Design Rules for Sequestration of Viruses into Polypeptide Complex Coacervates

Supplementary Information

Python Scripts

Design Rules for Sequestration of Viruses Into Polypeptide-based Complex Coacervates

Pratik U. Joshi[1,2], Claire Decker[1], Xianci Zeng[3], Arvind Sathyavageeswaran[3,], Sarah L. Perry[3,4]*, Caryn L. Heldt[1,2,]*

[1]Department of Chemical Engineering, Michigan Technological University, Houghton, MI 49931, USA

[2]Health Research Institute, Michigan Technological University, Houghton, MI 49931, USA

[3]Department of Chemical Engineering, University of Massachusetts Amherst, Amherst, MA 01003, USA

[4]Institute for Applied Life Sciences, University of Massachusetts Amherst, Amherst, MA 01003, USA

*Correspondence: heldt@mtu.edu, perrys@engin.umass.edu

#Hydrophobicity Patch Analysis

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib.ticker import MaxNLocator


##import the pdb excel file.
#The file should be pre-processed and reformated before importing
path = r' pathfile/pdbfile.xlsx'

data2 = pd.read_excel(path, header=None, index_col = None)
data2.columns = ['model','atom_index','atom','resi','chain',
                 'resi_index','x_coord','y_coord','z_coord',
                 'unknwn1','unknwn2','atom_name']
del data2['unknwn1'], data2['unknwn2'], data2['atom_name']
cres = pd.DataFrame([], [], columns = ['model', 'atom', 'resi', 'chain', 'resi_index', 'x_coord', 'y_coord',
'z_coord'])

##Data arrangement- find hydrophobic residues and add it in the cres matrix
```

```
row1 = 1
model = 1
while row1 < len(data2):

    if data2.loc[row1, 'atom'] == "CA":
        if data2.loc[row1, 'resi'] == "ALA" or data2.loc[row1, 'resi'] =="MET" or data2.loc[row1, 'resi'] ==
"TRP" or data2.loc[row1, 'resi'] == "LEU" or data2.loc[row1, 'resi'] == "VAL" or data2.loc[row1, 'resi'] ==
"PHE" or data2.loc[row1, 'resi'] == "ILE":
            cres = cres.append({'model':data2.loc[row1, 'model'],
                                'atom_index':data2.loc[row1, 'atom_index'],
                                'resi':data2.loc[row1, 'resi'],
                                'resi_index':data2.loc[row1, 'resi_index'],
                                'chain' : data2.loc[row1, 'chain'],
                                'x_coord': data2.loc[row1, 'x_coord'],
                                'y_coord':data2.loc[row1, 'y_coord'],
                                'z_coord':data2.loc[row1, 'z_coord']},
                               ignore_index=True)

    row1 += 1
    continue

##Extracting SAR from cres table
#Surface accessible residues extracted from viperdb
#Activate this for PPV hydrophobicity
sar = pd.DataFrame([74, 92, 93, 157, 171, 201, 230, 250,
                    293, 309, 310, 329, 372, 414, 517, 567])



#Activate this for HRV hydrophobicity.
#These are the chain identity numbered as A = 1, B = 3, C = 2 as per pdb file. A, B, and C are the chain
#identities in viperdb. 1, 2, and 3 are the chain identities in pdb

# sar = pd.DataFrame([[79, 87, 90, 96, 134, 163, 180, 238,
#                      73, 80, 86, 197, 232,
#                      97, 140, 146, 159, 162, 230, 234],
#                     [1,1,1,1,1,1,1,1,
#                      3,3,3,3,3,
```

```
#                   2,2,2,2,2,2,2]])

# sar = sar.transpose()

cres_sar = pd.DataFrame([], [], columns = ['model', 'atom', 'resi',
                                    'resi_index', 'x_coord', 'y_coord',
                                    'z_coord'])
row2 = 0
row3 = 0

for row2 in range(0, len(cres)):
    row3 = 0

    for row3 in range(0, len(sar)):
        if  cres.loc[row2, 'resi_index'] == sar.iloc[row3, 0]: #and cres.loc[row2, 'chain'] == sar.iloc[row3, 1]:
#Activate this double boolean condition for viruses with multiple chains
            cres_sar = cres_sar.append({'model':cres.loc[row2, 'model'],
                            'atom_index':cres.loc[row2, 'atom_index'],
                            'resi':cres.loc[row2, 'resi'],
                            'resi_index':cres.loc[row2, 'resi_index'],
                            'x_coord': cres.loc[row2, 'x_coord'],
                            'y_coord': cres.loc[row2, 'y_coord'],
                            'z_coord': cres.loc[row2, 'z_coord']},
                           ignore_index=True)

        row3 += 1

    row2 += 1

##Calculating distance between the charged matrix
i = 0
dist_matrix = pd.DataFrame([], [])

#Loop to define all the residues as a reference residue
while  i < len(cres_sar):
    j = 0
    refx = cres_sar.loc[i, 'x_coord']
```

```
    refy = cres_sar.loc[i, 'y_coord']
    refz = cres_sar.loc[i, 'z_coord']

    #Loop for calculating the distance from the reference res to
    # the rest of res
    while j < len(cres_sar):
        testx = cres_sar.loc[j, 'x_coord']
        testy = cres_sar.loc[j, 'y_coord']
        testz = cres_sar.loc[j, 'z_coord']

        dist_matrix.loc[i, j] = np.sqrt((testx-refx)**2 +
                                (testy-refy)**2 +
                                (testz-refz)**2)
        dist_matrix.loc[j, i] = dist_matrix.loc[i, j]

        j += 1
        continue
    i += 1
    continue

##determination of g(r) for model 1
maxdist = max(dist_matrix.max())
particledensity = len(cres_sar)/(4/3*(np.pi)*(maxdist**3))
gofr = pd.DataFrame([],[])
rstart = 2
step = 2
rmax = 50
pcount_c = int((rmax - rstart)/step)
pcount = np.zeros(len(cres_sar)*pcount_c).reshape(len(cres_sar), pcount_c)


#Loop for gofr from each charged residue. This is a correlation or radial distribution function over spatial
data
for m in range (0, len(cres_sar)):
    r2 = 0
    gofrindex = 1
```

```
    for r in range (rstart, rmax, step):
        particlecount = 0

        for k in range (1, len(cres_sar)):

            if dist_matrix.loc[m, k] <= r:
                particlecount += 1

        gofr.loc[m, gofrindex] = particlecount/len(cres_sar)/(4*np.pi*(r**2)*step)/particledensity
        gofrindex += 1

        pcount[m, r2] = particlecount
        r2 += 1

#remove the backwall with high gofr in the data
gofr_mod = gofr.iloc[:, 1:]

#remove the particlecount backwall with high gofr in the data
pcount_mod = pcount[:, 1:]

##Graphing 3D bar plot
#setup the figure and axes
fig = plt.figure(figsize = [10, 5], dpi = 1200)
fig1 = fig.add_subplot(projection='3d')
fig1.view_init(20, -20)

#making x, y, z axes grid
_x = np.arange( 4, rmax, step)
_y = np.arange(len(cres_sar))
_xx, _yy = np.meshgrid(_x, _y)
x = _xx.ravel()
y = _yy.ravel()
z = np.zeros(gofr_mod.size)

#Define height, width, and breadth of each bar
dx = 0.75* np.ones_like(x)
dy = 0.75* np.ones_like(y)
```

```
dz = gofr_mod.to_numpy().flatten()

#plotting data in fig1 figure
color = plt.get_cmap('coolwarm')
colormap = [color((x_t-np.min(x))/np.max(x)) for x_t in x]
fig1.bar3d(x, y, z, dx, dy, dz, color = colormap, shade = True)

#Detailing axes
plt.xlabel('radius')
plt.xlim(0,50)
plt.ylabel('Hydrophobic residue')
plt.ylim(0, len(sar))
fig1.set_zlim(0, max(gofr_mod.max()) * 1.5)
fig1.yaxis.set_major_locator(MaxNLocator(integer = True))
fig1.zaxis.set_major_locator(MaxNLocator(integer = True))
mpl.rcParams.update({'font.size': 10})

plt.show()
```

#Charge Patch Analysis

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl


##import pdb excel file.
#The file should be pre-processed and reformated before importing
path = r'pathfile/pdbfile.xlsx'

data2 = pd.read_excel(path, header=None, index_col = None)
data2.columns = ['model','atom_index','atom','resi','chain',
                 'resi_index','x_coord','y_coord','z_coord',
```

```
            'unknwn1','unknwn2','atom_name']
del data2['unknwn1'], data2['unknwn2'], data2['atom_name']

cres = pd.DataFrame([], [], columns = ['model', 'atom', 'resi',
                                       'chain', 'resi_index', 'x_coord',
                                       'y_coord', 'z_coord'])

##Data arrangement- find charged residues and add it in the cres matrix
row1 = 1
model = 1
while row1 < len(data2):

    if data2.loc[row1, 'atom'] == "CA":
        if data2.loc[row1, 'resi'] == "LYS" or data2.loc[row1, 'resi'] =="ARG" or data2.loc[row1, 'resi'] == "HIS"
or data2.loc[row1, 'resi'] == "GLU" or data2.loc[row1, 'resi'] == "ASP":
            cres = cres.append({'model':data2.loc[row1, 'model'],
                                'atom_index':data2.loc[row1, 'atom_index'],
                                'resi':data2.loc[row1, 'resi'],
                                'resi_index':data2.loc[row1, 'resi_index'],
                                'chain' : data2.loc[row1, 'chain'],
                                'x_coord': data2.loc[row1, 'x_coord'],
                                'y_coord':data2.loc[row1, 'y_coord'],
                                'z_coord':data2.loc[row1, 'z_coord']},
                               ignore_index=True)

    row1 += 1
    continue

##Extracting SAR from cres table
#Surface accessible residues extracted from viperdb
#Activate this for PPV charge
# sar = pd.DataFrame([78, 89, 135, 155, 232, 272, 361, 366, 390, 392, 393, 407,
#                     475, 515, 551, 565]) #Activate this for PPV

#Activate this for HRV charge
#These are the chain indentity numbered as A = 1, B = 3, C = 2 as per pdb file. A, B, and C are the chain
identities in viperdb. 1, 2, and 3 are the chain identities in pdb
```

```
sar = pd.DataFrame([[81, 85, 86, 91, 94, 97, 101, 138, 161, 165, 208, 236,
                     276, 283, 285, 61, 63, 75, 78, 141, 69, 76, 87, 152,
                     155, 161, 210, 257],
                    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3,
                     3, 2, 2, 2, 2, 2, 2, 2, 2, 2]])
sar = sar.transpose()

cres_sar = pd.DataFrame([], [], columns = ['model', 'atom', 'resi',
                                    'resi_index', 'x_coord', 'y_coord',
                                    'z_coord'])
row2 = 0
row3 = 0

for row2 in range(0, len(cres)):
    row3 = 0

    for row3 in range(0, len(sar)):
        if  cres.loc[row2, 'resi_index'] == sar.iloc[row3, 0] and cres.loc[row2, 'chain'] == sar.iloc[row3, 1]:
            #Activate the and boolean for viruses with multiple chains
            cres_sar = cres_sar.append({'model':cres.loc[row2, 'model'],
                                'atom_index':cres.loc[row2, 'atom_index'],
                                'resi':cres.loc[row2, 'resi'],
                                'resi_index':cres.loc[row2, 'resi_index'],
                                'x_coord': cres.loc[row2, 'x_coord'],
                                'y_coord': cres.loc[row2, 'y_coord'],
                                'z_coord': cres.loc[row2, 'z_coord']},
                               ignore_index=True)

        row3 += 1

    row2 += 1

##Calculating distance between the charged matrix
i = 0
dist_matrix = pd.DataFrame([], [])

#Loop to define all the residues as a reference residue
```

```
while  i < len(cres_sar):
    j = 0
    refx = cres_sar.loc[i, 'x_coord']
    refy = cres_sar.loc[i, 'y_coord']
    refz = cres_sar.loc[i, 'z_coord']

    #Loop for calculating distance from the reference res to the rest of res
    while j < len(cres_sar):
        testx = cres_sar.loc[j, 'x_coord']
        testy = cres_sar.loc[j, 'y_coord']
        testz = cres_sar.loc[j, 'z_coord']

        dist_matrix.loc[i, j] = np.sqrt((testx-refx)**2 +
                                (testy-refy)**2 +
                                (testz-refz)**2)
        dist_matrix.loc[j, i] = dist_matrix.loc[i, j]

        j += 1
        continue
    i += 1
    continue


##determination of g(r) for model 1
maxdist = max(dist_matrix.max())
particledensity = len(cres_sar)/(4/3*(np.pi)*(maxdist**3))
gofr = pd.DataFrame([],[])
rstart = 2
step = 2
rmax = 50
pcount_c = int((rmax - rstart)/step)
pcount = np.zeros(len(cres_sar)*pcount_c).reshape(len(cres_sar), pcount_c)


#Loop for gofr from each charged residue.
#This is a correlation or radial distribution function over spatial data
for m in range (0, len(cres_sar)):
```

```
    r2 = 0
    gofrindex = 1

    for r in range (rstart, rmax, step):
        particlecount = 0

        for k in range (1, len(cres_sar)):

            if dist_matrix.loc[m, k] <= r:
                particlecount += 1

        gofr.loc[m, gofrindex] = particlecount/len(cres_sar)/(4*np.pi*(r**2)*step)/particledensity
        gofrindex += 1

        pcount[m, r2] = particlecount
        r2 += 1
#remove the backwall with high gofr in the data
gofr_mod = gofr.iloc[:, 1:]
#remove the particlecount backwall with high gofr in the data
pcount_mod = pcount[:, 1:]

##Graphing 3D bar plot
#setup the figure and axes
fig = plt.figure(figsize = [10, 5], dpi = 1200)
fig1 = fig.add_subplot(projection='3d')
fig1.view_init(20, -20)

#making x, y, z axes grid
_x = np.arange( 4, rmax, step)
_y = np.arange(len(cres_sar))
_xx, _yy = np.meshgrid(_x, _y)
x = _xx.ravel()
y = _yy.ravel()
z = np.zeros(gofr_mod.size)

#Define height, width, and breadth of each bar
dx = 0.75* np.ones_like(x)
```

```
dy = 0.75* np.ones_like(y)
dz = gofr_mod.to_numpy().flatten()

#plotting data in fig1 figure
color = plt.get_cmap('coolwarm')
colormap = [color((x_t-np.min(x))/np.max(x)) for x_t in x]
fig1.bar3d(x, y, z, dx, dy, dz, color = colormap, shade = True)

#Detailing axes
plt.xlabel('radius')
plt.xlim(0,50)
plt.ylabel('charged residue')
# plt.ylim(0, 30)
fig1.set_zlim(0, max(gofr_mod.max()) * 1.5)
mpl.rcParams.update({'font.size': 10})

plt.show()
```