Community_centrality_calculations

November 25, 2017

In [1]:
```
import pandas as pd
pd.set_option('display.max_rows', 10)
pd.set_option('display.max_columns', 100)

# To remove pandas copy warnings (may need to turn on if writing new functions):
import warnings
warnings.filterwarnings('ignore')

import numpy as np
from Bio.PDB import *
import community
import networkx as nx
import igraph as ig
from sklearn.metrics.cluster import normalized_mutual_info_score
import plotly.plotly as py
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)
%matplotlib inline
from IPython.display import Image
```

In [2]:
```
ThTh_edges = pd.read_csv('../Ring_output/1VY4/1VY4_LSU_rRNA+rProtein_edges.txt', sep='\t
ThTh_edges
```

Out[2]:

	NodeId1	Interaction	NodeId2	Distance	Angle	Energy	Atom1	\
0	0:3001:_:MG	IAC:LIG_MC	0:21:_:LEU	5.220	-999.900	0.0	MG	
1	0:3001:_:MG	IAC:LIG_SC	0:22:_:GLY	3.071	-999.900	0.0	MG	
2	0:3001:_:MG	IAC:LIG_SC	0:23:_:VAL	1.977	-999.900	0.0	MG	
3	0:3001:_:MG	IAC:LIG_MC	0:24:_:LYS	6.141	-999.900	0.0	MG	
4	0:3001:_:MG	IAC:LIG_SC	0:26:_:TYR	4.767	-999.900	0.0	MG	
...	...	...	...	...	...	...	...	...
40442	Z:152:_:ALA	VDW:SC_SC	Z:155:_:LEU	3.720	-999.900	6.0	CB	
40443	Z:153:_:SER	VDW:SC_SC	Z:167:_:PRO	3.510	-999.900	6.0	CB	
40444	Z:157:_:LEU	VDW:MC_SC	Z:161:_:VAL	3.313	-999.900	6.0	C	
40445	Z:158:_:PRO	VDW:MC_SC	Z:161:_:VAL	3.589	-999.900	6.0	C	
40446	Z:166:_:SER	HBOND:MC_MC	Z:169:_:GLU	3.000	10.706	17.0	O	

```
      Atom2        Donor Positive Cation Orientation
0         O          NaN      NaN    NaN         NaN
1       HA2          NaN      NaN    NaN         NaN
2         H          NaN      NaN    NaN         NaN
3         N          NaN      NaN    NaN         NaN
4       HE1          NaN      NaN    NaN         NaN
...     ...          ...      ...    ...         ...
40442   CD2          NaN      NaN    NaN         NaN
40443    CB          NaN      NaN    NaN         NaN
40444   CG1          NaN      NaN    NaN         NaN
40445   CG1          NaN      NaN    NaN         NaN
40446     N  Z:169:_:GLU      NaN    NaN         NaN

[40447 rows x 12 columns]
```

In [3]:
```
ThTh_nodes = pd.read_csv('../Ring_output/1VY4/1VY4_LSU_rRNA+rProtein_nodes_xyz_modified.
ThTh_nodes
```

Out[3]:
```
          NodeId Chain  Position Residue Dssp  Degree  Bfactor_CA    Rapdf  \
0        B:1:_:U     B         1       U  NaN       2     -999.90 -999.900
1        B:2:_:C     B         2       C  NaN       3     -999.90 -999.900
2        B:3:_:C     B         3       C  NaN       6     -999.90 -999.900
3        B:4:_:C     B         4       C  NaN      11     -999.90 -999.900
4        B:5:_:C     B         5       C  NaN      16     -999.90 -999.900
...          ...   ...       ...     ...  ...     ...         ...      ...
7369  3:55:_:ARG     3        55     ARG    E       1       32.68  -16.082
7370  3:56:_:VAL     3        56     VAL    E       1       53.08 -168.072
7371  3:57:_:GLU     3        57     GLU    E       2       61.18  -67.546
7372  3:59:_:VAL     3        59     VAL    E       3       47.99   -0.117
7373  3:60:_:GLU     3        60     GLU            3       80.20   -2.371

          Tap  Accessibility rProtein           x           y           z
0    -999.900       -999.900       5S -140.216003  170.479996  158.654007
1    -999.900       -999.900       5S -143.912003  165.106995  157.557999
2    -999.900       -999.900       5S -144.324005  161.835999  152.091003
3    -999.900       -999.900       5S -143.281998  161.699997  145.934006
4    -999.900       -999.900       5S -139.960999  163.332001  140.940002
...       ...            ...      ...         ...         ...         ...
7369   -0.278          0.423     uL30  -97.103996  140.802002  196.343002
7370    0.394          0.339     uL30 -100.068001  143.119995  196.835007
7371   -0.069          0.408     uL30 -102.952003  143.319000  199.233994
7372    0.493          0.155     uL30 -105.612999  147.953003  202.595001
7373    0.000          0.834     uL30 -107.748001  150.294006  204.746994

[7374 rows x 14 columns]
```

In [4]:
```
def plot_nodes(df):
```

```
    data = []

    for rPro in set(df['rProtein']):

        rPro_df = df[df['rProtein'] == rPro]
        data.append(
        go.Scatter3d(
            x = rPro_df['x'],
            y = rPro_df['y'],
            z = rPro_df['z'],
            text =
                rPro_df['Residue']
                +' '+rPro_df['Dssp'],
            mode = 'markers',
            name = rPro
            )
        )

    layout = go.Layout(
        title = 'Thermus thermophilus Nodes (Atoms) Colored by rProtein',
        showlegend = True
    )

    fig = go.Figure(data=data, layout=layout)
    iplot(fig)
```

```
In [5]: plot_nodes(ThTh_nodes)
```

Unofortunately, I only modified the txt files that I use to make the plots, the xml file still does not have an updated x, y, z. However, I still try and use the dataframe as often as possible

```
In [6]: G_ThTh = nx.read_graphml('../Ring_output/1VY4/1VY4_LSU_rRNA+rProtein_network.xml')
```

```
In [7]: G_ThTh.nodes['n0']
```

```
Out[7]: {'Accessibility': -999.9,
         'Bfactor_CA': -999.9,
         'Degree': 2.0,
         'NodeId': 'B:1:_:U',
         'Position': 1.0,
         'Rapdf': -999.9,
         'Residue': 'U',
         'Tap': -999.9,
         'name': 'B:1:_:U',
         'pdbFileName': '1VY4_LSU_rRNA+rProtein.pdb#1.B',
         'x': -999.9,
         'y': -999.9,
         'z': -999.9}
```

```
In [8]: G_ThTh.edges[('n0', 'n118', 0)]
```

```
Out[8]: {'Angle': -999.9,
         'Atom1': 'O4',
         'Atom2': 'O6',
         'Cation': 'None',
         'Distance': 5.796,
         'Donor': 'None',
         'Energy': 0.0,
         'Interaction': 'IAC:LIG_LIG',
         'NodeId1': 'B:1:_:U',
         'NodeId2': 'B:119:_:G',
         'Orientation': 'None',
         'Positive': 'None'}
```

```
In [9]: def print_top_bottom_5(metric):
            top5 = {key: metric[key] for key in sorted(metric, key=metric.get, reverse=True)[:5]
            bottom5 = {key: metric[key] for key in sorted(metric, key=metric.get, reverse=False)
            print('top5:')
            for x in top5:
                print(x, '\t', top5[x])
            print('bottom5:')
            for x in bottom5:
                print(x, '\t', bottom5[x])
```

```
In [10]: def print_centrality(graph):
             degree = nx.degree_centrality(graph)
             #closeness = nx.closeness_centrality(graph) #takes a long time
             #harmonic = nx.harmonic_centrality(graph) #takes a long time
             #betweenness = nx.betweenness_centrality(graph) #takes a long time
             eigenvector = nx.eigenvector_centrality_numpy(graph)
             # pagerank_085 = nx.pagerank_numpy(graph, alpha=0.85) #takes a long time
             # Katz does not work on multigraph
             print('degree:')
             print_top_bottom_5(degree)
             #print('\ncloseness:')
             #print_top_bottom_5(closeness)
             #print('\nharmonic:')
             #print_top_bottom_5(harmonic)
             #print('\nbetweenness:')
             #print_top_bottom_5(betweenness)
             print('\neigenvector:')
             print_top_bottom_5(eigenvector)
             #print('\npagerank alpha=0.85:')
             #print_top_bottom_5(pagerank_085)
```

0.0.1 Takes a while to run

```
In [11]: print_centrality(G_ThTh)
```

```
degree:
top5:
n2136           0.005425200054252
n1497           0.0050183100501831
n1745           0.0050183100501831
n2027           0.0047470500474705
n1775           0.0046114200461142
bottom5:
n1074           0.0001356300013563
n1075           0.0001356300013563
n1133           0.0001356300013563
n1214           0.0001356300013563
n1482           0.0001356300013563

eigenvector:
top5:
n1501           0.125720188589
n1751           0.123478124241
n1500           0.120049611853
n6221           0.11919371863
n1752           0.118313609061
bottom5:
n5526           -3.76686419472e-18
n7029           -3.11276105447e-18
n5316           -2.79708755182e-18
n5279           -2.29685151072e-18
n5553           -2.2920960841e-18
```

```
In [12]: def plot_nodes_partitions(df):

             data = []

             for partition_count in range(df['partition'].max()):

                 partition_df = df[df['partition'] == partition_count]
                 data.append(
                 go.Scatter3d(
                     x = partition_df['x'],
                     y = partition_df['y'],
                     z = partition_df['z'],
                     text =
                         partition_df['Residue']
                         +' '+partition_df['Dssp']
                         +' '+partition_df['Chain']
                         +' '+partition_df['rProtein'],
                     mode = 'markers',
                     name = 'partition'+str(partition_count)
```

```
                )
            )

        layout = go.Layout(
            title = 'Coloring SaCe rProteins by Community',
            showlegend = True
        )

        fig = go.Figure(data=data, layout=layout)
        iplot(fig)
```

```
In [13]: def plot_louvain(res, G, make_plot=True):
             partition = community.best_partition(G, resolution=res, weight='Energy')
             partition_df = pd.DataFrame.from_dict(partition, orient='index').reset_index()
             partition_df.rename(columns={0:'partition'}, inplace=True)
             ThTh_partition = ThTh_nodes.join(partition_df)
             ThTh_partition = ThTh_partition.drop(['index'], axis=1)
             print('Resolution:', res)
             print('Number of partitions:',len(set(partition.values())))
             print('Modularity:', community.modularity(partition, G))
             if make_plot == True:
                 plot_nodes_partitions(ThTh_partition)
             return(partition, ThTh_partition)
```

```
In [14]: louvain5, lv5_df = plot_louvain(5, G_ThTh)
```

```
Resolution: 5
Number of partitions: 47
Modularity: 0.7610006622112557
```

```
In [15]: louvain10, lv10_df = plot_louvain(10, G_ThTh, False)
```

```
Resolution: 10
Number of partitions: 109
Modularity: 0.7604881694865643
```

```
In [16]: lv10_df
```

Out[16]:

	NodeId	Chain	Position	Residue	Dssp	Degree	Bfactor_CA	Rapdf	\
0	B:1:_:U	B	1	U	NaN	2	-999.90	-999.900	
1	B:2:_:C	B	2	C	NaN	3	-999.90	-999.900	
2	B:3:_:C	B	3	C	NaN	6	-999.90	-999.900	
3	B:4:_:C	B	4	C	NaN	11	-999.90	-999.900	
4	B:5:_:C	B	5	C	NaN	16	-999.90	-999.900	
...	...	...	...	...	...	...	...	...	
7369	3:55:_:ARG	3	55	ARG	E	1	32.68	-16.082	
7370	3:56:_:VAL	3	56	VAL	E	1	53.08	-168.072	

```
7371  3:57:_:GLU     3        57     GLU    E       2       61.18  -67.546
7372  3:59:_:VAL     3        59     VAL    E       3       47.99   -0.117
7373  3:60:_:GLU     3        60     GLU            3       80.20   -2.371

          Tap  Accessibility rProtein           x           y           z  \
0    -999.900       -999.900       5S -140.216003  170.479996  158.654007
1    -999.900       -999.900       5S -143.912003  165.106995  157.557999
2    -999.900       -999.900       5S -144.324005  161.835999  152.091003
3    -999.900       -999.900       5S -143.281998  161.699997  145.934006
4    -999.900       -999.900       5S -139.960999  163.332001  140.940002
...       ...            ...      ...         ...         ...         ...
7369   -0.278          0.423     uL30  -97.103996  140.802002  196.343002
7370    0.394          0.339     uL30 -100.068001  143.119995  196.835007
7371   -0.069          0.408     uL30 -102.952003  143.319000  199.233994
7372    0.493          0.155     uL30 -105.612999  147.953003  202.595001
7373    0.000          0.834     uL30 -107.748001  150.294006  204.746994

      partition
0             0
1             0
2             0
3             0
4             0
...         ...
7369        108
7370          5
7371        108
7372        108
7373        108

[7374 rows x 15 columns]
```

In [17]:
```
normalized_mutual_info_score(list(louvain5.values()), list(louvain10.values()))
```

Out[17]:
```
0.84171647439049602
```

In [18]:
```
resolution = np.linspace(0.1, 10, num=49, endpoint=True, retstep=False, dtype=None)
```

In [19]:
```
def make_prtn_mod_res_df(resolution_list, G):
    modularity_list = []
    partition_list = []

    for res in resolution:
        partition = community.best_partition(G, resolution=res, weight='Energy')
        num_partitions = len(set(partition.values()))
        modularity = community.modularity(partition, G)
        modularity_list.append(modularity)
        partition_list.append(num_partitions)
```

```
    df = pd.DataFrame(
        {'Resolution':resolution_list,
         'Num_Partitions':partition_list,
         'Modularity':modularity_list})

    return(df)
```

0.0.2 This next cell takes a crazy long time to run, graph below may be an old version

In [25]:
```
prtn_mod_res_df = make_prtn_mod_res_df(resolution, G_ThTh)
```

```
# Create traces
trace0 = go.Scatter(
    x = prtn_mod_res_df['Resolution'],
    y = prtn_mod_res_df['Num_Partitions'],
    mode = 'lines',
    name = 'Partitions'
)
trace1 = go.Scatter(
    x = prtn_mod_res_df['Resolution'],
    y = prtn_mod_res_df['Modularity'],
    mode = 'lines',
    name = 'Modularity',
    yaxis='y2'
)


layout = go.Layout(
    title='Modularity and Parition Number vs. Louvain Resoution',
    xaxis=dict(
        title='Louvain Resolution'
    ),
    yaxis=dict(
        title='Number of Partitions'
    ),
    yaxis2=dict(
        title='Modularity',
        titlefont=dict(
            color='rgb(148, 103, 189)'
        ),
        tickfont=dict(
            color='rgb(148, 103, 189)'
        ),
        overlaying='y',
        side='right'
    )
)
```

```
data = [trace0, trace1]
fig = go.Figure(data=data, layout=layout)
iplot(fig)
```

0.0.3 Output makes no sense, all nodes are communities

```
In [20]: ig_G = ig.Graph.Read_GraphML('../Ring_output/1VY4/1VY4_LSU_rRNA+rProtein_network.xml')
```

```
In [21]: def walktrap_output(stps):
             walktrap = ig.Graph.community_walktrap(ig_G, weights='Energy', steps=stps)
             print('Steps:', stps)
             print('Optimal count:', walktrap.optimal_count)
             print('Modularity:', ig_G.modularity(membership=walktrap.as_clustering()))
             return([e for l in walktrap.merges for e in l])
```

```
In [22]: walktrap2 = walktrap_output(2)
```

```
Steps: 2
Optimal count: 5711
Modularity: 0.07320135301685493
```

```
In [23]: walktrap4 = walktrap_output(4)
```

```
Steps: 4
Optimal count: 5711
Modularity: 0.07320135301685493
```

```
In [24]: walktrap6 = walktrap_output(6)
```

```
Steps: 6
Optimal count: 5711
Modularity: 0.07320135301685493
```

```
In [25]: walktrap8 = walktrap_output(8)
```

```
Steps: 8
Optimal count: 5711
Modularity: 0.07320135301685493
```

```
In [26]: walktrap10 = walktrap_output(10)
```

```
Steps: 10
Optimal count: 5711
Modularity: 0.07320135301685493
```

```
In [27]: normalized_mutual_info_score(walktrap2, walktrap4)
```

```
Out[27]: 1.0
```