```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 11 21:17:28 2024

@author: khaniyev
"""

import numpy as np
import pickle
import pandas as pd
from tqdm import tqdm
from datetime import datetime
import networkx
import gurobipy as gp
import scipy

from nilearn.image import get_data
from nilearn.image import new_img_like
from nilearn import image
from nilearn import plotting
from dipy.io.streamline import load_tractogram, save_tractogram
from nilearn.plotting import plot_epi
from dipy.io.vtk import transform_streamlines

#%%

mrtrix_connectome = pd.read_csv("/Users/khaniyev/Desktop/BRAIN/EgemenGok/HCP_Database/sub-10-WUMINN.csv", header=None)

mrtrix_connectome = mrtrix_connectome.iloc[1:,:]
mrtrix_connectome = mrtrix_connectome.iloc[:,1:]
mrtrix_connectome = mrtrix_connectome.to_numpy()

#%%

hcp_atlas_nocoreg = image.load_img("/Users/khaniyev/Desktop/BRAIN/EgemenGok/HCP_Database/hcpmmp1_parcels_nocoreg.nii")
hcp_atlas_coreg = image.load_img("/Users/khaniyev/Desktop/BRAIN/EgemenGok/HCP_Database/hcpmmp1_parcels_coreg.nii")
b0_nifti = image.load_img("/Users/khaniyev/Desktop/BRAIN/EgemenGok/HCP_Database/mean_b0_preprocessed.nii")


hcp_atlas_nocoreg_data = get_data(hcp_atlas_nocoreg)
hcp_atlas_nocoreg_data = np.round(hcp_atlas_nocoreg_data).astype("int")

hcp_atlas_coreg_data = get_data(hcp_atlas_coreg)
hcp_atlas_coreg_data = np.round(hcp_atlas_coreg_data).astype("int")

#hcp_atlas_coreg_data_copy = hcp_atlas_coreg_data.copy()
#hcp_atlas_coreg_data_copy[hcp_atlas_coreg_data_copy!=120] = 0

#hcp_atlas_coreg_copy  = image.new_img_like(hcp_atlas_coreg, hcp_atlas_coreg_data_copy)
#plot_epi(hcp_atlas_coreg_copy, display_mode="z")


plot_epi(hcp_atlas_nocoreg, display_mode="z", cut_coords=(-10,))
plot_epi(hcp_atlas_coreg, display_mode="z", cut_coords=(-10,))
plot_epi(b0_nifti, display_mode="z", cut_coords=(-10,))


coreg_resampled = image.resample_img(hcp_atlas_coreg, target_affine = b0_nifti.affine, target_shape = b0_nifti.shape,  interpolation="nearest")
hcp_atlas_coreg_data = get_data(coreg_resampled)
hcp_atlas_coreg_data = np.round(hcp_atlas_coreg_data).astype("int")

plot_epi(coreg_resampled, display_mode="z", cut_coords=(-10,))

#%%


all_tracts = load_tractogram("/Users/khaniyev/Desktop/BRAIN/EgemenGok/HCP_Database/tracks_1m.tck", reference="/Users/khaniyev/Desktop/BRAIN/EgemenGok/HCP_Database/mean_b0_preprocessed.

all_streamlines = all_tracts.streamlines

affine = all_tracts.affine
inv_affine = np.linalg.inv(affine)

aaa = transform_streamlines(all_streamlines, inv_affine)
bbb = np.round(np.vstack(aaa))
#bbb = np.floor(np.vstack(aaa))
ccc = [len(i) for i in aaa]
ddd = np.repeat(np.arange(0,len(aaa)), ccc)
coord = pd.DataFrame()
coord["SID"] = ddd
coord[["x", "y", "z"]] = bbb
coord.x = coord.x.astype("int")
coord.y = coord.y.astype("int")
coord.z = coord.z.astype("int")

#%%

coord_copy = coord.copy()
coord_copy.columns = ["SID","x2","y2","z2"]
ends = pd.concat([coord.groupby('SID').first(),coord_copy.groupby('SID').last()],axis = 1)
ends["SID"] = ends.index

#%%

voxel_to_parcel = pd.DataFrame(np.vstack((np.where(hcp_atlas_coreg_data !=0)[0],
np.where(hcp_atlas_coreg_data !=0)[1],
np.where(hcp_atlas_coreg_data !=0)[2],
hcp_atlas_coreg_data[hcp_atlas_coreg_data!=0])).T)
voxel_to_parcel.columns = ['x', 'y', 'z', 'Parcel_ID']

#%%

ends['voxel_str'] = ends["x"].astype("str") + "_" + ends["y"].astype("str") + "_" + ends["z"].astype("str")
ends['voxel_str2'] = ends["x2"].astype("str") + "_" + ends["y2"].astype("str") + "_" + ends["z2"].astype("str")
voxel_to_parcel['voxel_str'] = voxel_to_parcel["x"].astype("str") + "_" + voxel_to_parcel["y"].astype("str") + "_" + voxel_to_parcel["z"].astype("str")
voxel_to_parcel["VID"] = voxel_to_parcel.index

#%%

start_voxels = ends[["x","y","z","voxel_str","SID"]]
start_parcels = pd.merge(start_voxels, voxel_to_parcel, on='voxel_str', suffixes=('_start','_parcel_start'), how="left")
start_parcels = start_parcels[np.invert(start_parcels.Parcel_ID.isna())]

###
end_voxels = ends[["x2","y2","z2","voxel_str2","SID"]]
end_voxels.columns = ["x","y","z","voxel_str","SID"]

end_parcels = pd.merge(end_voxels, voxel_to_parcel, on='voxel_str', suffixes=('_end','_parcel_end'), how="left")
end_parcels = end_parcels[np.invert(end_parcels.Parcel_ID.isna())]
```

```
start_parcels_ = start_parcels.drop(['x_start','y_start','z_start'],axis=1)

start_parcels_.columns = ['voxel_str', 'SID', 'x_start', 'y_start','z_start', 'Parcel_ID', 'VID']

end_parcels_ = end_parcels.drop(['x_end','y_end','z_end'],axis=1)
end_parcels_.columns = ['voxel_str', 'SID', 'x_end', 'y_end','z_end', 'Parcel_ID', 'VID']


stream_start_end = pd.merge(start_parcels_, end_parcels_, on='SID', how='inner', suffixes=('_start', '_end'))


#%% assign stream ends to closest parcels
"""

start_voxels = ends[["x","y","z","voxel_str","SID"]]
start_parcels = pd.merge(start_voxels, voxel_to_parcel, on='voxel_str', suffixes=('_start','_parcel_start'), how="left")

indices_to_fill = start_parcels.Parcel_ID.isna()
df_1 = start_parcels[indices_to_fill][["x_start", "y_start", "z_start"]]

closest_parcel = np.zeros(len(df_1))
closest_voxel = pd.DataFrame(np.zeros((len(df_1), 6)))
closest_voxel.columns = ["VID", "x", "y", "z", "Parcel_ID", "voxel_str"]
distance_to_closest_parcel = 1000*np.ones(len(df_1))
for i in range(1, (N+1)):
    if (i!=120)&(i!=300):
        print(i)
        df_2 = voxel_to_parcel[voxel_to_parcel.Parcel_ID == i]
        dist_from_voxels = scipy.spatial.distance.cdist(df_1, df_2[["x", "y", "z"]], metric='cityblock')
        dist_from_parcel = np.min(dist_from_voxels, axis=1)
        cl_voxel = df_2.iloc[np.argmin(dist_from_voxels, axis=1)][["VID", "x", "y", "z", "Parcel_ID","voxel_str"]]
        cl_voxel = cl_voxel.reset_index(drop=True)

        indices_to_update = dist_from_parcel < distance_to_closest_parcel
        closest_parcel[indices_to_update] = i
        closest_voxel.loc[indices_to_update, :] = cl_voxel[indices_to_update]
        distance_to_closest_parcel[indices_to_update] = dist_from_parcel[indices_to_update]
        print(np.sum(indices_to_update))


start_parcels.loc[indices_to_fill, ["VID", "x_parcel_start", "y_parcel_start", "z_parcel_start", "Parcel_ID", "voxel_str"]] = closest_voxel.values
start_parcels["dist_to_parcel"] = 0
start_parcels.loc[indices_to_fill, "dist_to_parcel" ] = distance_to_closest_parcel
start_parcels__ = start_parcels.copy()
start_parcels__ = start_parcels__[start_parcels__.dist_to_parcel <= 1]


###
end_voxels = ends[["x2","y2","z2","voxel_str2","SID"]]
end_voxels.columns = ["x","y","z","voxel_str","SID"]

end_parcels = pd.merge(end_voxels, voxel_to_parcel, on='voxel_str', suffixes=('_end','_parcel_end'), how="left")

indices_to_fill = end_parcels.Parcel_ID.isna()
df_1 = end_parcels[indices_to_fill][["x_end", "y_end", "z_end"]]

closest_parcel = np.zeros(len(df_1))
closest_voxel = pd.DataFrame(np.zeros((len(df_1), 6)))
closest_voxel.columns = ["VID", "x", "y", "z", "Parcel_ID", "voxel_str"]


distance_to_closest_parcel = 1000*np.ones(len(df_1))
for i in range(1, (N+1)):
    if (i!=120)&(i!=300):
        print(i)
        df_2 = voxel_to_parcel[voxel_to_parcel.Parcel_ID == i]
        dist_from_voxels = scipy.spatial.distance.cdist(df_1, df_2[["x", "y", "z"]], metric='cityblock')
        dist_from_parcel = np.min(dist_from_voxels, axis=1)
        cl_voxel = df_2.iloc[np.argmin(dist_from_voxels, axis=1)][["VID", "x", "y", "z", "Parcel_ID", "voxel_str"]]
        cl_voxel = cl_voxel.reset_index(drop=True)

        indices_to_update = dist_from_parcel < distance_to_closest_parcel
        closest_parcel[indices_to_update] = i
        closest_voxel.loc[indices_to_update, :] = cl_voxel[indices_to_update]
        distance_to_closest_parcel[indices_to_update] = dist_from_parcel[indices_to_update]
        print(np.sum(indices_to_update))

end_parcels.loc[indices_to_fill, ["VID", "x_parcel_end", "y_parcel_end", "z_parcel_end", "Parcel_ID", "voxel_str"]] = closest_voxel.values
end_parcels["dist_to_parcel"] = 0
end_parcels.loc[indices_to_fill, "dist_to_parcel" ] = distance_to_closest_parcel
end_parcels__ = end_parcels.copy()
end_parcels__ = end_parcels__[end_parcels__.dist_to_parcel <= 1]


start_parcels_ = start_parcels__.drop(['x_start','y_start','z_start', "dist_to_parcel"],axis=1)

start_parcels_.columns = ['voxel_str', 'SID', 'x_start', 'y_start','z_start', 'Parcel_ID', 'VID', "voxel_idx"]

end_parcels_ = end_parcels__.drop(['x_end','y_end','z_end', "dist_to_parcel"],axis=1)
end_parcels_.columns = ['voxel_str', 'SID', 'x_end', 'y_end','z_end', 'Parcel_ID', 'VID', "voxel_idx"]


stream_start_end = pd.merge(start_parcels_, end_parcels_, on='SID', how='inner', suffixes=('_start', '_end'))
"""
#%% Creating indx for voxels

shape = hcp_atlas_coreg_data.shape
stream_start_end["voxel_idx_start"] = shape[1] * shape[2] * stream_start_end["x_start"] + shape[2] * stream_start_end["y_start"] + stream_start_end["z_start"]
stream_start_end["voxel_idx_end"] = shape[1] * shape[2] * stream_start_end["x_end"] + shape[2] * stream_start_end["y_end"] + stream_start_end["z_end"]
voxel_to_parcel["voxel_idx"] = shape[1] * shape[2] * voxel_to_parcel["x"] + shape[2] * voxel_to_parcel["y"] + voxel_to_parcel["z"]

stream_start_end.VID_start = stream_start_end.VID_start.astype("int")
stream_start_end.x_start = stream_start_end.x_start.astype("int")
stream_start_end.y_start = stream_start_end.y_start.astype("int")
stream_start_end.z_start = stream_start_end.z_start.astype("int")
stream_start_end.Parcel_ID_start = stream_start_end.Parcel_ID_start.astype("int")
stream_start_end.voxel_idx_start = stream_start_end.voxel_idx_start.astype("int")

stream_start_end.VID_end = stream_start_end.VID_end.astype("int")
stream_start_end.x_end = stream_start_end.x_end.astype("int")
stream_start_end.y_end = stream_start_end.y_end.astype("int")
stream_start_end.z_end = stream_start_end.z_end.astype("int")
stream_start_end.Parcel_ID_end = stream_start_end.Parcel_ID_end.astype("int")
stream_start_end.voxel_idx_end = stream_start_end.voxel_idx_end.astype("int")

#%%

N = mrtrix_connectome.shape[0]
```

```
connectivity_graph = np.zeros((N,N)) #G
for i in range(len(stream_start_end)):
    start = stream_start_end["Parcel_ID_start"].iloc[i] -1
    end = stream_start_end["Parcel_ID_end"].iloc[i] -1
    connectivity_graph[start, end] = connectivity_graph[start, end] + 1
    connectivity_graph[end,start] = connectivity_graph[end, start] + 1


connectivity_graph_nx = networkx.from_numpy_array(connectivity_graph)
connectivity_graph_mrtrix = networkx.from_numpy_array(mrtrix_connectome)

nx_binary = connectivity_graph.copy()
nx_binary[nx_binary>0] = 1

mrtx_binary = mrtrix_connectome.copy()
mrtx_binary[mrtx_binary>0] = 1


#$$

mrtx_network = networkx.from_numpy_array(mrtx_binary)
nx_network = networkx.from_numpy_array(nx_binary)

np.sum(mrtx_binary)/(N*(N-1))
np.sum(nx_binary)/(N*(N-1))

networkx.global_efficiency(mrtx_network)
networkx.global_efficiency(nx_network)
```