BrunoFBessa_5881890_P2_codigo

May 24, 2021

0.1 SFI5904 - Redes Complexas

Projeto Prático 2: Modelo de redes espaciais Primeiro Semestre de 2021

Docente: Luciano da Fontoura Costa (luciano@ifsc.usp.br) Estudante: Bruno F. Bessa (num. 5881890, bruno.fernandes.oliveira@usp.br) Universidade de São Paulo, São Carlos, Brasil.

Escopo do projeto:

Implementar redes espaciais (geográficos) considerando Voronoi, círculos de raio R e Waxman a partir de de um determinado número de nós N e um grau médio desejado.

Visualizar algumas das redes geradas. Apresentar: - os histogramas de frequência relativa dos graus, - coeficientes de aglomeração e distâncias mínimas, identificando nas respectivas legendas a média e o desvio padrão.

[1]:
```
# Importação de bibliotecas necessárias para o processamento e visualização

import random
import numpy as np
import scipy
import math
import networkx as nx
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme()
from IPython import display
```

[18]:
```
# Definição das redes e rotinas para cálculo de métricas

class SpatialPoint():

    """
    Define os pontos do grafo em um plano 2d.
    As redes espaciais serão formadas por um conjunto de objetos desta classe.
    """

    def __init__(self, index: str, box_size: int) -> None:
        self.index = index
        self.x = random.uniform(0, 1) * box_size
```

```
        self.y = random.uniform(0, 1) * box_size

    def get_coordinates(self) -> np.array:
        return np.array([(self.x, self.y)])

    def get_distance(self, other) -> float:
        p1_coord = self.get_coordinates()
        p2_coord = other.get_coordinates()

        dist = scipy.spatial.distance.cdist(p1_coord, p2_coord, 'euclidean')
        return dist[0][0]

    def __repr__(self) -> None:
        return repr([(self.x, self.y)])


def spatial_network_voronoi(N: int,
                            box_size: int = 1,
                            plot: bool = True,
                            file_name: str = None) -> nx.classes.graph.Graph:

    """
    Define as conexões (i,j) = (j,i) para todos os pares de pontos se eles␣
↪possuem fronteiras adjacentes
    nas células de Voronoi.

    """

    G = nx.Graph()

    spatial_points = [SpatialPoint(i, box_size) for i in range(N)]
    points2d_aux = [point_arr.get_coordinates() for point_arr in spatial_points]
    points2d = []
    for point_arr_aux in points2d_aux:
        points2d.append(list(point_arr_aux[0]))

    # Cálculo das fronteiras de Voronoi:
    vor = scipy.spatial.Voronoi(points2d)

    # Criação da rede baseada em céluas adjacentes
    edges = vor.ridge_points
    for edge in list(edges):
        G.add_edge(list(edge)[0], list(edge)[1])

    # Para calcularmos medidas de distância precisaremos remover nós não␣
↪conectados
    # No trecho abaixo mantemos somente o maior componente conctado da rede.
```

```
    G = G.to_undirected()
    G.remove_edges_from(nx.selfloop_edges(G))
    Gcc = sorted(nx.connected_components(G), key=len, reverse=True)
    G = G.subgraph(Gcc[0])
    G = nx.convert_node_labels_to_integers(G, first_label=0)

    if plot:
        fig, (ax1, ax2) = plt.subplots(1, 2)
        fig.suptitle("Células de Voronoi e Rede para {} pontos espaciais␣
↪aleatórios".format(N), fontsize=15)
        scipy.spatial.voronoi_plot_2d(vor, ax1)
        pos = nx.spring_layout(G)
        nx.draw(G, pos, node_color='w', node_size=1, with_labels=False, ax=ax2)
        plt.show()

    if file_name != None:
        fig, (ax1, ax2) = plt.subplots(1, 2)
        fig.suptitle("Células de Voronoi e Rede para {} pontos espaciais␣
↪aleatórios".format(N), fontsize=15)
        scipy.spatial.voronoi_plot_2d(vor, ax1)
        pos = nx.spring_layout(G)
        nx.draw(G, pos, node_color='w', node_size=1, with_labels=False, ax=ax2)
        plt.savefig("images/"+file_name)
        plt.close(fig)

    return G


def spatial_network_radius(N: int,
                           box_size: int = 1,
                           radius = 0.3,
                           plot: bool = True,
                           file_name: str = None) -> nx.classes.graph.Graph:

    """
    Define uma rede em que as conexões entre os nós aleatoriamente distribuidos␣
↪no espaço
    são dadas pela distância menor ou igual a um raio definido.
    """

    G = nx.Graph()

    spatial_points = [SpatialPoint(i, box_size) for i in range(N)]
    points2d_aux = [point_arr.get_coordinates() for point_arr in spatial_points]
    points2d = []
    for point_arr_aux in points2d_aux:
        points2d.append(list(point_arr_aux[0]))
```

```
    # Para cada par de nós (i,j) a aresta criada se distância(i,j)<=radius
    edges = [(node_i.index, node_j.index) for node_i in spatial_points for␣
↪node_j in spatial_points if node_i.get_distance(node_j) > 0 and node_i.
↪get_distance(node_j) <= radius]
    for edge in list(edges):
        G.add_edge(list(edge)[0], list(edge)[1])

    # Para calcularmos medidas de distância precisaremos remover nós não␣
↪conectados
    # No trecho abaixo mantemos somente o maior componente conctado da rede.
    G = G.to_undirected()
    G.remove_edges_from(nx.selfloop_edges(G))
    Gcc = sorted(nx.connected_components(G), key=len, reverse=True)
    G = G.subgraph(Gcc[0])
    G = nx.convert_node_labels_to_integers(G, first_label=0)

    if plot:
        fig, (ax1, ax2) = plt.subplots(1, 2)
        plt.suptitle("{} pontos aleatórios conectados em uma rede se distância␣
↪entre si é <={}".format(N,radius))
        ax1.scatter([x[0] for x in points2d], [y[1] for y in points2d])
        pos = nx.spring_layout(G)
        nx.draw(G, pos, node_color='w', node_size=1, with_labels=False, ax=ax2)
        plt.show()

    if file_name != None:
        fig, (ax1, ax2) = plt.subplots(1, 2)
        plt.suptitle("{} pontos aleatórios conectados em uma rede se distância␣
↪entre si é <={}".format(N,radius))
        ax1.scatter([x[0] for x in points2d], [y[1] for y in points2d])
        pos = nx.spring_layout(G)
        nx.draw(G, pos, node_color='w', node_size=1, with_labels=False, ax=ax2)
        plt.savefig("images/"+file_name)
        plt.close(fig)

    return G

def spatial_network_waxman(N: int,
                           box_size: int = 1,
                           alpha = 0.3,
                           plot: bool = True,
                           file_name: str = None) -> nx.classes.graph.Graph:
    """
    Dedine uma rede em que os pontos espaciais aleatórios são conectados na␣
↪ocorrência de um
```

```
    evento aleatório de probabilidade regulada por um parâmetro alpha e da␣
↪distância.
    """
    G = nx.Graph()

    spatial_points = [SpatialPoint(i, box_size) for i in range(N)]
    points2d_aux = [point_arr.get_coordinates() for point_arr in spatial_points]
    points2d = []
    for point_arr_aux in points2d_aux:
        points2d.append(list(point_arr_aux[0]))


    # Os pontos (i,j) são conectados se um evento aleatório de probabilidade p␣
↪é < exp^(distância(i,j)/alpha)
    edges = [(node_i.index, node_j.index) for node_i in spatial_points for␣
↪node_j in spatial_points if random.random() < np.exp(-1*node_i.
↪get_distance(node_j)/alpha)]
    for edge in list(edges):
        G.add_edge(list(edge)[0], list(edge)[1])

    # Para calcularmos medidas de distância precisaremos remover nós não␣
↪conectados
    # No trecho abaixo mantemos somente o maior componente conctado da rede.
    G = G.to_undirected()
    G.remove_edges_from(nx.selfloop_edges(G))
    Gcc = sorted(nx.connected_components(G), key=len, reverse=True)
    G = G.subgraph(Gcc[0])
    G = nx.convert_node_labels_to_integers(G, first_label=0)

    if plot:
        fig, (ax1, ax2) = plt.subplots(1, 2)
        plt.suptitle("{} pontos aleatórios e a rede de Waxman com alpha={}".
↪format(N,alpha))
        ax1.scatter([x[0] for x in points2d], [y[1] for y in points2d])
        pos = nx.spring_layout(G)
        nx.draw(G, pos, node_color='w', node_size=1, with_labels=False, ax=ax2)
        plt.show()

    if file_name != None:
        fig, (ax1, ax2) = plt.subplots(1, 2)
        plt.suptitle("{} pontos aleatórios e a rede de Waxman com alpha={}".
↪format(N,alpha))
        ax1.scatter([x[0] for x in points2d], [y[1] for y in points2d])
        pos = nx.spring_layout(G)
        nx.draw(G, pos, node_color='w', node_size=1, with_labels=False, ax=ax2)
        plt.savefig("images/"+file_name)
```

```
        plt.close(fig)

    return G

def degree_distribution(G: nx.classes.graph.Graph) -> list:

    """
    Retorna a lista de valores de grau (k) para todos os nós da rede.
    """

    dict_degree = dict(G.degree())
    list_k = []
    for node, k_value in dict_degree.items():
        list_k.append(k_value)

    return list_k

def clustering_coef_distribution(G: nx.classes.graph.Graph) ->list:

    """
    Retorna a lista de valores de cluster coefficient (cc) para todos os nós da␣
 ↪rede.

    """

    list_cc_nodes = []
    for node in G.nodes():
        list_cc_nodes.append(nx.clustering(G, node))

    return list_cc_nodes

def spl_distribution(G: nx.classes.graph.Graph) ->list:
    """
    Retorna a lista de valores de shortest path length (spl) para todos os nós␣
 ↪da rede.

    """

    N = len(G)
    if nx.is_connected(G) == True:
        distance_matrix = np.zeros(shape=(N,N))
        diameter = nx.diameter(G)
        slp_values = []
        for i in np.arange(0,N):
            for j in np.arange(i+1, N):
                if(i != j):
                    aux = nx.shortest_path(G,i,j)
```

```
                    dij = len(aux)-1
                    distance_matrix[i][j] = dij
                    distance_matrix[j][i] = dij
                    slp_values.append(dij)
        return slp_values
    else:
        pass

def shannon_entropy(G: nx.classes.graph.Graph) ->float:

    """
    Calcula a entropia de Shannon para um grafo G recebido como parâmetro.
    """
    list_k = degree_distribution(G)
    min_k = np.min(list_k)
    max_k = np.max(list_k)

    k_values= np.arange(0,max_k+1)
    k_prob = np.zeros(max_k+1)
    for k in list_k:
        k_prob[k] = k_prob[k] + 1
    k_prob = k_prob/sum(k_prob)

    H = 0
    for p in k_prob:
        if(p > 0):
            H = H - p*math.log(p, 2)
    return H

def distribution_plot(list_values: list,
                      plot_title: str = "Histograma de densidade",
                      var_name: str = "Variável",
                      file_name: str = None) -> None:
    """
    Produz histgrama de uma medida recebida na forma de lista.
    """

    avg_value = np.mean(list_values)
    var_value = np.var(list_values)

    fig, ax = plt.subplots()
    n, bins, patches = ax.hist(list_values, density=True)
    ax.set_xlabel(var_name)
    ax.set_ylabel("Densidade de probabilidade")
    ax.set_title("{} de {}: média={:.2f}, var={:.2f}".format(plot_title,
                                                            var_name,
                                                            avg_value,
```

```
                                                                    var_value),
                                                                    fontsize=15)
    plt.show(True)
    if file_name != None:
        fig.savefig("images/"+file_name)

def correlation_plot(x: list,
                     y: list,
                     x_label: str = "x",
                     y_label: str = "y",
                     file_name: str = None) -> None:
    """
    Produz gráfico de dispersão de duas variáveis x e y recebidas na forma de␣
 ↪listas.
    Calcula correlação de Pearson e Spearman para x e y.
    """

    pearson_corr = np.corrcoef(x, y)[0,1]
    spearman_corr, spearman_pval = scipy.stats.spearmanr(x, y)

    fig, ax = plt.subplots()
    ax.scatter(x, y)
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    ax.set_title("Dispersão de {} e {}: Pearson: {:.2f}, Spearman: {:.2f}␣
 ↪(p-val: {:.3f})".format(x_label,
                                                                          ␣
 ↪           y_label,
                                                                          ␣
 ↪           pearson_corr,
                                                                          ␣
 ↪           spearman_corr,
                                                                          ␣
 ↪           spearman_pval),
                                                                          ␣
 ↪           fontsize=15)
    plt.show(True)
    if file_name != None:
        fig.savefig("images/"+file_name)

def simple_plot2d(x: list,
                  y: list,
                  x_label: str = "x",
                  y_label: str = "y",
                  file_name: str = None) -> None:
    """
```

```
    Produz gráfico simples com associação entre suas variáveis x e y recebidas␣
↪na forma de listas.
    """
    fig, ax = plt.subplots()
    ax.plot(x, y)
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    ax.set_title("Dispersão de {} e {}".format(x_label, y_label, fontsize=15))
    plt.show(True)
    if file_name != None:
        fig.savefig("images/"+file_name)
```

Vamos agora reproduzir alguns experimentos com as redes espaciais. Para conseguirmos visualizar as conexões da rede, usaremos diferentes valores de número de nós (10 50 100, 500).

[ ]:
```
# Voronoi
list_N = [10, 50, 100, 500, 1000]
network_model = "spatial_voronoi"

for _n in list_N:
    G = spatial_network_voronoi(_n, 1, True, "graph_{}_n{}.jpg".
↪format(network_model, str(_n)))
    degree_dist = degree_distribution(G)
    cluster_dist = clustering_coef_distribution(G)
    spl_dist = spl_distribution(G)
    distribution_plot(degree_dist, var_name="Grau",␣
↪file_name="degree_dist_{}_n{}.jpg".format(network_model, str(_n)))
    distribution_plot(cluster_dist, var_name="Cluster␣
↪Coefficient",file_name="cc_dist_{}_n{}.jpg".format(network_model, str(_n)))
    distribution_plot(spl_dist, var_name="SPL", file_name="spl_dist_{}_n{}.jpg".
↪format(network_model, str(_n)))

list_N = [50, 100, 250, 500, 1000]
list_mean_k = []
list_mean_cc = []
list_mean_spl = []
list_shannon_entropy = []
network_model = "spatial_voronoi"

for _n in list_N:
    G = spatial_network_voronoi(_n, 1, False)
    degree_dist = degree_distribution(G)
    cluster_dist = clustering_coef_distribution(G)
    spl_dist = spl_distribution(G)
    list_mean_k.append(np.mean(degree_dist))
    list_mean_cc.append(np.mean(cluster_dist))
```

```
    list_mean_spl.append(np.mean(spl_dist))
    list_shannon_entropy.append(shannon_entropy(G))

simple_plot2d(list_N, list_mean_k, "N", "Grau médio", "plot2d_n_p_mean_k_{}.
 ↪jpg".format(network_model))
simple_plot2d(list_N, list_mean_cc, "N", "Cluster Coef.", "plot2d_n_mean_cc_{}.
 ↪jpg".format(network_model))
simple_plot2d(list_N, list_mean_spl, "N", "SPL médio", "plot2d_n_mean_spl_{}.
 ↪jpg".format(network_model))
simple_plot2d(list_N, list_shannon_entropy, "N", "Entropia de Shannon",␣
 ↪"plot2d_n_shannon_{}.jpg".format(network_model))
```

Vejamos qual o efeito de aumento do número de nós nas medidas da rede espacial de Voronoi.

[ ]:
```
# Círculos de raio R
list_N = [10, 50, 100, 500, 1000]
list_radius = [0.05, 0.25, 0.5, 0.75, 1]
network_model = "spatial_radius"

for _n in list_N:
    for _radius in list_radius:
        G = spatial_network_radius(_n, 1, _radius, True, "graph_{}_n{}_radius{}.
 ↪jpg".format(network_model, str(_n), str(_radius)))
        degree_dist = degree_distribution(G)
        cluster_dist = clustering_coef_distribution(G)
        spl_dist = spl_distribution(G)
        distribution_plot(degree_dist, var_name="Grau",␣
 ↪file_name="degree_dist_{}_n{}_radius{}.jpg".format(network_model, str(_n),␣
 ↪str(_radius)))
        distribution_plot(cluster_dist, var_name="Cluster␣
 ↪Coefficient",file_name="cc_dist_{}_n{}_radius{}.jpg".format(network_model,␣
 ↪str(_n), str(_radius)))
        distribution_plot(spl_dist, var_name="SPL",␣
 ↪file_name="spl_dist_{}_n{}_radius{}.jpg".format(network_model, str(_n),␣
 ↪str(_radius)))


list_N = [500]
list_radius = [0.05, 0.25, 0.5, 0.75, 1]
list_mean_k = []
list_mean_cc = []
list_mean_spl = []
list_shannon_entropy = []

for _n in list_N:
    for _radius in list_radius:
        G = spatial_network_radius(_n, 1, _radius, False)
```

```
        degree_dist = degree_distribution(G)
        cluster_dist = clustering_coef_distribution(G)
        spl_dist = spl_distribution(G)
        list_mean_k.append(np.mean(degree_dist))
        list_mean_cc.append(np.mean(cluster_dist))
        list_mean_spl.append(np.mean(spl_dist))
        list_shannon_entropy.append(shannon_entropy(G))

simple_plot2d(list_radius, list_mean_k, "Raio", "Grau médio",␣
 ↪"plot2d_radius_mean_k_{}.jpg".format(network_model))
simple_plot2d(list_radius, list_mean_cc, "Raio", "Cluster Coef.",␣
 ↪"plot2d_radius_mean_cc_{}.jpg".format(network_model))
simple_plot2d(list_radius, list_mean_spl, "Raio", "SPL médio",␣
 ↪"plot2d_radius_mean_spl_{}.jpg".format(network_model))
simple_plot2d(list_radius, list_shannon_entropy, "Raio", "Entropia de Shannon",␣
 ↪"plot2d_radius_shannon_{}.jpg".format(network_model))
```

[ ]:
```
# Waxman
list_N = [10, 50, 100, 500, 1000]
list_alpha = [0.05, 0.25, 0.5, 0.75, 1]
network_model = "spatial_waxman"

for _n in list_N:
    for _alpha in list_alpha:
        G = spatial_network_waxman(_n, 1, _alpha, True, "graph_{}_n{}alpha{}.
 ↪jpg".format(network_model, str(_n), str(_alpha)))
        degree_dist = degree_distribution(G)
        cluster_dist = clustering_coef_distribution(G)
        spl_dist = spl_distribution(G)
        distribution_plot(degree_dist, var_name="Grau",␣
 ↪file_name="degree_dist_{}_n{}_alpha{}.jpg".format(network_model, str(_n),␣
 ↪str(_alpha)))
        distribution_plot(cluster_dist, var_name="Cluster␣
 ↪Coefficient",file_name="cc_dist_{}_n{}_alpha{}.jpg".format(network_model,␣
 ↪str(_n), str(_alpha)))
        distribution_plot(spl_dist, var_name="SPL",␣
 ↪file_name="spl_dist_{}_n{}_alpha{}.jpg".format(network_model, str(_n),␣
 ↪str(_alpha)))


list_N = [500]
list_alpha = [0.05, 0.25, 0.5, 0.75, 1]
list_mean_k = []
list_mean_cc = []
list_mean_spl = []
list_shannon_entropy = []
```

```
for _n in list_N:
    for _alpha in list_alpha:
        G = spatial_network_radius(_n, 1, _radius, False)
        degree_dist = degree_distribution(G)
        cluster_dist = clustering_coef_distribution(G)
        spl_dist = spl_distribution(G)
        list_mean_k.append(np.mean(degree_dist))
        list_mean_cc.append(np.mean(cluster_dist))
        list_mean_spl.append(np.mean(spl_dist))
        list_shannon_entropy.append(shannon_entropy(G))

simple_plot2d(list_alpha, list_mean_k, "alpha", "Grau médio",␣
 ↪"plot2d_alpha_mean_k_{}.jpg".format(network_model))
simple_plot2d(list_alpha, list_mean_cc, "alpha", "Cluster Coef.",␣
 ↪"plot2d_alpha_mean_cc_{}.jpg".format(network_model))
simple_plot2d(list_alpha, list_mean_spl, "alpha", "SPL médio",␣
 ↪"plot2d_radius_alpha_spl_{}.jpg".format(network_model))
simple_plot2d(list_alpha, list_shannon_entropy, "alpha", "Entropia de Shannon",␣
 ↪"plot2d_alpha_shannon_{}.jpg".format(network_model))
```