               0.002                0.285491
  110               0.003                0.138021
*******************************************************************
Nodes  CommunityGirvanNewman_time_s CommunityGirvanNewman_modularity
  50                       1.174005                         0.126976
  70                       3.713012                         0.173867
  90                      10.818001                         0.293837
 110                       9.795999                         0.032134
```

We observed that the GirvanNewman algorithm cannot run, with the given restrictions of time and memory, for more than a few hundred nodes. On the other hand, the Clauset-Newman-Moore community method was able to compute the modularity for nodes having thousands of nodes. For this reason, the answers can be answered as follows:

Which of the two community detection algorithms are suitable for a network portraying the relationships of the employees of a medium enterprise (50 employees)?

- Either algorithm can be used, although the Clauset-Newman-Moore achieved higher modularity scores

Which of the two community detection algorithms would you use for the e-mail network of the employees of Google in NY (18,000 employees)?

- In this case only the Clauset-Newman-Moore can be used

Which of the two community detection algorithms are suitable for the friendship graph of a social network such as Facebook (1 billion users)?

- Based on the simulations executed, it seems that neither algorithm is suitable for networks of this size

2.2 Plots and results

After executing the program we fetched the maximum nodes for both algorithms, which can be found in Table 1.

Table 1: Max nodes reached for each method.

Algorithm	Max Nodes	Time (s)
Girvan Newman	450	247.94
Clauset Newman Moore	26050	451.69

In order to illustrate the change of moduarity, a scatter plot with the number of nodes vs the modularity score, is presented in Figure 1 for the Clauset Newman Moore algorithm. Moreover, a regression line a added, which shows that the modularity increases as the number of nodes increases.

Finally, the largest graph generated (26050 nodes) was used in order to calculate the PageRank, Betweeness, Closeness and Hub & Authorities scores using the class `CompareMetrics`. The results were saved into a DataFrame and only the top 30, by PageRank score, nodes were saved.

The top 30 nodes are also printed in the logs and presented below (only first and last 5). We may observe that due to the small number of out-degree, compared to the total number of nodes, present in the network, the PageRank values are small.

On the other hand, and only for demonstration purposes, we executed the algorithm for 500 nodes, in which case the PageRank vaslues are higher.

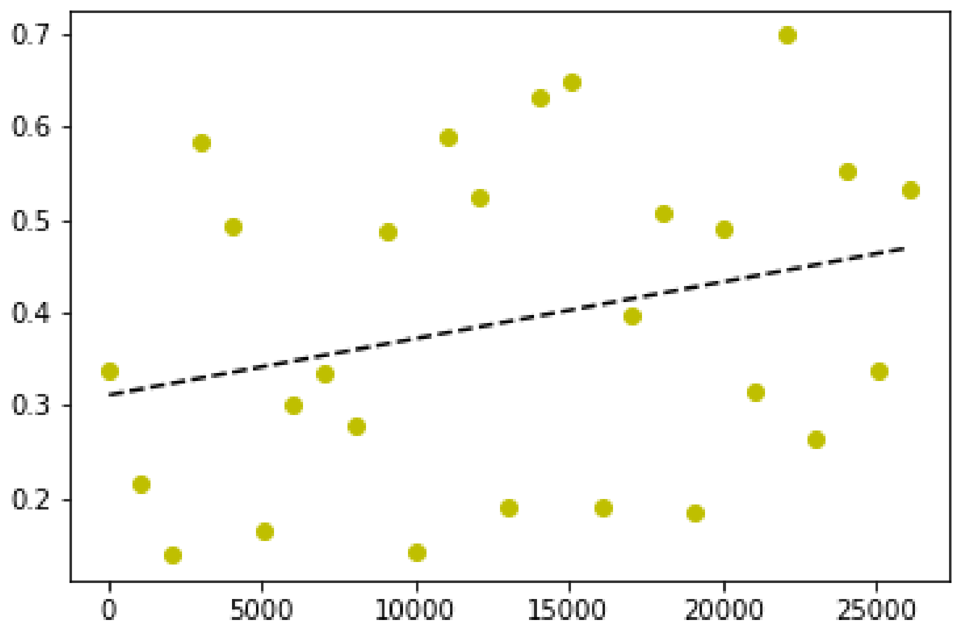

Figure 1: Nodes vs modulatiry Score for the CommunityCNM algorithm.

```
 Generating plots for 26050 nodes
    ID PageRank  BetweennessCentr  ClosenessCentr  HubScore  AuthScore
17817  0.000045     131707.031789        0.264387  0.007248   0.007248
7527   0.000045     119331.137021        0.262919  0.007369   0.007369
17351  0.000045     150583.522720        0.264847  0.007280   0.007280
20236  0.000045     101074.055350        0.260157  0.007491   0.007491
16397  0.000045     155578.041870        0.264406  0.007550   0.007550

3926   0.000044      82382.410653        0.261965  0.007177   0.007177
1802   0.000044      97859.866911        0.263127  0.007280   0.007280
13090  0.000044     135014.670078        0.264215  0.007372   0.007372
16599  0.000044      81338.131000        0.260819  0.007304   0.007304
568    0.000044      85882.344300        0.261888  0.007262   0.007262
```

```
 Generating plots for 500 nodes
 ID  PageRank  BetweennessCentr  ClosenessCentr  HubScore  AuthScore
222  0.002796        840.519119        0.405361  0.058931   0.058931
53   0.002759        844.681384        0.411716  0.065606   0.065606
254  0.002649        857.642367        0.415487  0.062725   0.062725
93   0.002626        709.142469        0.409016  0.067261   0.067261
343  0.002566        743.032946        0.403722  0.056254   0.056254

276  0.002450        576.585681        0.393533  0.054815   0.054815
279  0.002444        663.689223        0.400482  0.056002   0.056002
284  0.002439        647.245892        0.401448  0.056793   0.056793
13   0.002435        594.120122        0.397610  0.057070   0.057070
92   0.002431        608.246812        0.402419  0.057943   0.057943
```

Finally, two graphs were generates for the 26050 node network case:

- Betweenness, Closeness and PageRanK

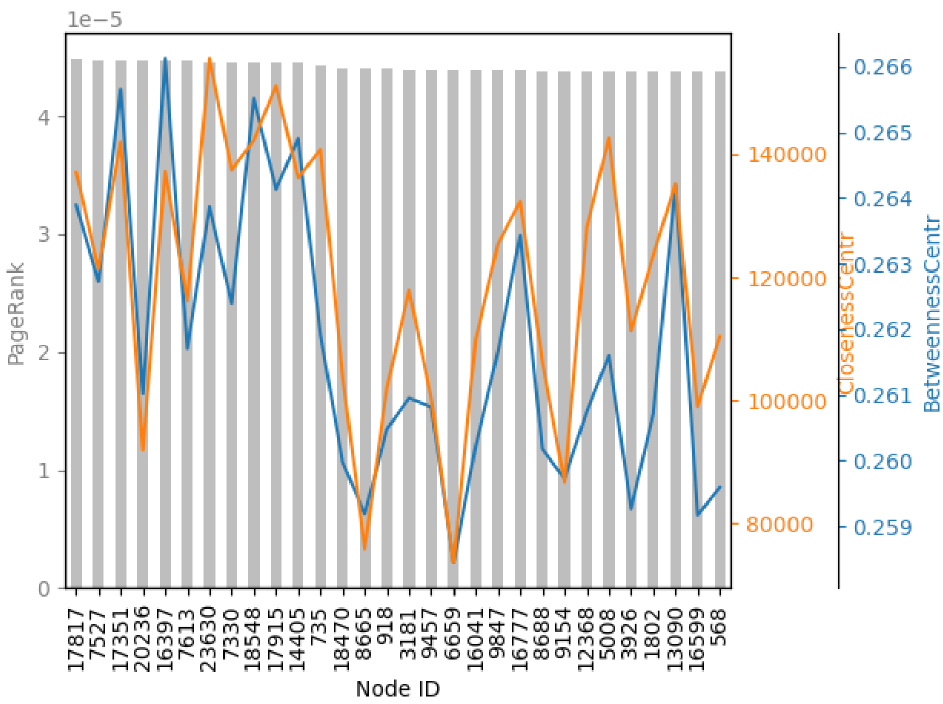

Figure 2: Betweenness, Closeness and PageRank scores ranked by decreasing order of PageRank.

- PageRank, Authority score and Hub score

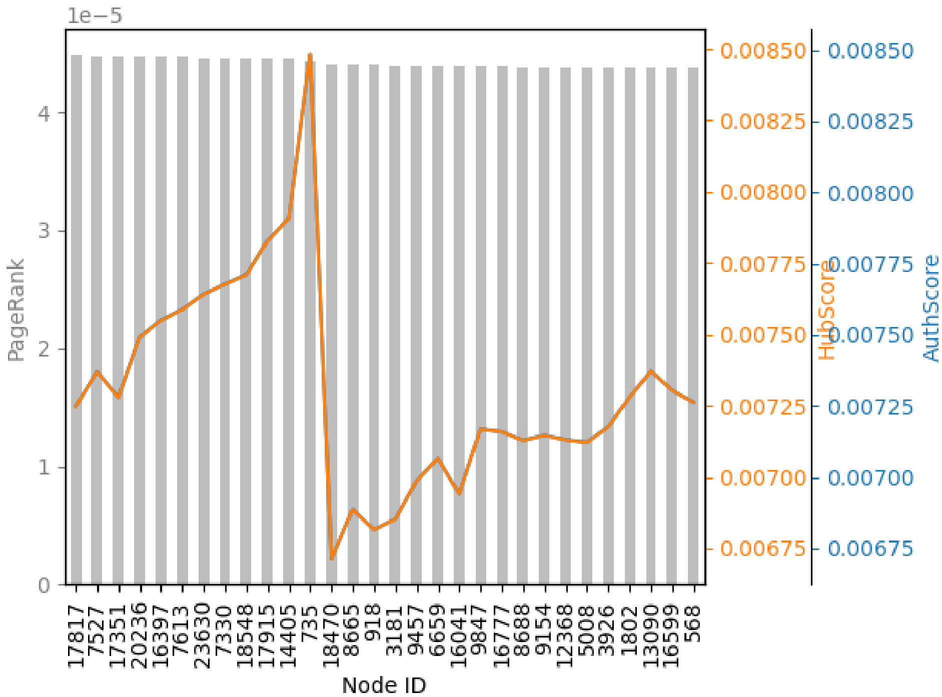

Figure 3: PageRank, Authority and Hub scores ranked by decreasing order of PageRank.

As seen in the graph, the Authority and Hub scores match.

Appendix

The runs executed are presented below. The node iteration was 1000, starting from 50. The CommunityCNM algorithm has a timeout error at 27050 nodes, while the CommunityGirvanNewman could not reach more than 50. For this reason, a separate run was executed, only for the CommunityGirvanNewman which is included in the `zip` file names `CommunityGirvanNewman_1.csv`.

Nodes	NodeOutDeg	RewireProb	highest_deg_ID	highest_deg_degree
50	7	0.02	8	16
1050	8	0.97	349	28
2050	17	0.98	1108	49
3050	7	0.23	343	21
4050	20	0.23	2512	52
5050	14	0.78	3625	43
6050	10	0.56	5235	34
7050	17	0.52	1484	50
8050	12	0.58	3642	38
9050	19	0.24	3814	50
10050	16	0.9	5160	48
11050	9	0.18	10465	26
12050	17	0.23	10768	44
13050	17	0.69	5103	48
14050	8	0.05	6867	20
15050	7	0.13	2763	20
16050	9	0.99	6265	32
17050	14	0.45	8362	41
18050	9	0.32	13819	27
19050	15	0.71	4753	49
20050	20	0.27	5254	52
21050	5	0.68	3071	20
22050	6	0.06	7942	16
23050	5	0.91	5714	22
24050	16	0.15	2513	42
25050	5	0.63	10800	20
26050	15	0.19	12133	39
27050	16	0.86	21911	53

Nodes	highest_hubscore_ID	highest_hubscore_degree
50	8	0.165835
1050	349	0.056619

```
 2050                 1108                 0.031827
 3050                 1746                 0.030866
 4050                 3089                 0.020865
 5050                 3625                 0.021381
 6050                 4466                 0.021542
 7050                 1484                  0.01794
 8050                 3642                 0.017712
 9050                 3814                 0.013766
10050                 7501                 0.015314
11050                10465                 0.015673
12050                11682                 0.012507
13050                 6521                 0.012598
14050                12163                 0.015267
15050                 9733                 0.014677
16050                 6265                 0.015015
17050                 9831                 0.011205
18050                14927                 0.011651
19050                 4753                 0.011921
20050                11599                 0.009458
21050                11031                  0.01579
22050                 1183                 0.012859
23050                 1392                 0.015696
24050                21356                 0.008905
25050                10800                 0.014256
26050                15577                 0.008784
27050                21911                 0.010075
-------------------------------------------------------------------
Nodes highest_authscore_ID highest_authscore_degree
   50                    8                 0.165835
 1050                  349                 0.056619
 2050                 1108                 0.031827
 3050                 1746                 0.030865
 4050                 3089                 0.020865
 5050                 3625                 0.021381
 6050                 4466                 0.021542
 7050                 1484                  0.01794
 8050                 3642                 0.017712
 9050                 3814                 0.013766
10050                 7501                 0.015314
11050                10465                 0.015672
12050                11682                 0.012507
13050                 6521                 0.012598
```

```
14050              12163                0.015206
15050               9733                0.014672
16050               6265                0.015015
17050               9831                0.011205
18050              14927                0.011651
19050               4753                0.011921
20050              11599                0.009458
21050              11031                 0.01579
22050               1183                0.012816
23050               1392                0.015696
24050              21356                0.008905
25050              10800                0.014256
26050              15577                0.008784
27050              21911                0.010075
-------------------------------------------------------------------
Nodes CommunityCNM_time_s CommunityCNM_modularity
   50                0.0                0.337218
 1050           0.218012                0.215835
 2050           1.375136                0.139071
 3050           1.148904                0.584937
 4050           6.032226                0.493149
 5050          15.065178                0.166195
 6050          18.994812                 0.30089
 7050             25.724                0.333517
 8050          35.498995                0.278361
 9050          43.423997                0.488393
10050             60.913                0.142859
11050          32.568001                0.590779
12050             76.281                 0.52545
13050         145.007994                0.191016
14050          10.879997                0.633173
15050          41.994999                0.647814
16050         108.967003                0.190374
17050         230.931001                0.396991
18050         160.175993                0.507803
19050         327.993999                0.185375
20050         380.669001                0.490629
21050         362.491003                0.316211
22050          26.315001                0.698954
23050              81.59                0.264904
24050         327.938994                0.553903
25050         494.655004                0.336995
```

```
26050          451.685994                    0.532494
27050                 NaN                         NaN
-----------------------------------------------------------------
Nodes CommunityGirvanNewman_time_s CommunityGirvanNewman_modularity
   50                     0.340006                         0.351282
 1050                          NaN                              NaN
 2050                          NaN                              NaN
 3050                          NaN                              NaN
 4050                          NaN                              NaN
 5050                          NaN                              NaN
 6050                          NaN                              NaN
 7050                          NaN                              NaN
 8050                          NaN                              NaN
 9050                          NaN                              NaN
10050                          NaN                              NaN
11050                          NaN                              NaN
12050                          NaN                              NaN
13050                          NaN                              NaN
14050                          NaN                              NaN
15050                          NaN                              NaN
16050                          NaN                              NaN
17050                          NaN                              NaN
18050                          NaN                              NaN
19050                          NaN                              NaN
20050                          NaN                              NaN
21050                          NaN                              NaN
22050                          NaN                              NaN
23050                          NaN                              NaN
24050                          NaN                              NaN
25050                          NaN                              NaN
26050                          NaN                              NaN
27050                          NaN                              NaN
-----------------------------------------------------------------
```