BIOINFORMATICS Class – PROJECT part 2 Group 9
Neuroscience application
Brain network study during resting states

Appendix

This appendix includes the figures which are explained in the report

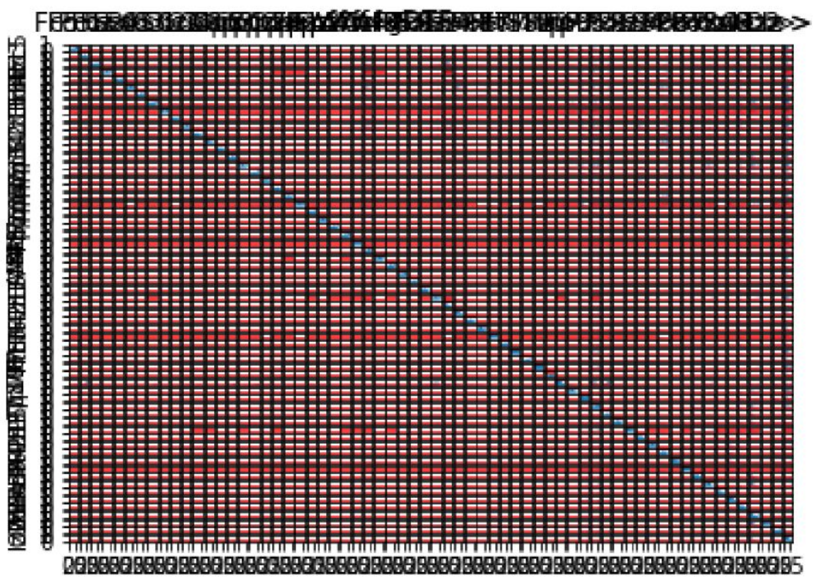

Figure1.1connDTF

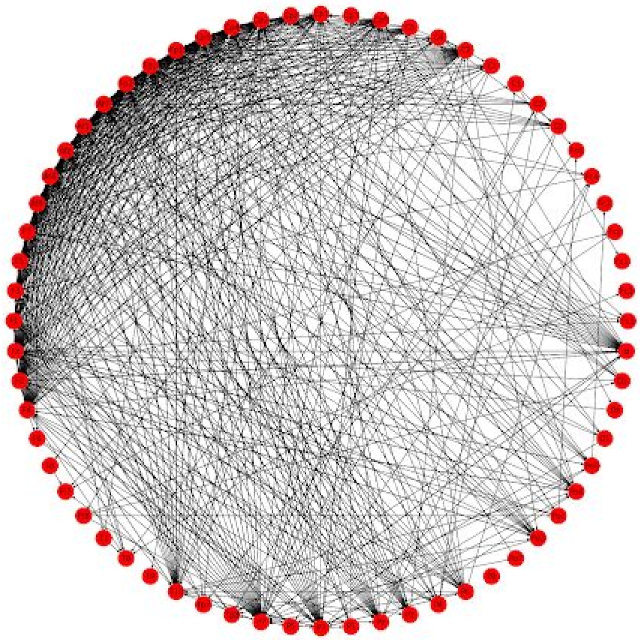

Figure1.1 GraphDTF

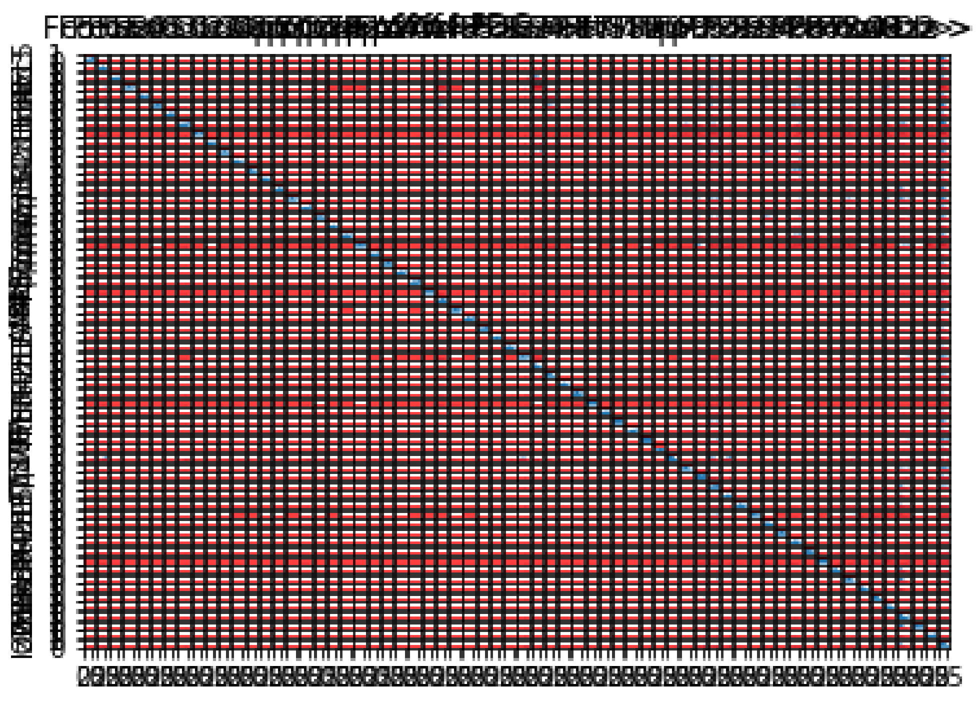

Figure1.2 connPDC

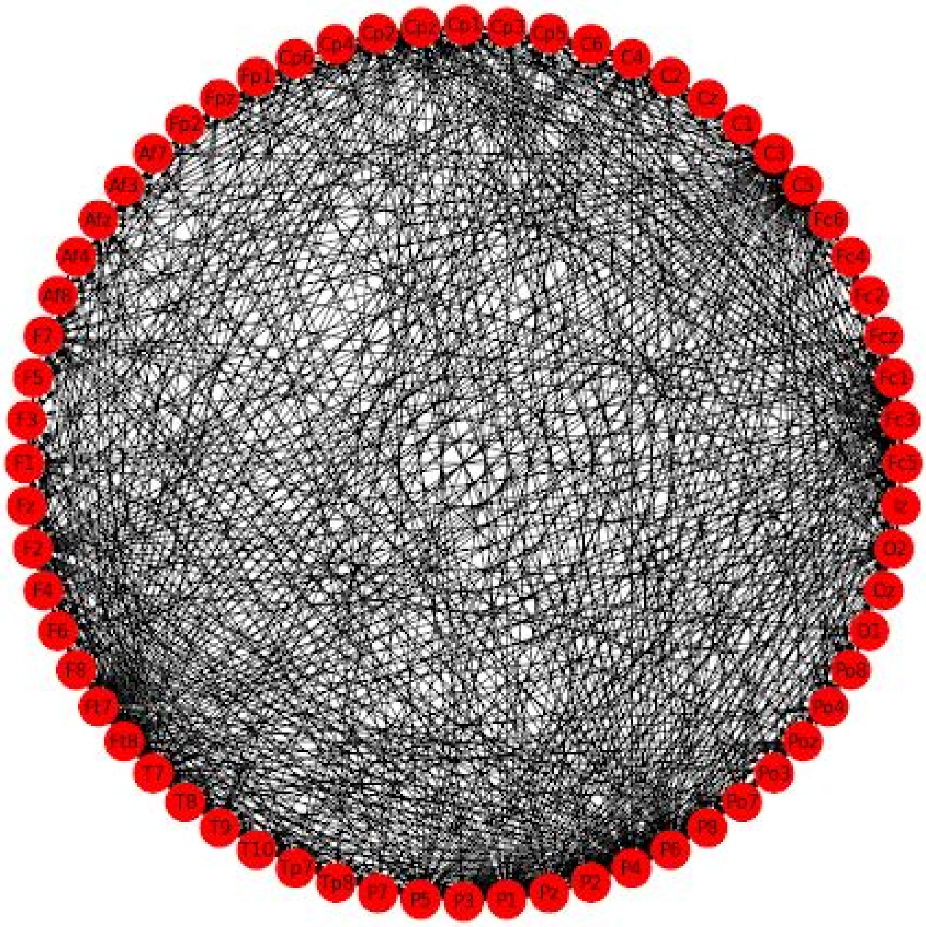

Figure1.2graphPDC

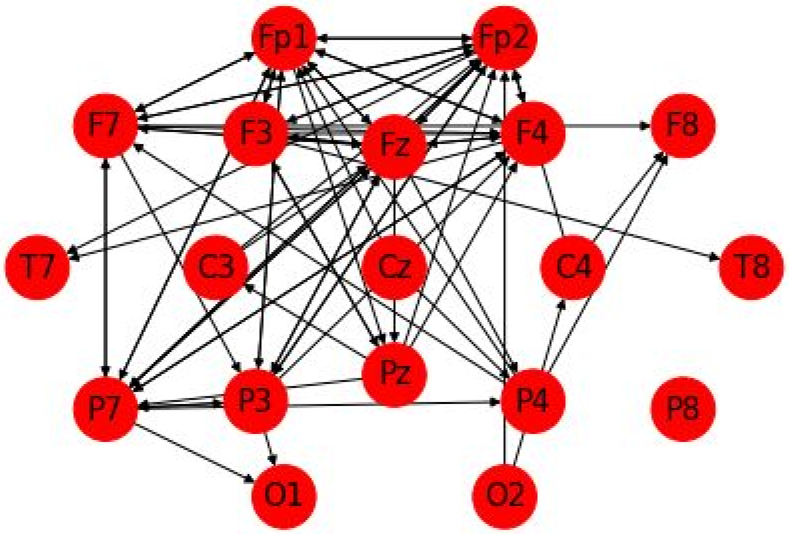

Figure1.5

```
[529]: nx.average_clustering(G_dtf.to_undirected())
[529]: 0.5549611432367636
```

Figure: 2.1.1 , Global Clustering Coefficients

```
[530]: nx.average_shortest_path_length(G_dtf)
[530]: 1.765625
```

Figure: 2.1.2 , Global Shortest Path Length

```
[116]: tmpList = []
       for n in G_1.nodes():
           tmpList.append([n, G_dtf.degree(n),G_dtf.in_degree(n),G_dtf.out_degree(n)])
       nodes_df = pd.DataFrame(tmpList, columns=['Node','Degree', 'In', 'Out'])
```

Figure: 2.1.3 , Code - Calculate nodes degrees

```
[118]:
```

	Node	Degree	In	Out
24	Af7	73	39	34
21	Fp1	67	36	31
22	Fpz	66	33	33
33	Fz	65	33	32
27	Af4	64	31	33
23	Fp2	61	32	29
26	Afz	58	28	30
20	Cp6	56	28	28
28	Af8	56	30	26
35	F4	55	24	31

Figure: 2.1.4 , top 10 nodes degrees

```
[502]: # the average degree:
        nodes_df['Degree'].mean()/len(nodes_df)

[502]: 0.39990234375

[506]: erdos = nx.erdos_renyi_graph(64,0.4, directed=True)

[507]: nx.average_clustering(erdos.to_undirected())

[507]: 0.6388776780989694

[508]: nx.average_shortest_path_length(erdos)

[508]: 1.6063988095238095
```

Figure: 2.2.1 , Small world network indices (using Erdos Renyi)

```
[153]: G_pdc
       nx.average_clustering(G_pdc.to_undirected())

[153]: 0.6437981099380977
```

Figure: 2.3.1 , Average Clustering Coefficient using PDC

```
[154]: nx.average_shortest_path_length(G_pdc)

[154]: 0.8467261904761905
```

Figure: 2.3.2 , Average Shortest Path length using PDC

```
[ ]: # find all the possible subgraphs in G_dtf
     from itertools import permutations
     subs = []
     for item in permutations(list(G_dtf.nodes), 3):
         subs.append(item)

[357]: len(subs)

[357]: 249984
```

Figure: 3.1.1 , Number of all 3-node combinations

▶ 13 different isomorphic types of 3-node connected subgraph

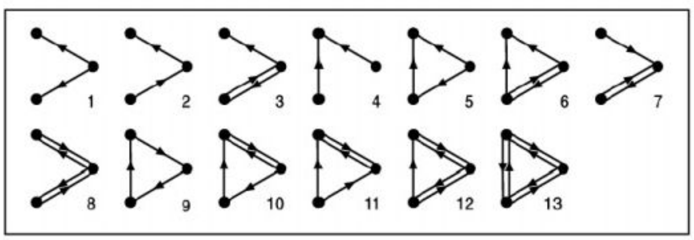

Figure: 3.1.2 , All 3-node motifs scenarios

```
[320]: mo_stats = [0]*13
        for s in subs:
            mo_stats = [sum(x) for x in zip(mo_stats, findMotif3(G_dtf, s))]

[321]: # the number of the found 3-nodes motifs in the origianl graph:
       mo_stats

[321]: [29950, 35526, 9000, 9398, 9398, 5391, 9398, 4966, 9000, 1566, 1566, 866, 435]
```

Figure: 3.1.3 , count of each motif foundings in the graph

```
[375]: # motif concentraion:
       G_concentration = [round(x/len(subs)*1000,2) for x in mo_stats]
       print(G_concentration)

       [119.81, 142.11, 36.0, 37.59, 37.59, 21.57, 37.59, 19.87, 36.0, 6.26, 6.26, 3.46, 1.74]
```

Figure: 3.1.4 , concentration of each motif foundings in the graph

```
[342]: # create 50 random graph with same number of nodes and degree distribution:

random_graphs = []
for i in range(50):
    random_graphs.append(nx.expected_degree_graph(degrees))

[ ]: motif3 = []
for g in random_graphs:
    print(len(motif3))
    motif3.append(findMotif3(g))
```

Figure: 3.1.5 , Code, create 50 random graphs with same degree, and then calculate their motifs foundings

```
[ ]: # checking the statistical representing of motifs:
motif_delta = []
for mo in R_concentration:
    motif_delta.append([(G_concentration[i] - mo[i])/(G_concentration[i] + mo[i]) for i in range(13)])


#motif_delta

[389]: motif_stats = []
for i in range(13):
    motif_stats.append(round(sum([x[i] for x in motif_delta])/len(motif_delta),2))

[390]: print(motif_stats)

[0.37, 0.19, 0.24, -0.16, -0.16, 0.28, -0.15, 1.0, 0.22, 1.0, 1.0, 1.0, 1.0]
```

Figure: 3.1.6 , Code, The stats of the motifs in the 50 random graphs

```
[423]: #
G_32 = nx.DiGraph()
G_32.add_nodes_from(channels)
for ed in G_dtf.edges:
    for n in G_32.nodes:
        if n != ed[0] and n != ed[1]:
            if ed[1] in list(G_dtf.neighbors(n)):
                G_32.add_edge(ed[0],ed[1])
                G_32.add_edge(n,ed[1])
```

Figure: 3.2.1 , Code, create a graph of the edges of the motif no.04

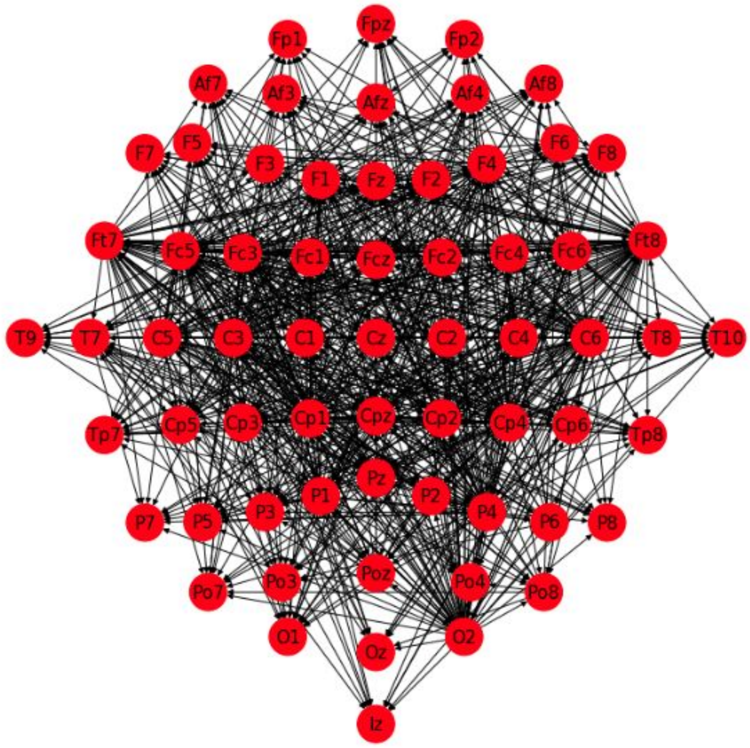

Figure: 3.2.2 , topographical representation of the networks considering only the connections involved in motif no.04