CZ4071 – Network Science

Assignement 1– 09/04/18

De Bodinat, Jean
N1701748F

Index

1. Introduction

In order to make an GUI that could be used by any one, without requiring prior knowledge in programming, or in network science, I chose to user Python's Tkinter library. Even if the GUI is unfortunately not very good looking, it is rather functional, easy to use and easy to launch.

I preferred to have a interactive GUI rather than a question/answer system. The user can load any network any time and can ask for any characteristic anytime.

2. Challenges.

- Making a method that can output **a graph given any PostgreSQL** database as an input.

While building the program, I felt like the hardest task was to build a method that could read any PostgreSQL database. Indeed, as we do not know beforehand the dependencies between different tables, the program has to be very general and accept any type of table.
Also, for example, the TPC-H database used as a test, tables do not have a unique ID, thus since the nodes need a unique ID, it is necessary to create one.
Reading any PostgreSQL database creates a lot of difficulties:

- Making unique IDs.
- Complex SQL commands.
- Verifying if nodes already exist (Unknown dependencies order).

- Having a **functional GUI**.

Making a good-looking GUI in Python is a complex task. I first wanted to use tools like Electron, where the UI is programmed in JavaScript, but it caused problems when packaging the app if its runs a python script in the background.
Thus, I decided to use Tkinter, which, even if not very good-looking, is nonetheless functional.
The difficulty was to restart everything from the beginning because the GUI we made in Project 1, was not at all adapted to accept a graph as a parameter.

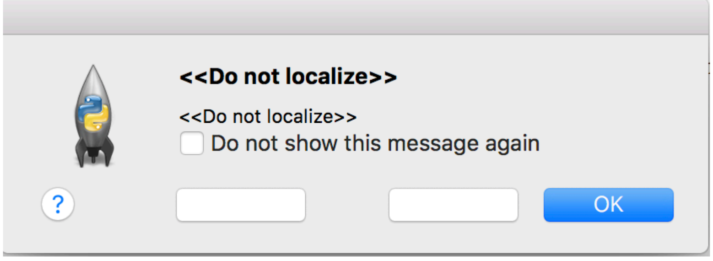
3. Problems, Limits and Bugs

- **Limits**:

No Management of Errors. If the SQL database name, or user name or password are wrong, nothing will happen.

- **Bugs**:

When a graph is displayed, all the previously opened window reappears, and this window appears:

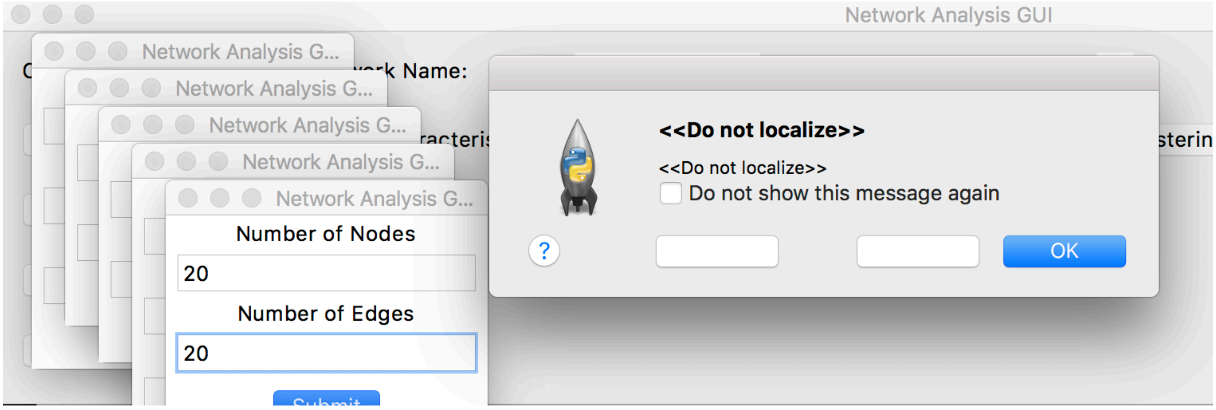


<<Do not localize>>

<<Do not localize>>

□ Do not show this message again

Thus, the user has to manually close all of those windows.



<<Do not localize>>

<<Do not localize>>

□ Do not show this message again

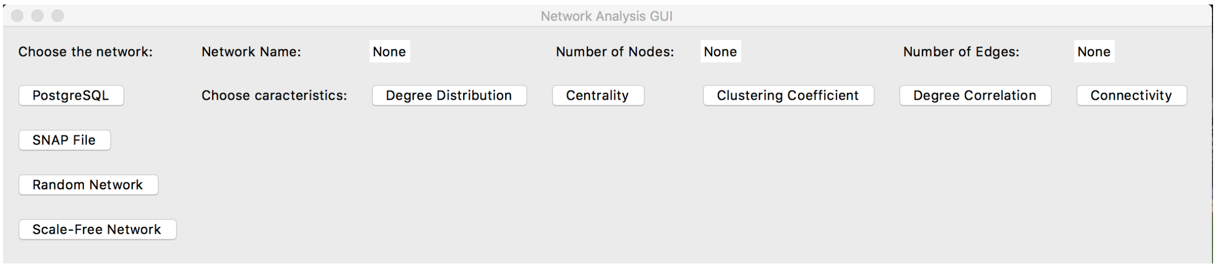
Number of Nodes: 20

Number of Edges: 20

Example of the windows bug

4. The GUI

- The main GUI:



Choose the network:

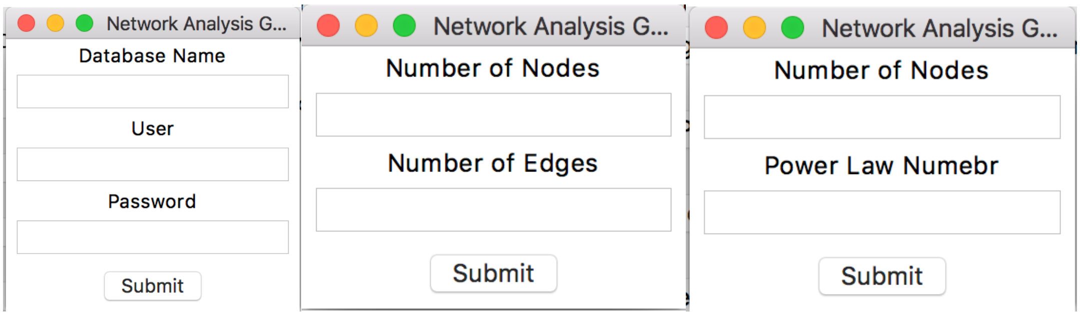
Network Name: None

Number of Nodes: None

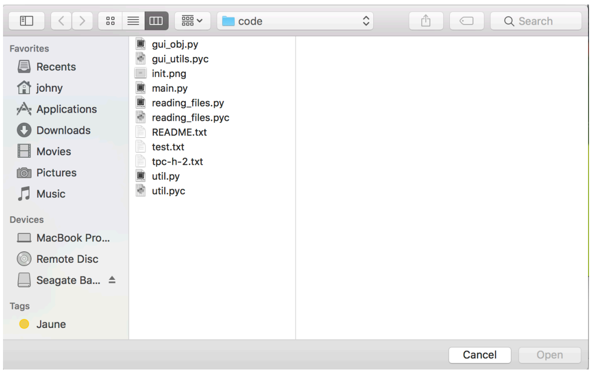
Number of Edges: None

Choose caracteristics:

- Entering parameters to create a network.

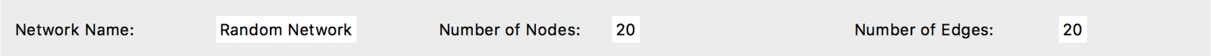


- Selecting a SNAP File.



gui_obj.py
gui_utils.pyc
init.png
main.py
reading_files.py
reading_files.pyc
README.txt
test.txt
tpc-h-2.txt
util.py
util.pyc

- Updated labels:



Network Name: Random Network

Number of Nodes: 20

Number of Edges: 20

5. Methods Explanation

- **Reading_files.py:**

- Find_table_id(table_name, table_list):

Find the table's id in the table list according to the table name.

- Build_table_id(table_name, table_list):

Build a string that will be added at the end of the non-unique IDs in tables. This string will make every ID unique, as one number will be allocated to every table.
Its difficulty resides in the fact that we do not know how many tables will exist, thus it has to be very moldable.

- Read_from_sql(dbname, user, password):

Using the two previous functions, this one will output a network if the inputs were correct.

- **Util.py:**

A file which contains a few methods used to calculate some network properties.

- **Gui_obj.py:**

The main GUI File. It contains an Object that defines the GUI. When initialized, it creates the GUI, and its methods will create networks and calculate their properties.

6. Softcopy of the Code

The main file is gui_obj, as the gui will be launched from it.

a. Gui_obj.py

```
import ttk
import Tkinter as tk
import snap, numpy as np, matplotlib.image as mpimg, matplotlib.pyplot as
plt
import gui_utils as gu
import reading_files as rf
from Tkinter import *
from tkFileDialog import askopenfilename
import tkMessageBox
import util
import PIL.Image
import PIL.ImageTk



class NetworkMenu(tk.Tk):

    def __init__(self, *args, **kwargs):

        #INTIALIZING GRAPH CHARACTERISTICS
        self.graph = snap.GenRndGnm(snap.PUNGraph, 10, 10)
        self.graph_name = ""
        self.number_of_nodes = 0
        self.number_of_edges = 0

        #CREATING THE ROOT WINDOW
        self.root = tk.Tk()
        self.root.title("Network Analysis GUI")

        #CREATING THE DIPLAYED FRAME
        self.mainframe = ttk.Frame(self.root, padding="3 3 12 12")
        self.mainframe.grid(column=0, row=0, sticky=(N, W, E, S))
        self.mainframe.columnconfigure(0, weight=1)
        self.mainframe.rowconfigure(0, weight=1)

        #NETWORK CHOICE
        ttk.Label(self.mainframe, text="Choose the
network:").grid(column=0, row=0, sticky=W)
        ttk.Button(self.mainframe, text="PostgreSQL",
command=self.load_sql).grid(column=0, row=1, sticky=W)
        ttk.Button(self.mainframe, text="SNAP File",
command=self.open_snap_file).grid(column=0, row=2, sticky=W)
        ttk.Button(self.mainframe, text="Random Network",
command=self.random_network).grid(column=0, row=3, sticky=W)
        ttk.Button(self.mainframe, text="Scale-Free Network",
command=self.scale_free_network).grid(column=0, row=4, sticky=W)

        #CHARACTERISTICS CHOICE
        ttk.Label(self.mainframe, text="Choose
caracteristics:").grid(column=1, row=1, sticky=W)
```

```
        ttk.Button(self.mainframe, text="Degree Distribution",
command=self.degree_distribution).grid(column=2, row=1, sticky=W)
        ttk.Button(self.mainframe, text="Centrality",
command=self.centrality).grid(column=3, row=1, sticky=W)
        ttk.Button(self.mainframe, text="Clustering Coefficient",
command=self.clust_coef).grid(column=4, row=1, sticky=W)
        ttk.Button(self.mainframe, text="Degree Correlation",
command=self.degree_correlation).grid(column=5, row=1, sticky=W)
        ttk.Button(self.mainframe, text="Connectivity",
command=self.connectivity).grid(column=6, row=1, sticky=W)

        #NETWORK CHARACTERISTICS DISPLAY
        ttk.Label(self.mainframe, text="Network Name: ").grid(column=1,
row=0, sticky=W)
        ttk.Label(self.mainframe, text=" Number of Nodes: ").grid(column=3,
row=0, sticky=W)
        ttk.Label(self.mainframe, text=" Number of Edges: ").grid(column=5,
row=0, sticky=W)


        self.nameLabel = tk.Label( self.mainframe, text = "None" )
        self.nameLabel.grid( column = 2, row = 0, sticky = W )
        self.nameLabel.pack()

        self.numberOfNodes = tk.Label (self.mainframe, text = "None" )
        self.numberOfNodes.grid( column = 4, row = 0, sticky = W )
        self.numberOfNodes.pack()

        self.numberOfEdges = tk.Label( self.mainframe, text = "None" )
        self.numberOfEdges.grid( column = 6, row = 0, sticky = W )
        self.numberOfEdges.pack()

        for child in self.mainframe.winfo_children():
child.grid_configure(padx=10, pady=10)

    #UPDATE THE NETWORK'S DIPLAYED CHARACTERISTICS
    def add_characteristics( self, graph, graph_name ):

        nodes = str(graph.GetNodes())
        edges = str(graph.GetEdges())

        self.nameLabel.configure( text = graph_name )
        self.numberOfNodes.configure( text = nodes )
        self.numberOfEdges.configure( text = edges )


    #OPEN THE SNAP FILE -> NEEDS TO BE AN EDGE LIST
    def open_snap_file( self ):

        filename = askopenfilename() # show an "Open" dialog box and return
the path to the selected file
        self.graph = snap.LoadEdgeList(snap.PUNGraph, filename, 0, 1)

        self.graph_name = filename
        self.add_characteristics(  self.graph, self.graph_name )

    #LOAD THE SQL FILE -> NEEDS TO HAVE THE RIGHT CREDENTIALS
    def load_sql( self ):

        d = SQLDialog(self.root)
        self.root.wait_window(d.top)
        dbname = d.dbname
        user = d.user
        password = d.password


        self.graph = rf.read_from_sql( dbname, user, password )

        self.graph_name = dbname
        self.add_characteristics(  self.graph, self.graph_name )

    #CREATE A RANDOM NETWORK
    def random_network( self ):

        d = RnDialog(self.root)
        self.root.wait_window(d.top)

        nodes = d.nodes
        edges = d.edges

        self.graph = snap.GenRndGnm(snap.PUNGraph, nodes, edges)

        self.graph_name = "Random Network"
        self.add_characteristics( self.graph, self.graph_name )

    #CREATE A SCALE FREE NETWORK
    def scale_free_network( self ):

        d = SfDialog(self.root)
        self.root.wait_window(d.top)

        nodes = d.nodes
        power = d.power

        self.graph = snap.GenRndPowerLaw(nodes, power)

        self.graph_name = "Scale-Free Network"
        self.add_characteristics(  self.graph, self.graph_name )

    #COMPUTES THE DEGREE DISTRIBUTION, SHOW THE GRAPH
    def degree_distribution( self ):

        snap.PlotOutDegDistr( self.graph, "Degree_Distribution", " Graph
Degree Distribution")
        img = mpimg.imread("outDeg.Degree_Distribution.png")
        plt.figure()
        imgplot = plt.imshow(img)
        plt.show()

    def centrality( self ):
        tkMessageBox.showinfo(message = 'Graph degree centrality: ' +
str(util.getDegCentr(self.graph)))

    def clust_coef( self ):
        snap.PlotClustCf( self.graph, "Clust_Coef", "Graph Clustering
Coefficient")
        img = mpimg.imread("ccf.Clust_Coef.png")
        plt.figure()
        imgplot = plt.imshow(img)
        plt.show()
    def degree_correlation( self ):
        plotPath = util.plotDegCorr( self.graph, "Degree_Correlation")
        img = mpimg.imread(plotPath)
        plt.figure()
        imgplot = plt.imshow(img)
        plt.show()


    def connectivity( self ):
        snap.PlotSccDistr(self.graph, "Connectivity", "Connectivity")
        img = mpimg.imread("scc.Connectivity.png")
        plt.figure()
        imgplot = plt.imshow(img)
        plt.show()


#THE SQL DIALOG TO ENTER NAME, USER AND PASSWORD
class SQLDialog:

    def __init__(self, parent):

        top = self.top = Toplevel(parent)

        Label(top, text="Database Name").pack()
        self.dbname_entry = Entry(top)
        self.dbname_entry.pack(padx=5)

        Label(top, text="User").pack()
        self.user_entry = Entry(top)
        self.user_entry.pack(padx=5)

        Label(top, text="Password").pack()
        self.password_entry = Entry(top)
        self.password_entry.pack(padx=5)

        b = Button(top, text="Submit", command=self.ok)
        b.pack(pady=5)

    def ok(self):

        self.dbname = str(self.dbname_entry.get())
        self.user = str(self.user_entry.get())
        self.password = str(self.password_entry.get())

        self.top.destroy()

#SIMPLE DIALOG TO ENTER RANDOM NETWORK CHARACTERISTICS
class RnDialog:

    def __init__(self, parent):

        top = self.top = Toplevel(parent)

        Label(top, text="Number of Nodes").pack()
        self.nodes_entry = Entry(top)
        self.nodes_entry.pack(padx=5)

        Label(top, text="Number of Edges").pack()
        self.edges_entry = Entry(top)
        self.edges_entry.pack(padx=5)
        b = Button(top, text="Submit", command=self.ok)
        b.pack(pady=5)

    def ok(self):

        self.nodes = int(self.nodes_entry.get())
        self.edges = int(self.edges_entry.get())

        self.top.destroy()

#SIMPLE DIALOG TO ENTER SCALE FREE CHARACTERISTICS
class SfDialog:

    def __init__(self, parent):

        top = self.top = Toplevel(parent)

        Label(top, text="Number of Nodes").pack()
        self.nodes_entry = Entry(top)
        self.nodes_entry.pack(padx=5)

        Label(top, text="Power Law Numebr").pack()
        self.power_entry = Entry(top)
        self.power_entry.pack(padx=5)

        b = Button(top, text="Submit", command=self.ok)
        b.pack(pady=5)

    def ok(self):

        self.nodes = int(self.nodes_entry.get())
        self.power = float(self.power_entry.get())

        self.top.destroy()

#LAUNCH THE GUI LOOP
app = NetworkMenu()
app.root.mainloop()
```

b. **Reading_files.py**

```
import psycopg2
import snap

#TO FIND THE TABLE ID ACCORDING TO ITS NAME
def find_table_id( table_name, table_list ):

    for i in table_list:
        if (i[0] ==  table_name):
            return i[1]

    return 0

#BUILDING THE TABLE ID
def build_table_id( table_name, table_list ):

    #INTIALIZING COUNTERS
    list_number = len(table_list)
    power_ten_list = list_number
    count_list = 0

    #GETTING THE NUMBER OF TABLES DIVIDED BY TEN
    while (power_ten_list > 0 ):

        power_ten_list = power_ten_list//10
        count_list +=1

    #FIND THE TABLE IN THE LIST
    table_number = find_table_id( table_name, table_list )
    power_ten_table = table_number
    count_table = 0

    #WHILE THE PO
    while (power_ten_table > 0 ):
        power_ten_table = power_ten_table//10
        count_table +=1

    table_str_id = ""

    iterator = 0
    diff = count_list - count_table
    while( iterator < diff ):
        table_str_id += "0"
        iterator +=1

    table_str_id += str( table_number )

    return table_str_id



def read_from_sql( dbname, user, password ) :

    try:
        #CONNECT TO POSTGRESQL -> CHANGE NECESSARY PARAMETERS
        connection_str = "dbname='" + dbname + "' user='" + user + "'
password='" + password + "' "
        conn = psycopg2.connect( connection_str )
    except:
        print("I am unable to connect to the database")
    cur = conn.cursor()

    #CREATING A UNDIRECTED GRAPH -> make it easy
    database_network = snap.TUNGraph.New()

    #GET ALL TABLES FROM THE RELATIONAL DATABASE
    all_items_str = "SELECT information_schema.TABLES.TABLE_NAME FROM
information_schema.TABLES where table_schema='public'"
    cur.execute("" + all_items_str + "")
    tables = cur.fetchall()

    #CREATING THE TABLE ID CODE TAB
    id_code = []
    for i, table in enumerate(tables, 1):
        #ASSIGNING TO EACH TABLE A UNIQUE INTEGER ID
        id_code.append([table[0], i ])

    #FOREIGN KEYS TAB INIT
    foreign_keys_ref = []

    for table in tables :
        all_requested_keys = ""
        all_requested_foreign_keys = ""

        #FOREIGN KEYS REFERENCES TAB INIT
        foreign_keys_ref = []

        #NAME OF THE TABLE IS THE FIRST ELEMENT
        table_name = table[0]

        #REQUEST TO GET THE PRIMARY KEY(S) NAME(S) OF THE TABLE
        primary_key_request = "SELECT c.column_name, c.ordinal_position
FROM information_schema.key_column_usage AS c LEFT JOIN
information_schema.table_constraints AS t ON t.constraint_name =
c.constraint_name WHERE t.table_name = '" + table_name + "' AND
t.constraint_type = 'PRIMARY KEY'"
        cur.execute( "" + primary_key_request  + "" )
        primary_key_char = cur.fetchall()

        #ADD ALL PRIMARY KEYS TO THE REQUEST STRING
        for primary in primary_key_char:
            all_requested_keys +=  "\"" + primary[0] + "\","

        #REQUEST TO GERT THE FOREIGN KEY(S) NAME(S) OF THE TABLE
        foreign_key_request = "SELECT c.column_name, c.ordinal_position
FROM information_schema.key_column_usage AS c LEFT JOIN
information_schema.table_constraints AS t ON t.constraint_name =
c.constraint_name WHERE t.table_name = '" + table_name + "' AND
t.constraint_type = 'FOREIGN KEY'"
        cur.execute( "" + foreign_key_request + "" )
        foreign_key_char = cur.fetchall()

        #ADD ALL FOREIGN KEYS TO THE REQUEST
        for foreign in foreign_key_char:

            #MODIFIYING THE NAME TO MATCH THE INFORMATION SCHEMA NAMING
            foreign_key_name = "" + table_name + "_" + foreign[0] + "_fkey"
            all_requested_keys +=  "\"" + foreign[0] + "\","

            #GET THE REFERENCED PRIMARY KEYS
            foreign_key_references_request = "SELECT
r.unique_constraint_name FROM information_schema.referential_constraints AS
r WHERE r.constraint_name = '" + foreign_key_name + "'  "
            cur.execute( "" + foreign_key_references_request + "" )
            foreign_keys_ref.append(cur.fetchall()[0][0][:-5])

        #DELETE THE LAST COMA FROM THE STRING
        all_requested_keys = all_requested_keys[:-1]

        #GET ALL THE NECESSARY KEYS
        item_request = "SELECT " + all_requested_keys + " FROM \"" +
table_name + "\""
        cur.execute(  "" + item_request + "" )
        items = cur.fetchall()

        #ITERATE THROUGH THE ITEMS
        for item in items:

            #GET THE NUMBER OF EACH KEY TYPE
            iterator = 0
            number_of_primary_keys = len(primary_key_char)
            number_of_foreign_keys = len(foreign_key_char)

            #BUILD THE ITEM'S ID BY ADDING ALL HIS PRIMARY KEYS
            item_id = ""
            for p_key in item[:number_of_primary_keys]:
                item_id += str(p_key)

            #CALL THE BUILD FUNCTION WHICH WILL ADD THE TABLE ID TO THE
ITEM
            item_id += build_table_id( table_name, id_code )
            #print item_id

            #PYTHON CANT HANDLE ID HIGHER THAN THIS NUMBER, WE HAVE NO
CHOICE BUT TO SKIP
            if int(item_id) > 999999999:
                continue

            #IF THE NODE DOESNT EXIST, THEN CREATE IT
            if not database_network.IsNode( int(item_id) ):
                    database_network.AddNode( int(item_id) )

            #CREATE AN EDGE FOR EACH FOREIGN KEY IN THE TABLE
            for f_key_num, f_key in
enumerate(item[number_of_primary_keys:],1):

                #CALL THE BUILD FUNCTION WHICH WILL ADD THE TABLE ID TO THE
FOREIGN KEY ID
                f_key_table = foreign_keys_ref[f_key_num-1]
                item_linked_id = str(f_key) + build_table_id( f_key_table,
id_code )

                #IF THE NODE DOESNT EXIST, THEN CREATE IT
                if not database_network.IsNode( int(item_linked_id) ):
                   database_network.AddNode( int(item_linked_id) )

                #ADD THE EDGE
                database_network.AddEdge( int(item_id), int(item_linked_id)
)

    return database_network
```

c. Utils.py

```
import snap, numpy as np, matplotlib.image as mpimg, matplotlib.pyplot as
plt, os

MAX_XTICKS_NUM = 25


def computeDegCorr(graph):
      knn = {}
      for u in graph.Nodes():
            ki = u.GetDeg()

            # Isolated nodes
            if ki == 0:
                  continue

            ksum = 0.
            for i in range(ki):
                  vid = u.GetNbrNId(i)
                  ksum += graph.GetNI(vid).GetDeg()
            ksum = ksum / ki

            if ki not in knn:
                  knn[ki] = []
            knn[ki].append(ksum)

      knn_arr = []
      for ki in knn:
            knn_arr.append( (ki, sum(knn[ki]) / len(knn[ki])) )
      knn_ndarr = np.array(knn_arr, dtype=float)

      sorted_ks = np.argsort(knn_ndarr[:, 0])
      knn_ndarr = knn_ndarr[sorted_ks]
      return knn_ndarr

def plotDegCorr(graph, name):
      out_fname = 'degcorr' + name + '.png'
      knn = computeDegCorr(graph)
      plt.clf()
      plt.figure(1)
      plt.plot(knn[:, 0], knn[:, 1], '-x')
      plt.subplots_adjust(left=0.1, bottom=0.075, right=1., top=1.,
wspace=0., hspace=0.)


      if knn[:, 0].max() > MAX_XTICKS_NUM:
            skip = int(knn[:, 0].max()) / MAX_XTICKS_NUM
            plt.xticks( np.arange(0, knn[:, 0].max() + 1 + skip, skip) )
      else:
            plt.xticks(np.arange(knn[:, 0].max() + 1))

      plt.ylim(knn[:, 1].min(), knn[:, 1].max())
      plt.xlabel('Degree', fontsize=16)
      plt.ylabel('Degree Correlation', fontsize=16)
      plt.yscale('log')
      plt.xscale('log')
      plt.grid(True)
      plt.savefig(out_fname, dpi=300, format='png')
      plt.close()
      return os.path.abspath(out_fname)


def getDegCentr(graph):
      nid = snap.GetMxDegNId(graph)
      CDn = snap.GetDegreeCentr(graph, nid)
      n = graph.GetNodes()

      freeman_nom = 0.

      for NI in graph.Nodes():
            CDi = snap.GetDegreeCentr(graph, NI.GetId())
            freeman_nom += CDn - CDi

      return freeman_nom / (n - 2)
```