Project: LRH-1

Article: Ligand regulation of full-length LRH-1 associates with a novel metric of compound binding energy.

Analyst: David Foutch

Date: 06/26/2023

Update: 07/019/2023

1. PURPOSE OF THIS NOTEBOOK

To Demonstrate how network analysis was used to identify physical mechanisms in structural data associated with differences in protein regulation.

The following is a detailed description of how I used network analysis to identify mechanistic associations in the stability and regulation of 18 protein structures. However, *any* data that can be organized into a matrix form can benefit from the insights generated by a network, or systems, analysis.

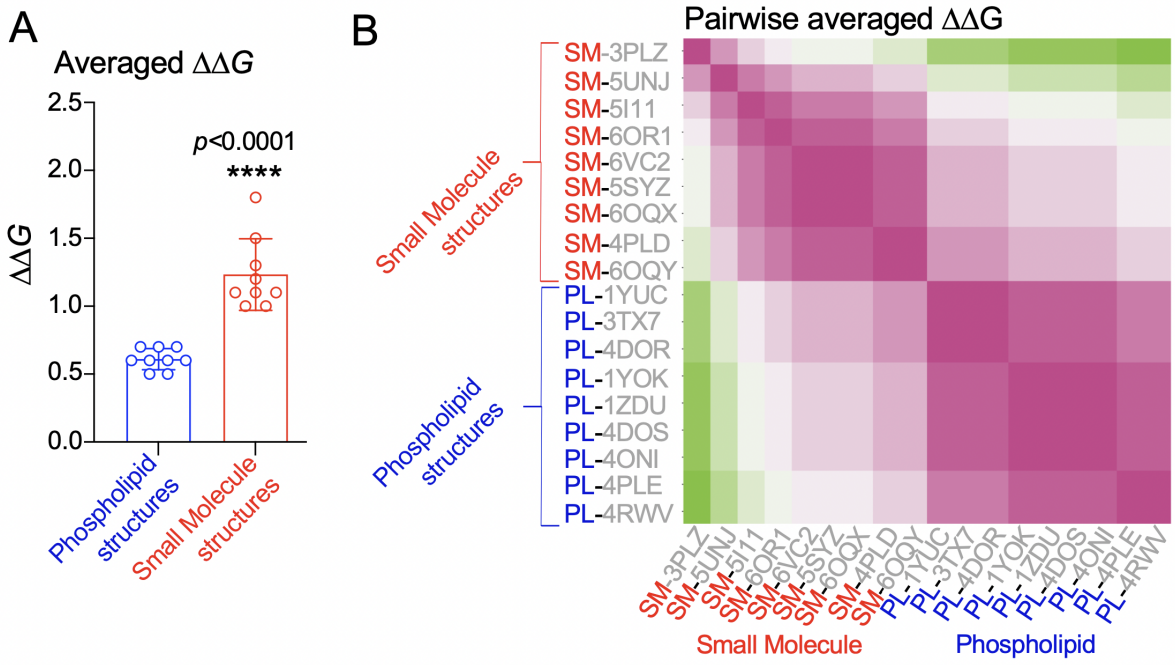
2. DESCRIBING THE PROBLEM:

Wet lab research identified a set of low $\Delta\Delta G$ and a set of high $\Delta\Delta G$ structures associated, respectively, with phospholipid and small molecule binding (see Fig 1A). This is a statistically significant, empirical observation.

My task was to use network analysis to identify, if possible, any network features that might offer an explanantion as to why these sets of proteins "behave" as they do.

NOTE: This work is in revision so details regarding this research may be omitted.

Figure 1

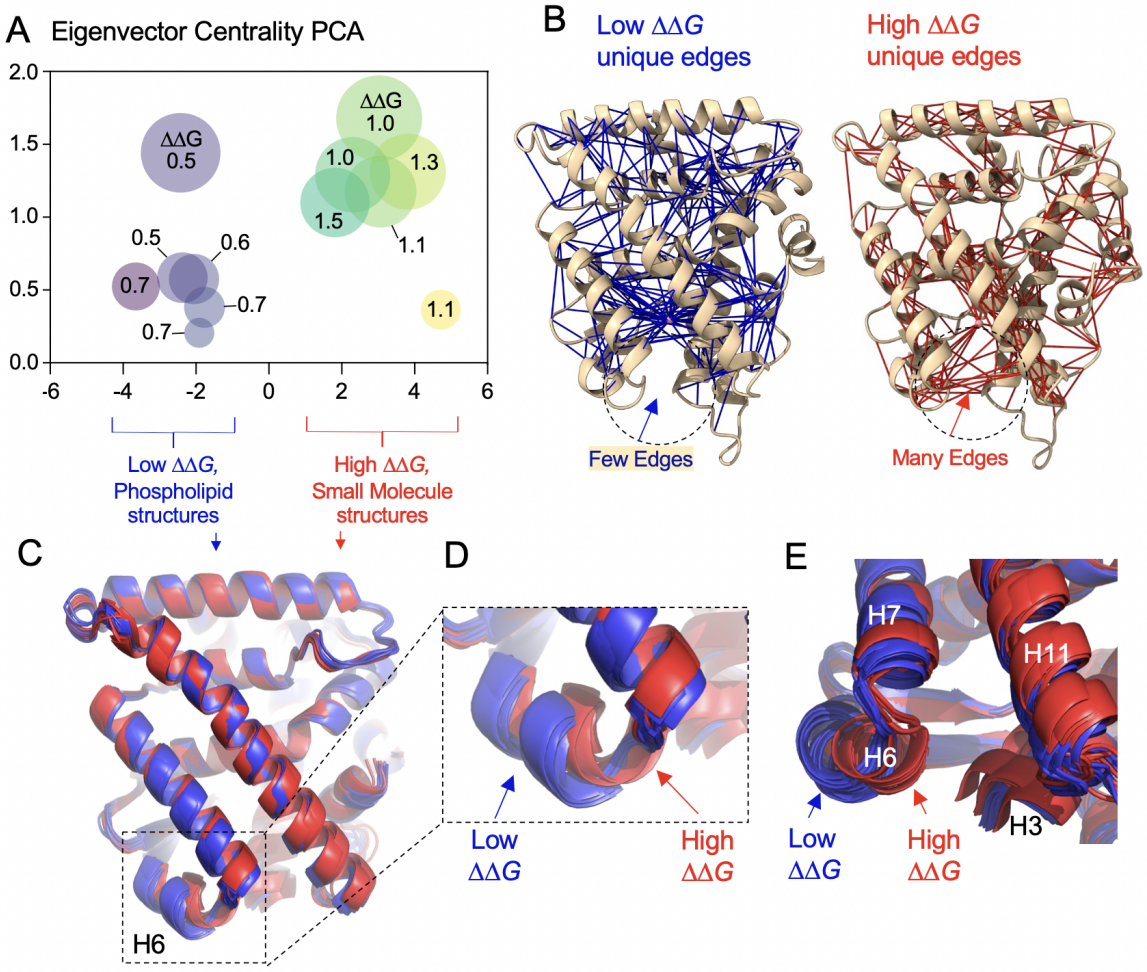


3. REVIEW OF THE RESULT:

There are two key results that offer insight into *why* these two groups of proteins may be sorted by their energetic equilibrium, or $\Delta\Delta G$ (see Fig 2). **First**, when the average eigenvector centrality of each secondary structure (helices, sheets, and loops) for each protein was used as features in the principal components analysis (PCA) the low $\Delta\Delta G$ and the high $\Delta\Delta G$ proteins clustereded nicely into two groups (see Fig 2A). What this indicates is that there is a set of secondary structures that are creating the variance between the equilibrium states. The bi-plot of the PCA indicated that helix 6 and helix 3 are significantfactors to understanding this variance between the two groups. **Second**, on inspection of the set of edges unique to the low $\Delta\Delta G$ PSNs and high $\Delta\Delta G$ PSNs. It is observed that there is a

greater degree of connectivity between helices 3 and 6 at the enzymatic site indicating that the opening is more restricted requiring greater energy to achieve small molecule binding to reach energetic equilibrium (see Fig 2B).

Figure 2



4. OVERVIEW OF THE PIPELINE:

1. The data consists of 18 X-ray crystallography files containing the molecular coordinates of all atoms for each protein structure. The protein of interest is the human liver receptor homolog-1 (LRH-1). The 18 LRH-1 structures are divided into two grous. One group contains small molecule binding to LRH-1. The other group contains

phospholipid binding to LRH-1. These 18 files were downloaded from the Protein Data Bank (PDB) (https://www.rcsb.org/) and used for analysis. The files are 1YOK, 1YUC, 1ZDU, 3PLZ, 3TX7, 4DOR, 4DOS, 4IS8, 4ONI, 4PLD, 4PLE, 4RWV, 5L11, 5SYZ, 6OQX, 6OQY, 6OR1, and 6VC2.

2. Each PDB file was submitted to the Residue Interaction Network Generator (RING) server (https://bio.tools/ring) which generates a link between amino acids based on a Euclidean, or distance, threshold measured in Ångstroms. The RING server generates a .pdb_ringnodes file which is a record of residue interactions for each protein structure. These files were used for creating adjacency lists for network visualization and analysis.

3. The Networkx (https://networkx.org/) Python package creates a graph object from the adjacency list generated by the RING server. This graph object is used by Networkx has a wide-range of methods to perform network analysis. In this work Networkx is used to evaluate the eigenvector centrality (EC) of each residue in the LRH-1 PSNs.

4. A number of straightforward anaylyses are performed in Python (https://www.python.org/) in order to understand the distribution of EC across the topology of the network. The PCA clusters protein structures and visualizes similarity in variation. The line graphs, Venn diagrams, and scatterplots were used to explore potential associations in variability.

5. Finally, ChimeraX (https://www.cgl.ucsf.edu/chimerax/) is used to visualize how the edges generated by the RING server are distributed throughout the 3D structure of the protein. This is useful in demonstrating how the EC variance in the PSNs is related to the physical features of the protein structure. This also aids in conceptualizing how these physical features may explain changes in protein function.

```
<h1>Figure 3.</h1>
<div>
<img src="attachment:pipeline2.png"/>
</div>
```

5. OVERVIEW OF THE DATA – PDB FILE ENTRIES:

PDB files are entensive records. It is beyond the scope of this project to go into all the details of a standard PDB file format. However, the following review of the fields of a typical PDB file will aid in following the progressive logic of each step of the analysis.

Table 1. A set of PDB file columns and their field values:

In [79]:
```
df=pd.read_csv('/Users/davidfoutch/Desktop/David/full_length/pdbDescripti
df.style.set_table_attributes('style="font-size: 16px"')
```

Out[79]:

	Column names	Description
0	Record:	Identifies the record type 'ATOM'
1	Count:	The serial number for each atom in the amino acid (AA) sequence
2	Atom id:	Atom identifier
3	Amino Acid:	Three letter AA code
4	Chain:	Identifier for a linear polymer terminated by an amine and carboxyl group
5	Position:	Residue sequence number
6	x-coord:	The X coordinate measured in Ångstroms
7	y-coord:	The Y coordinate measured in Ångstroms
8	z-coord:	The Z coordinate measured in Ångstroms
9	Occupancy:	1.0
10	B-factor:	Temperature (thermal shift)
11	Atom:	Single letter atom code

Table 2. Typical PDB file layout for ATOM entries:

In [18]:
```
df=pd.read_csv('/Users/davidfoutch/Desktop/David/1yok/1yok_pdb.txt',sep='
df.iloc[0:11].style.set_table_attributes('style="font-size: 16px"')
```

Out[18]:

	Record	Count	Atom id	Amino acid	Chain	Position	x-coord	y-coord
0	ATOM	1	N	SER	A	300	28.985000	54.267000
1	ATOM	2	CA	SER	A	300	27.739000	54.655000
2	ATOM	3	C	SER	A	300	27.105000	53.481000
3	ATOM	4	O	SER	A	300	27.492000	53.158000
4	ATOM	5	CB	SER	A	300	28.029000	55.799000
5	ATOM	6	OG	SER	A	300	26.852000	56.199000
6	ATOM	7	HN1	SER	A	300	29.407000	55.049000
7	ATOM	8	HN2	SER	A	300	29.656000	53.832000
8	ATOM	9	HN3	SER	A	300	28.816000	53.480000
9	ATOM	10	HG	SER	A	300	27.032000	56.908000
10	ATOM	11	N	ILE	A	301	26.118000	52.858000

6. GENERATING PSNS FROM PDB FILES USING THE RING SERVER:

Figure 4.

Figure 4. Each Protein Data Bank (PDB) (https://www.rcsb.org/) file was submitted to the Residue Interaction Network Generator (RING) server (https://bio.tools/ring) which generates a link between amino acids based on a Euclidean, or distance, threshold measured in Ångstroms. The RING server generates a .pdb_ringnodes file which is a record of residue interactions for each protein structure. These files were used for creating adjacency lists for network visualization and analysis.

Preliminary step: Importing Python packages

In [75]:
```
# Import necessary packages to perform the analysis demonstrated below.

import pandas as pd
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from matplotlib_venn import venn2,venn2_circles
from matplotlib import pyplot as plt
import csv
from plotly.subplots import make_subplots
import plotly.graph_objects as go
```

The RING server

Using the RING server is straightforward. The file of interest can be directly loaded from the PDB website or it can be uploaded from a local file. The RING server will generate a file similar to the _ringnodes file shown below. Of primary interest are columns "NodeId1" and "NodeId2". After some text editing, these two columns were used to create an adjacency or edge list (below).

Table 3. RING server _ringnodes output file

In [20]:
```
df=pd.read_csv('/Users/davidfoutch/Desktop/David/1yok/1yok_ringnodes.txt'
df.style.set_table_attributes('style="font-size: 16px"')
```

Out[20]:

	NodeId1	Interaction	NodeId2	Distance	Energy	Atom1	A
0	A:300:_:SER	VDW:MC_SC	A:488:_:ASN	3.976000	6.000000	C	
1	A:301:_:ILE	VDW:SC_SC	A:305:_:ILE	3.655000	6.000000	CG2	
2	A:301:_:ILE	VDW:SC_SC	A:306:_:LEU	3.774000	6.000000	CB	
3	A:301:_:ILE	VDW:SC_SC	A:446:_:ARG	3.918000	6.000000	CD1	
4	A:302:_:PRO	VDW:SC_SC	A:305:_:ILE	4.186000	6.000000	CB	
5	A:302:_:PRO	VDW:SC_SC	A:484:_:TYR	3.588000	6.000000	CD	
6	A:304:_:LEU	VDW:MC_SC	A:307:_:GLU	4.153000	6.000000	C	
7	A:304:_:LEU	VDW:MC_SC	A:308:_:LEU	4.021000	6.000000	C	
8	A:304:_:LEU	VDW:SC_SC	A:365:_:PHE	3.644000	6.000000	CD1	
9	A:304:_:LEU	VDW:SC_SC	A:476:_:GLN	3.651000	6.000000	CD1	
10	A:304:_:LEU	VDW:SC_SC	A:477:_:VAL	3.927000	6.000000	CD2	
11	A:304:_:LEU	VDW:SC_SC	A:480:_:ALA	4.146000	6.000000	CB	

Table 4. An example of an edge list after processing the _ringnodes file:

In [28]:
```
df=pd.read_csv('/Users/davidfoutch/Desktop/David/1yok/1yok_edgeList.txt',
df.columns=['Source node','Target node']
df.iloc[0:12].style.set_table_attributes('style="font-size: 16px"')
```

Out[28]:

	Source node	Target node
0	300:SER	488:ASN
1	301:ILE	305:ILE
2	301:ILE	306:LEU
3	301:ILE	446:ARG
4	302:PRO	305:ILE
5	302:PRO	484:TYR
6	304:LEU	307:GLU
7	304:LEU	308:LEU
8	304:LEU	365:PHE
9	304:LEU	476:GLN
10	304:LEU	477:VAL
11	304:LEU	480:ALA

7. GENERATING EIGENVECTOR CENTRALITY VALUES

FOR EACH PSN USING PYTHON'S NETWORKX:

Figure 6.

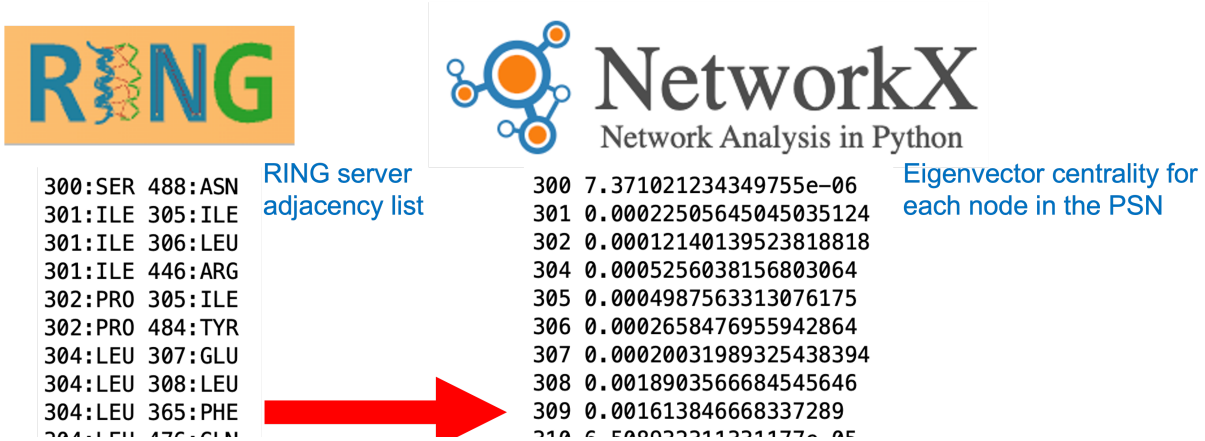


Figure 6. The Networkx (https://networkx.org/) Python package creates a graph object from the adjacency list generated by the RING server. This graph object is used by Networkx has a wide-range of methods to perform network analysis. In this work Networkx is used to evaluate the eigenvector centrality (EC) of each residue in the LRH-1 PSNs.

Table 5. Generating EC multiplots:

In [29]:
```
fig, axs = plt.subplots(5,4,figsize=(30,20),sharey=True, sharex=True)
pdbList=[['1yok','1yuc','1zdu','3plz'],['3tx7','4dor','4dos','4oni'],['4p
         ['4pld','5l11','5syz','5unj'],['6oqx','6oqy','6or1','NA']]

path = "/Users/davidfoutch/Desktop/David/"

for j in range(5):
    for k in range(4):
        X = []
        Y = []
        if pdbList[j][k] != 'NA':
            i=pdbList[j][k]
            pathToFile=path+i+"/eigen_"+i+"_plt"
            with open(pathToFile, 'r') as file:
                plotting = csv.reader(file, delimiter='\t')
                for ROWS in plotting:
                    X.append(int(ROWS[0]))
                    Y.append(float(ROWS[1]))

                axs[j,k].plot(X, Y, color="black", label=i)
                axs[j,k].legend(fontsize="12")


# plt.xticks(np.arange(210, 485, 25))

# # plt.title('')
fig.text(0.075, 0.5, 'Eigenvector Centrality', va='center', rotation='ver
fig.text(0.5, 0.055, 'Residue Position', ha='center',fontsize=18)

# plt.savefig('nr5a1_plts.png')
```

Out[29]: Text(0.5, 0.055, 'Residue Position')

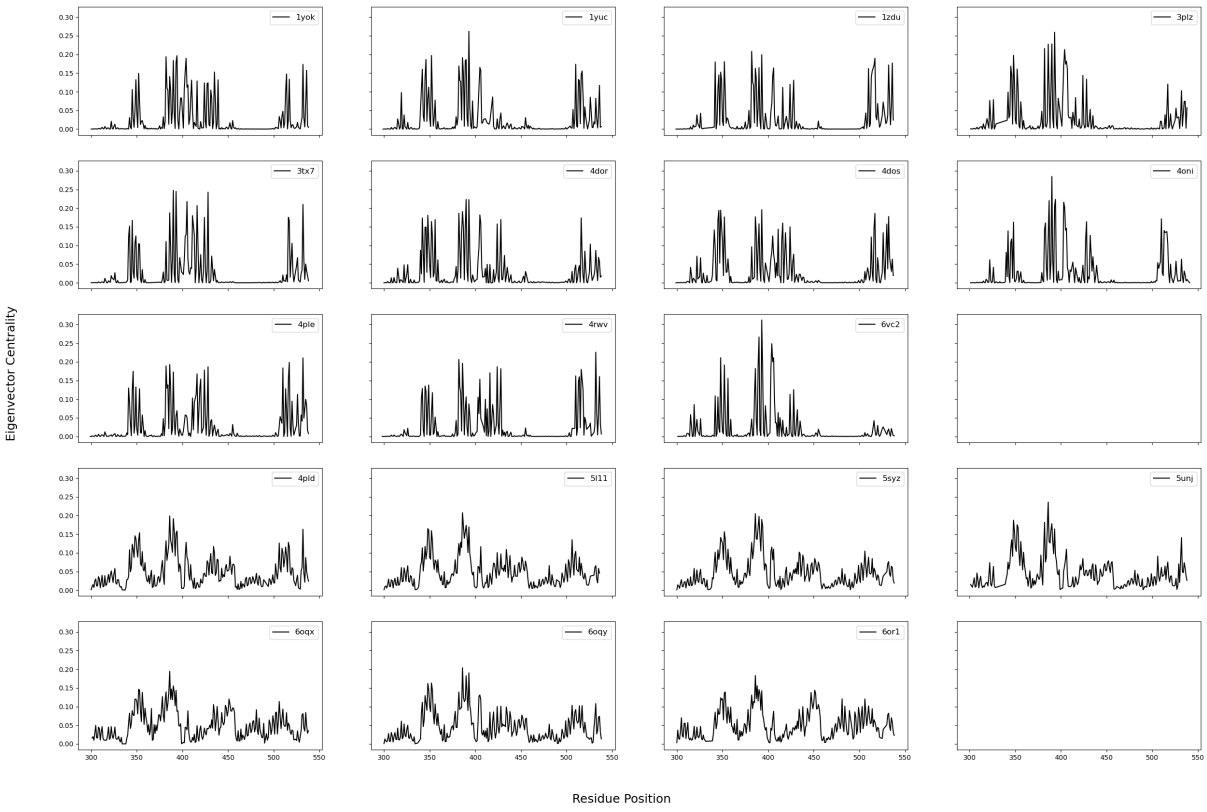


Table 5. The x-axis plots the physical sequence (or, primary structure) of each residue of the protein. The y-axis plots the eigenvector centrality value for each residue in the sequence. On visual inspection it seems reasonable that these these protein systems may be clustered into two groups. It also seems reasonable that principal components analysis PCA)is a reasonable approach for investigating what features are driving this variance.

8. PERFORMING PCA USING SECONDARY STRUCTURES:

Figure 8.

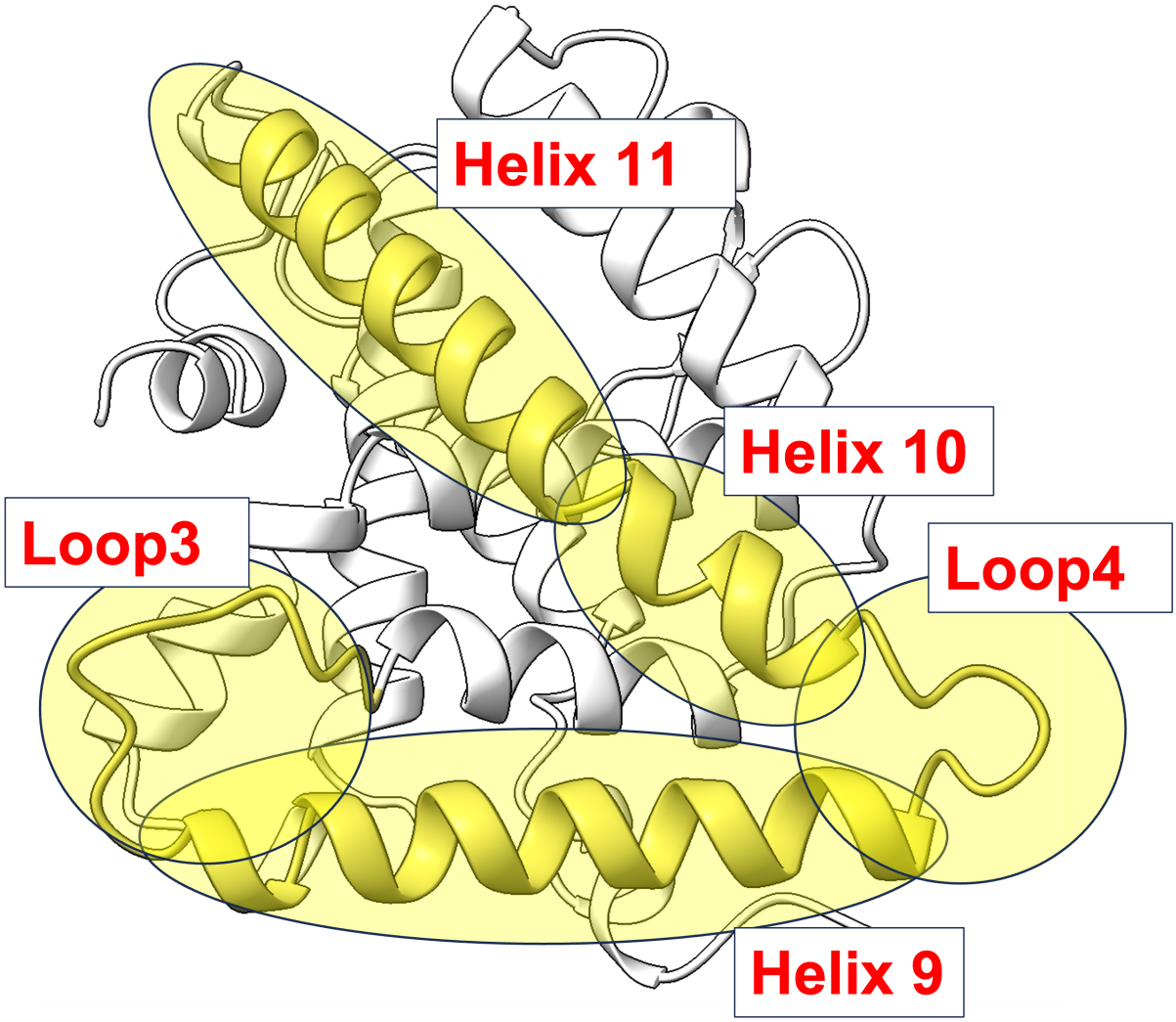


Figure 8. Targets and features. The LRH-1 protein is a sequence of 242 residues. To purpose of this PCA is to find some measure of variation in the eigenvector values that might reproduce the visual grouping of the 18 structures. The approach we adopted was to identify the primary structure (or, sequence) associated with each of the secondary structures and average the eigenvector values for each residue within each secondary structure. This figure illustrates how each secondary structure (loops, helices, and sheets) was treated as a group of residues. Here the protein structures associated with each PDB file are the targets and the secondary structures are the features (See table 5 below).

Table 6. The targets (rows) and features (columns) used in the PCA

In [53]:
```
#Load the table as a Pandas dataframe.
"""Note: That the 'pdb_file' entries contain the list of 'targets' used i
   The column names are the list of secondary structures of LRH-1. Each (
   for the secondary structure associated with the PDB file of that row."

df=pd.read_csv('/Users/davidfoutch/Desktop/David/secondary_structure_pca.
df.style.set_table_attributes('style="font-size: 16px"')
```

Out[53]:

	pdb_file	helix1	helix2	loop1	helix3	helix4	helix5
0	1yok	0.000576	0.003654	0.000314	0.031029	0.001018	0.052709
1	1yuc	0.000743	0.012840	0.000890	0.057354	0.001669	0.060717
2	1zdu	0.000202	0.007947	0.000460	0.050669	0.002127	0.050298
3	3plz	0.001271	0.015656	0.000000	0.055139	0.004365	0.054102
4	3tx7	0.000625	0.005513	0.000427	0.044520	0.000000	0.043169
5	4dor	0.002067	0.010955	0.001888	0.067321	0.004443	0.058594
6	4dos	0.000402	0.028617	0.005939	0.067000	0.000911	0.038745
7	4oni	0.000312	0.008313	0.000734	0.033683	0.000433	0.057376
8	4pld	0.019235	0.030206	0.007326	0.081821	0.029528	0.085718
9	4ple	0.001660	0.002323	0.001876	0.045064	0.001737	0.046692
10	4rwv	0.000914	0.003979	0.000283	0.043673	0.001150	0.043446
11	5l11	0.015617	0.033968	0.012121	0.087246	0.020994	0.090050
12	5syz	0.014761	0.029265	0.010711	0.085142	0.024236	0.094928
13	5unj	0.018808	0.020364	0.001765	0.096289	0.025358	0.088195
14	6oqx	0.026081	0.020759	0.005134	0.083458	0.041482	0.096881
15	6oqy	0.013663	0.026514	0.004330	0.087202	0.020875	0.086448
16	6or1	0.032763	0.021224	0.003412	0.079031	0.029667	0.083683
17	6vc2	0.000540	0.015342	0.000711	0.044759	0.001578	0.053363

Table 7. Inspect PC1 and PC2

In [54]:
```
df=pd.read_csv('/Users/davidfoutch/Desktop/David/transpose_for_pca.txt',s
df=df.drop(['seq_pos','seq_id','sec_struct'], axis=1)


scaler = StandardScaler()
scaler.fit(df)
df_scaled = scaler.transform(df)

pca = PCA(n_components=2)
PC_scores = pd.DataFrame(pca.fit_transform(df_scaled),
               columns = ['PC 1', 'PC 2'])
loadings = pd.DataFrame(pca.components_.T, columns=['PC1', 'PC2'],
                        index=list(df.columns.values))
loadings.style.set_table_attributes('style="font-size: 16px"')
```

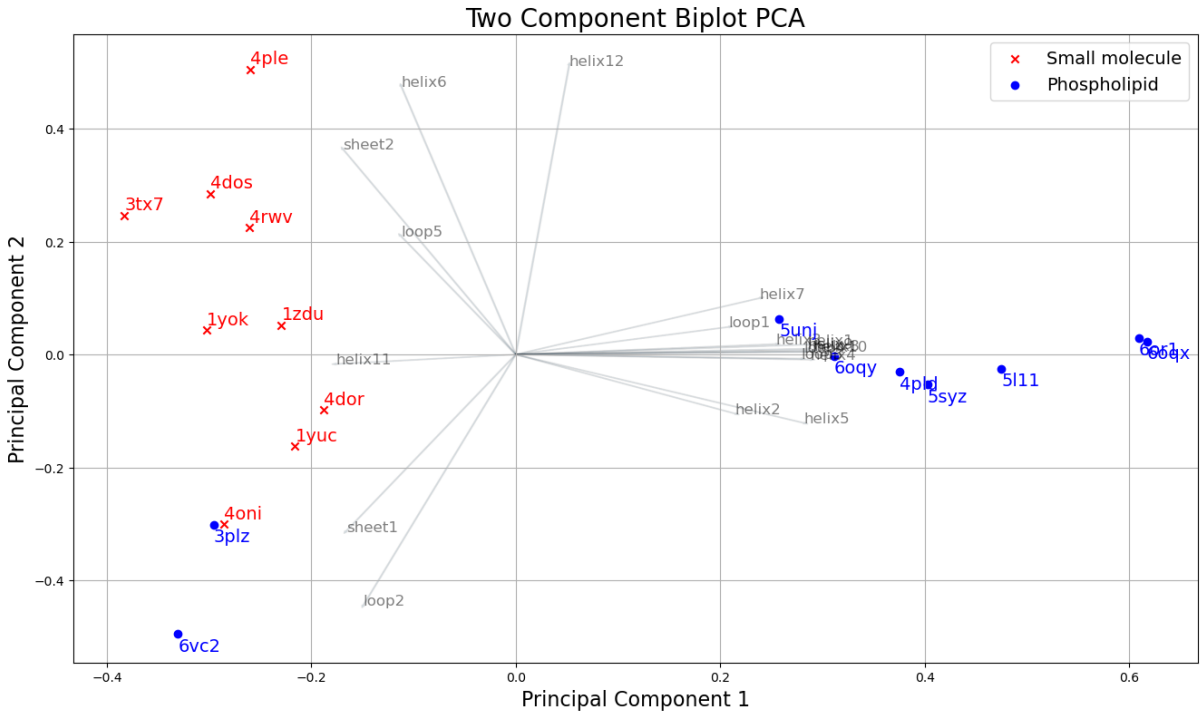
Out[54]:

	PC1	PC2
1yok	0.230914	-0.181243
1yuc	0.250479	-0.148688
1zdu	0.252150	-0.204559
3plz	0.250873	-0.165109
3tx7	0.228274	-0.262484
4dor	0.262657	-0.129846
4dos	0.024133	0.314485
4oni	0.234650	-0.186726
4pld	0.260003	0.169378
4ple	0.218371	-0.232409
4rwv	0.231815	-0.252638
5l11	0.253263	0.244395
5syz	0.258122	0.243177
5unj	0.254570	0.215808
6oqx	0.217383	0.383928
6oqy	0.273277	0.157543
6or1	0.208507	0.402805
6vc2	0.225098	-0.099870

Figure 9. Visualizing the PCA two component biplot

In [52]:
```
# The 'targets' are the protein structures represented by the four charac
# and the 'features' are the secondary structures (helices, strands, shee

targets = ['1yok','1yuc','1zdu','3plz','3tx7','4dor','4dos','4oni','4pld'
    '5l11','5syz','5unj','6oqx','6oqy','6or1','6vc2']

features = ['helix1', 'helix2', 'loop1', 'helix3', 'helix4', 'helix5', 's
       'loop2', 'sheet2', 'helix6', 'helix7', 'helix8', 'loop3', 'helix9'
       'loop4', 'helix10', 'helix11', 'loop5', 'helix12']

df=pd.read_csv('/Users/davidfoutch/Desktop/David/secondary_structure_pca.
df2=df.drop(columns=['pdb_file'])

x = df2.loc[:,features].values
y = df['pdb_file']
y = y.to_frame()


x = StandardScaler().fit_transform(x)

pca = PCA(n_components=2)

principalComponents = pca.fit_transform(x)

principalDF = pd.DataFrame(data = principalComponents,
                           columns = ['PC 1', 'PC 2'])

finalDF = pd.concat([principalDF, y], axis=1)

scalePC1 = 1.0/(finalDF['PC 1'].max() - finalDF['PC 1'].min())
scalePC2 = 1.0/(finalDF['PC 2'].max() - finalDF['PC 2'].min())


ldngs = pca.components_
featuresList = list(df2.columns.values)

fig, ax = plt.subplots(1,1,figsize=(16,9))

# ax = fig.add_subplot(1,1,1)

ldngs = pca.components_
featuresList = list(df2.columns.values)

for i, feature in enumerate(featuresList):
    ax.arrow(0, 0, ldngs[0, i],
             ldngs[1, i], alpha=0.1)
    ax.text(ldngs[0, i] * 1.0,
            ldngs[1, i] * 1.0,
            feature, fontsize=12, alpha=0.5)

ax.set_xlabel('Principal Component 1', fontsize = 16)
ax.set_ylabel('Principal Component 2', fontsize = 16)
ax.set_title('Two Component Biplot PCA', fontsize = 20)



set9 = finalDF[finalDF['pdb_file'].isin(['1yok','1yuc','1zdu','3tx7','4do
```

```
set10 = finalDF[finalDF['pdb_file'].isin(['3plz','4pld','5l11','5syz','5u



ax.scatter(set9['PC 1']*scalePC1, set9['PC 2']*scalePC2,color='red',marke
ax.scatter(set10['PC 1']*scalePC1, set10['PC 2']*scalePC2,color='blue',la

ax.legend(loc='upper right',fancybox=True,fontsize=14)
ax.grid()

for i in range(0,9):
    plt.text(set9.iloc[i,0]*scalePC1, set9.iloc[i,1]*scalePC2+0.01, set9.

for i in range(9):
    plt.text(set10.iloc[i,0]*scalePC1, set10.iloc[i,1]*scalePC2-0.03, set

# plt.savefig('/Users/davidfoutch/Desktop/David/new_two_component_biplot_
```



9. SET OF EDGES UNIQUE TO THE SMALL MOLECULE AND LIPID GROUPS:

Among some of the final questions we asked were "What edges are unique to each of the clusters?" and "What, if any, biological significance would these edges have?" To answer these questions we performed the follwoing steps:

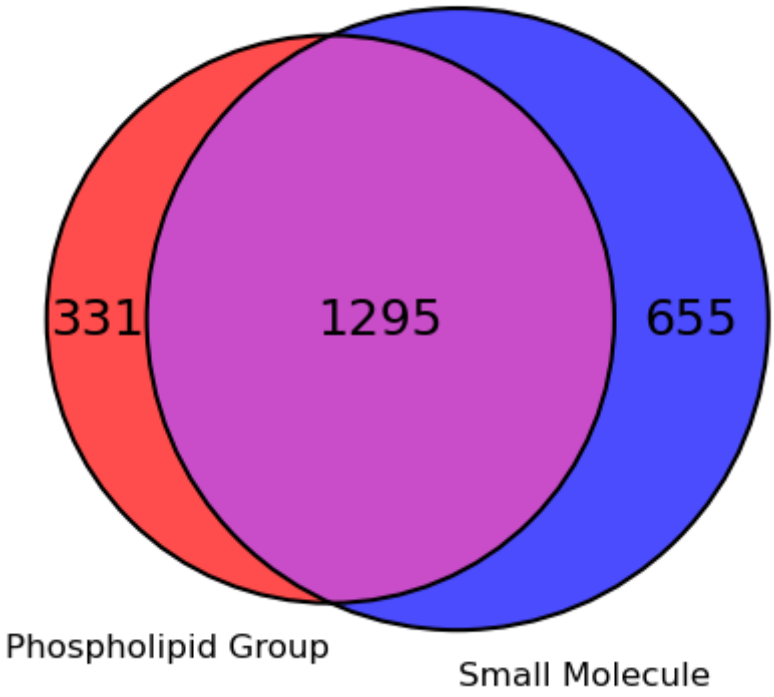
1. A complete set of unique edges for all small-molecule PSNs and a complete set of unique edges for all lipid PSNs were collected into separate lists.

2. These sets of uniques edges were compared and all edges that were shared in both lists were removed so that only those edges that were strictly unique to each group remained (See Table 8 below).

3. These edges were mapped back to diagrams of the 3D protein structure and these were investigated for biological significance.

Table 8. Venn diagram of the size of unique edges per group and the intersection

In [56]:
```
out=venn2(subsets = (331, 655, 1295), set_labels = ('Phospholipid Group',
venn2_circles(subsets=(331,655,1295), linewidth=1.5, color='black')

for text in out.subset_labels:
    if text == 0:
        break
    else:
        text.set_fontsize(18)
```



10. PROTEIN STRUCTURE NETWORK REPRESENTATIONS:

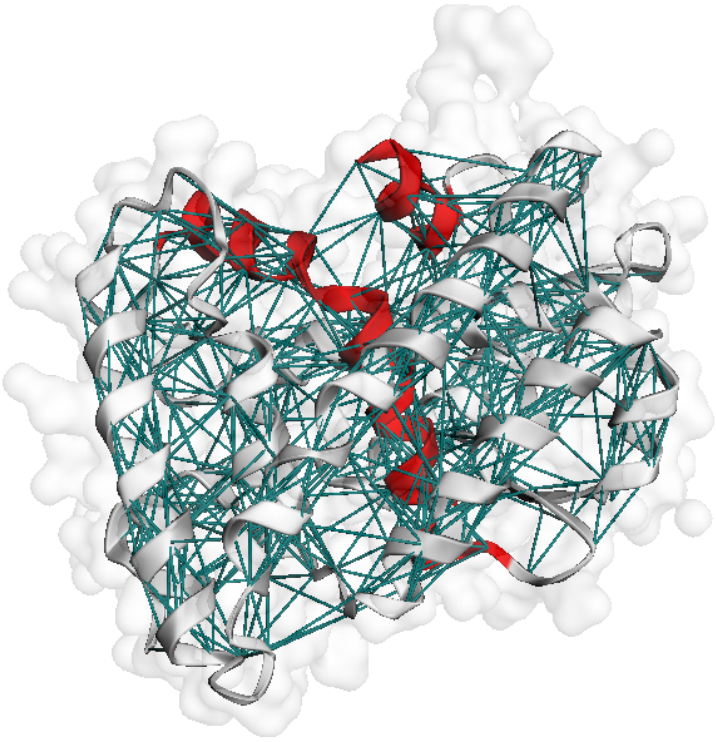
The Last three figures show different ways on visualizing PSNs. The first two figures show PSNs mapped to the 3D structure of LRH-1 using py3Dmol (https://pypi.org/project/py3Dmol/). The last figure is a PSN using the more familiar 2D network representation. This network was created using D3.js (https://d3js.org/)

1. This first 3D structure below shows the set of unique edges for all small molecule bound PSNs. The helices that are colored red was used to examine the role of helices 6,12.

2. This second 3D structure below shows the set of unique edges for all phospholipid bound PSNs. Again, the helices that are colored red was used to examine the role of helices 6,12.

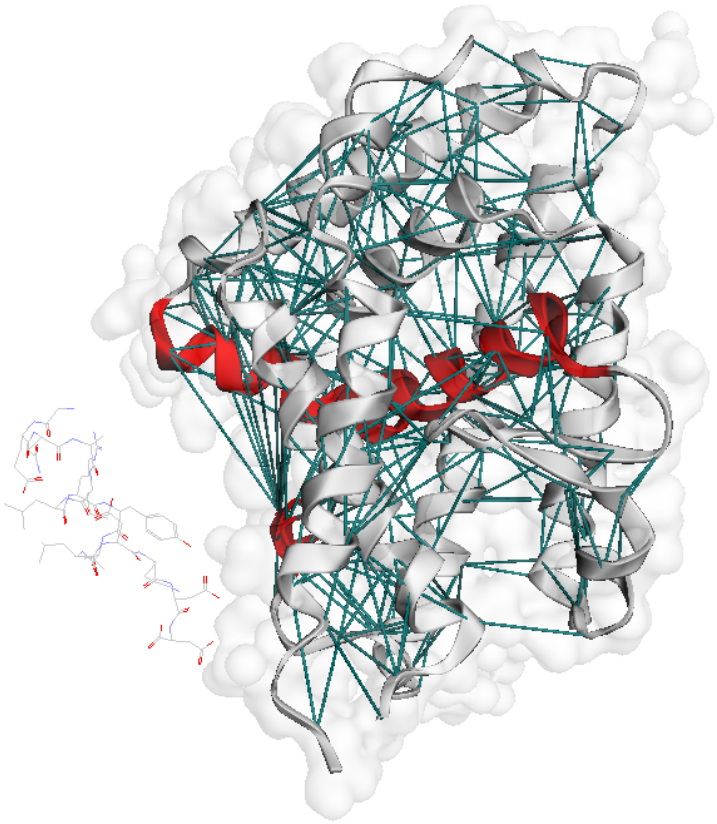
3. The third and last PSN was used simply to demonstrate an alternative means of presenting the same information.

In [76]:
```
import pandas as pd
import py3Dmol

df = pd.read_csv("/Users/davidfoutch/Desktop/4pld.txt", sep="\t")

view = py3Dmol.view(query='pdb:4PLD')
chA = {'chain':'A'}
chB = {'chain':'B'}
view.addSurface(py3Dmol.VDW,{'opacity':0.4,'color':'white'}, chA)
view.setStyle(chA,{'cartoon': {'color':'white'}})
view.setStyle(chB,{'': {'color':None}})
view.setHoverable({},True,'''function(atom,viewer,event,container) {
                    if(!atom.label) {
                     atom.label = viewer.addLabel(atom.resn+":"+atom.atom,
                    }}''',
                '''function(atom,viewer) {
                    if(atom.label) {
                     viewer.removeLabel(atom.label);
                     delete atom.label;
                    }
                 }''')



resset = [370,371,372,373,374,375,376,377,378,379,380,381,382,383,384,385
          390,391,392,393,394,395,396,397,398,530,531,532,533,534,535,536
for i in resset:
    view.addStyle({'chain':'A','resi': i},{'cartoon':{'color':'red'}})

for i in range(723):

    view.addCylinder(
        {'start':dict(x=df.iloc[i][2],y=df.iloc[i][3],z=df.iloc[i][4]),
        'end':dict(x=df.iloc[i][5],y=df.iloc[i][6],z=df.iloc[i][7]),
        'radius':0.12,
        'fromCap':1,
        'toCap':1,
        'color':'teal',
        'dashes':False
        }
    )

view.render()
```



Out[76]: <py3Dmol.view at 0x7fc0f4a4e590>

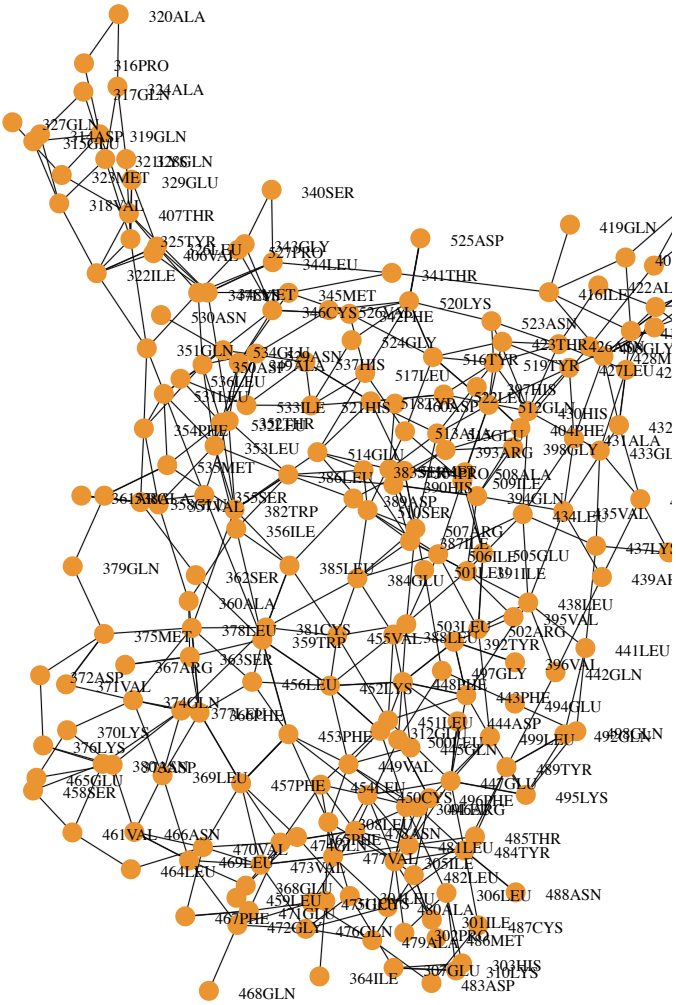
In [77]:
```
import pandas as pd
import py3Dmol

df = pd.read_csv("/Users/davidfoutch/Desktop/1yok.txt",sep="\t")

view = py3Dmol.view(query='pdb:1YOK')
chA = {'chain':'A'}
# chB = {'chain':'B'}
view.addSurface(py3Dmol.VDW,{'opacity':0.4,'color':'white'}, chA)
view.setStyle(chA,{'cartoon': {'color':'white'}})
view.setStyle(chB,{'': {'color':None}})
view.setHoverable({},True,'''function(atom,viewer,event,container) {
                   if(!atom.label) {
                    atom.label = viewer.addLabel(atom.resn+":"+atom.atom,
                   }}''',
                '''function(atom,viewer) {
                   if(atom.label) {
                    viewer.removeLabel(atom.label);
                    delete atom.label;
                   }
                 }''')



resset = [370,371,372,373,374,375,376,377,378,379,380,381,382,383,384,385
          390,391,392,393,394,395,396,397,398,530,531,532,533,534,535,536
for i in resset:
    view.addStyle({'chain':'A','resi': i},{'cartoon':{'color':'red'}})

for i in range(373):

    view.addCylinder(
        {'start':dict(x=df.iloc[i][2],y=df.iloc[i][3],z=df.iloc[i][4]),
        'end':dict(x=df.iloc[i][5],y=df.iloc[i][6],z=df.iloc[i][7]),
        'radius':0.12,
        'fromCap':1,
        'toCap':1,
        'color':'teal',
        'dashes':False
        }
    )

for i in range(373):

    view.addCylinder(
        {'start':dict(x=df.iloc[i][2],y=df.iloc[i][3],z=df.iloc[i][4]),
        'end':dict(x=df.iloc[i][5],y=df.iloc[i][6],z=df.iloc[i][7]),
        'radius':0.12,
        'fromCap':1,
        'toCap':1,
        'color':'teal',
        'dashes':False
        }
    )

view.render()
```



Out[77]: <py3Dmol.view at 0x7fc0f1e68df0>

In [78]:
```
from IPython.display import IFrame

IFrame(src='./d3graph2.html', width=900, height=600)
```

Out[78]:



In [ ]:
```

```