exercicio-05-giuliavieira

December 13, 2023

UNIVERSIDADE FEDERAL DE MINAS GERAIS INSTUTUTO DE CIÊNCIAS EXATAS GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DISCIPLINA: Introdução a Física Estatística e Computacional

ALUNA: Giulia Monteiro Silva Gomes Vieira MATRICULA: 2016006492

EXERCÍCIO AVALIATIVO 05: Redes Complexas

[1]:
```
import matplotlib.pyplot as plt
import networkx as nx
import rustworkx as rx
import numpy as np
from scipy.io import mmread
```

[3]:
```
def build_network(N, Z, p):
    G = nx.Graph()

    # Adicionar nós e arestas curtas
    for i in range(N):
        for j in range(1, Z // 2 + 1):
            G.add_edge(i, (i + j) % N)
            G.add_edge(i, (i - j) % N)

    # Adicionar aleatoriamente atalhos
    for i in range(N):
        for j in range(1, Z // 2 + 1):
            if np.random.rand() < p:
                G.add_edge(i, np.random.randint(0, N))

    return G
```

[4]:
```
def FindPathLengthsFromNode(graph, node):
    distances = {}  # Armazenar as distâncias mais curtas
    currentShell = [node]  # Inicializar a camada atual com o nó inicial
    d = 0  # Inicializar a distância como 0

    while currentShell:
        nextShell = []  # Inicializar a próxima camada
```

```
        for current_node in currentShell:
            if current_node not in distances:
                distances[current_node] = d  # Definir a distância do nó atual
                nextShell.extend(graph.neighbors(current_node))

        d += 1  # Incrementar a distância
        currentShell = nextShell  # Atualizar para a próxima camada

    return distances
```

[5]:
```
def FindAllPathLengths(graph):
    all_lengths = []

    for node in graph.nodes:
        distances = FindPathLengthsFromNode(graph, node)
        all_lengths.extend(list(distances.values()))

    return all_lengths
```

[32]:
```
def FindAveragePathLength(graph):
    total_length = 0
    num_pairs = 0

    for source in graph.nodes:
        distances = FindPathLengthsFromNode(graph, source)
        total_length += sum(distances.values())
        num_pairs += len(distances)

    return total_length / num_pairs
```

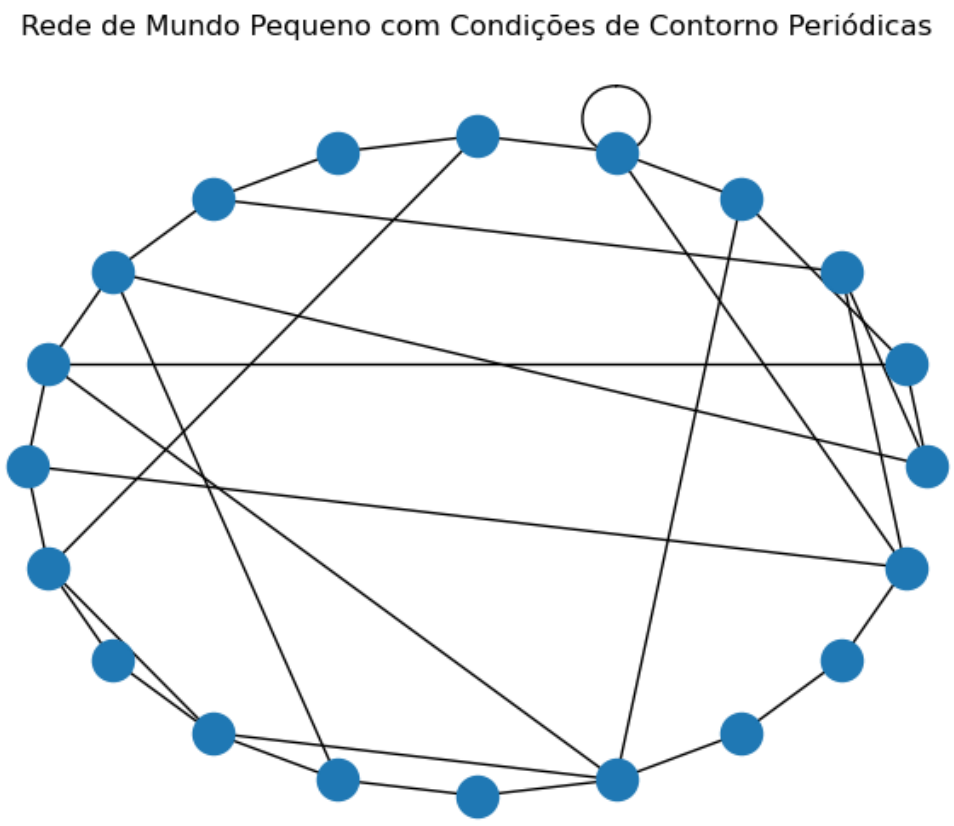
[8]:
```
def get_d_norm(graph, p_base=0):
    d_base = nx.average_shortest_path_length(build_network(N, Z, p_base))
    d_atual = nx.average_shortest_path_length(graph)
    return d_atual / d_base
```

[9]:
```
def get_redim_lengths(N, Z, p):
    while True:
        graph = nx.watts_strogatz_graph(N, Z, p)
        if nx.is_connected(graph):
            break

    d = nx.average_shortest_path_length(graph)
    delta_theta = np.pi * Z * d / N
    return delta_theta
```

01. Geração da rede mini mundo

[10]:
```
# Parâmetros
N = 20  # Número de nós
Z = 3   # Grau médio
p = 0.5 # Probabilidade de adicionar atalhos
```

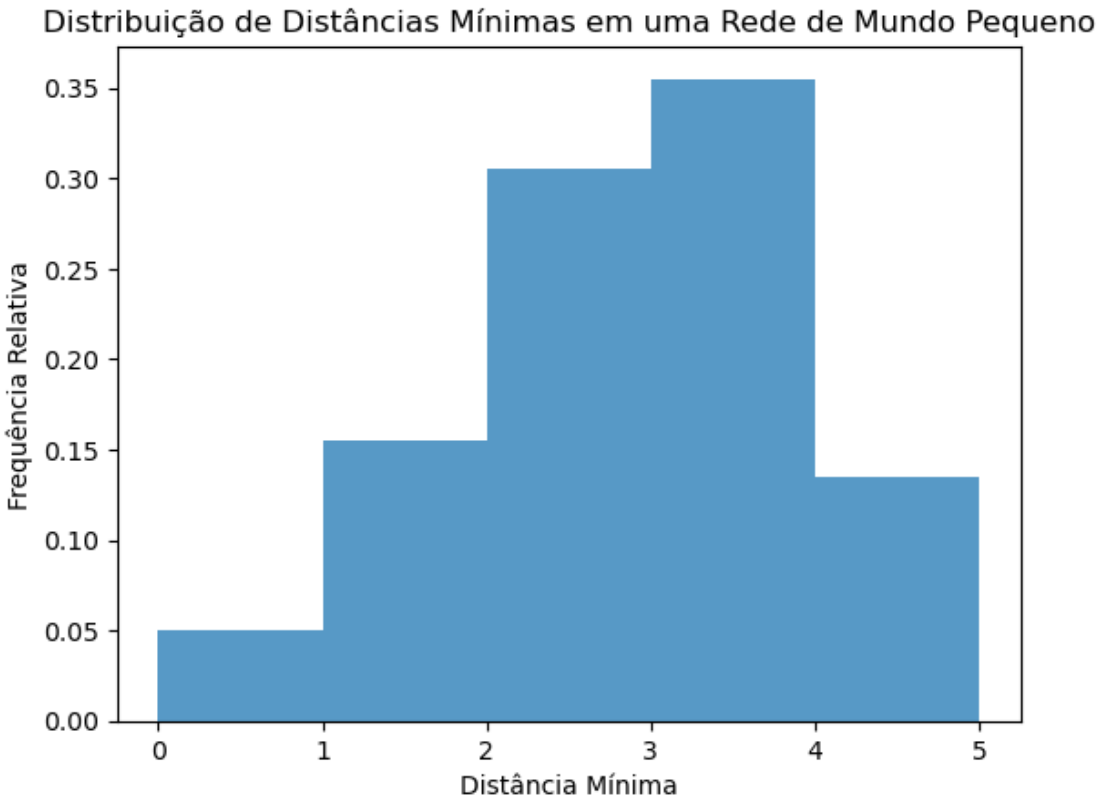
[11]:
```
# Construir a rede
G = build_network(N, Z, p)

# Desenhar o grafo de mundo pequeno
nx.draw_circular(G, font_weight='bold')
plt.title('Rede de Mundo Pequeno com Condições de Contorno Periódicas')
plt.show()
```



02. Distribuição das distâncias mínimas entre os nós

[12]:
```
caminhos_mais_curtos = dict(nx.all_pairs_shortest_path_length(G))

# Extrair distâncias mínimas para cada par de nós
distancias_minimas = [dist for node_dist in caminhos_mais_curtos.values() for␣
  ↪dist in node_dist.values()]
```

```
# Plotar histograma
plt.hist(distancias_minimas, bins=np.arange(0, max(distancias_minimas)+1, 1),␣
  ↪density=True, alpha=0.75)
plt.xlabel('Distância Mínima')
plt.ylabel('Frequência Relativa')
plt.title('Distribuição de Distâncias Mínimas em uma Rede de Mundo Pequeno')
plt.show()
```

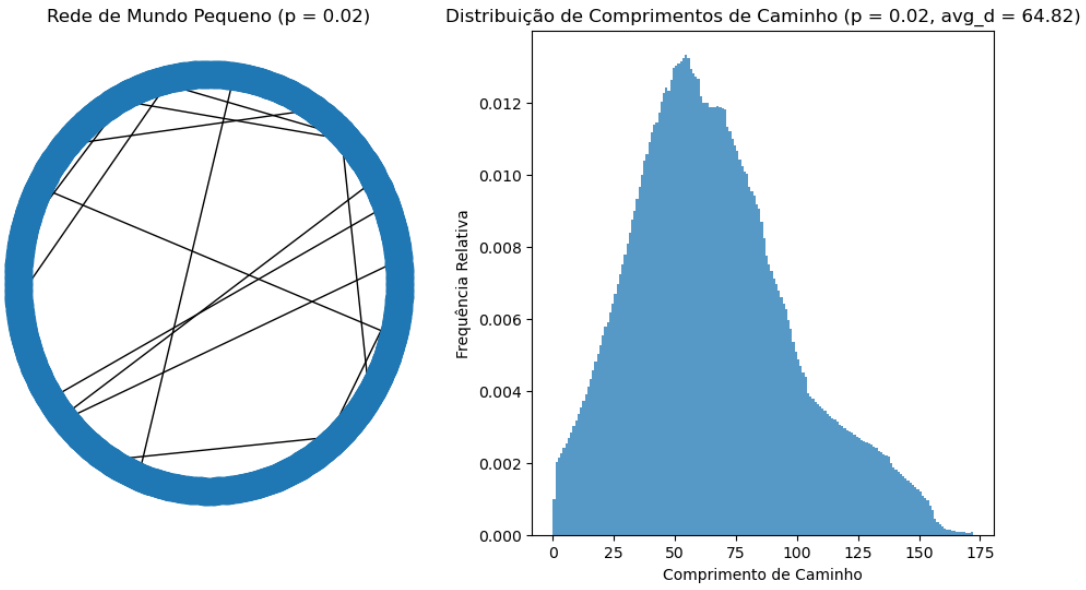


[13]:
```
N = 1000
Z = 2
```

[14]:
```
p = 0.02

# Construir o grafo
graph = build_network(N, Z, p)

# Calcular
d_avg = FindAveragePathLength(graph)

# Calcular os comprimentos de caminho
```

```
all_lengths = []
for node in graph.nodes:
    distances = FindPathLengthsFromNode(graph, node)
    all_lengths.extend(list(distances.values()))
```

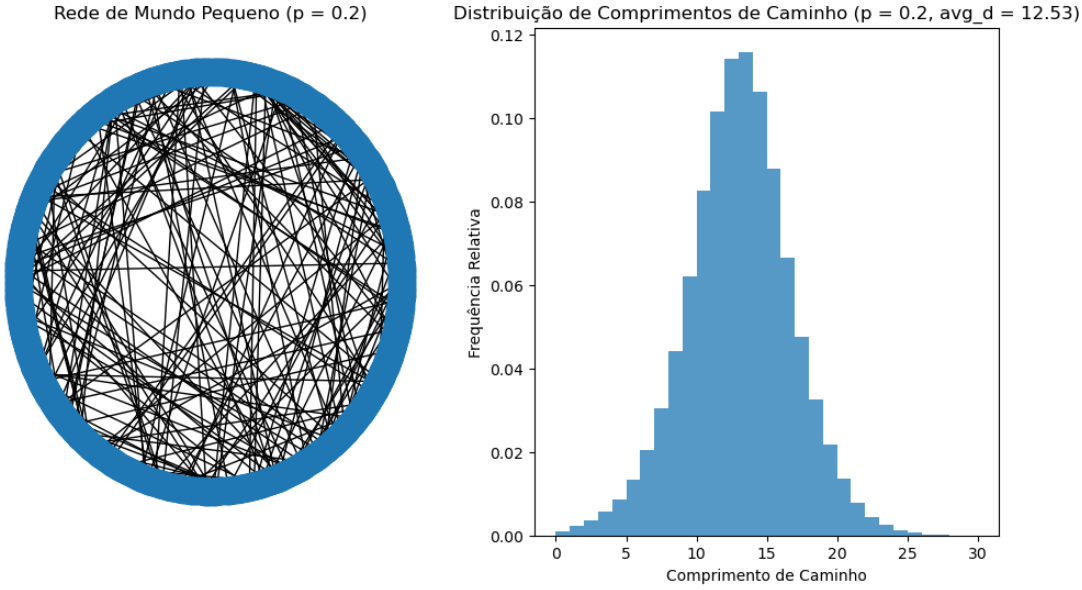
[15]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Rede
plt.subplot(1, 2, 1)
nx.draw_circular(graph)
plt.title(f'Rede de Mundo Pequeno (p = {p})')

# Histograma
plt.subplot(1, 2, 2)
plt.hist(all_lengths, bins=np.arange(0, max(all_lengths)+1, 1), density=True,␣
 ↪alpha=0.75)
plt.xlabel('Comprimento de Caminho')
plt.ylabel('Frequência Relativa')
plt.title(f'Distribuição de Comprimentos de Caminho (p = {p}, avg_d = {d_avg:.
 ↪2f})')

# Adjust layout to prevent overlap
#plt.tight_layout()
plt.show()
```



[16]:
```
p = 0.2
```

```
# Construir o grafo
graph = build_network(N, Z, p)

# Calcular
d_avg = FindAveragePathLength(graph)

# Calcular os comprimentos de caminho
all_lengths = []
for node in graph.nodes:
    distances = FindPathLengthsFromNode(graph, node)
    all_lengths.extend(list(distances.values()))
```

[17]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Rede
plt.subplot(1, 2, 1)
nx.draw_circular(graph)
plt.title(f'Rede de Mundo Pequeno (p = {p})')

# Histograma
plt.subplot(1, 2, 2)
plt.hist(all_lengths, bins=np.arange(0, max(all_lengths)+1, 1), density=True,␣
 ↪alpha=0.75)
plt.xlabel('Comprimento de Caminho')
plt.ylabel('Frequência Relativa')
plt.title(f'Distribuição de Comprimentos de Caminho (p = {p}, avg_d = {d_avg:.
 ↪2f})')

# Adjust layout to prevent overlap
#plt.tight_layout()
plt.show()
```



0.2.1. Análise dos histogramas: A distribuição das distâncias mínimas é dada por uma curva de bell. O valor p desloca essa curva no eixo x, de comprimento de caminho. Quando a distribuição normal se desloca desta maneira, isso significa que houve mudança na tendência central dos dados. O p de valor menor desloca a curva para a esquerda, deslocamento negativo. Representa diminuição da média. O p de valor maior desloca a curva para a direita, deslocamento positivo. Representa aumento da média. **02.2. Seis graus de separação** A ideia dos ”seis graus de separação” refere-se à hipótese de que, em uma rede social, é possível conectar duas pessoas escolhidas aleatoriamente por uma cadeia de conhecidos com, no máximo, seis etapas intermediárias.

No contexto de um modelo de mundo pequeno, o comprimento médio de caminho, , é frequentemente utilizado para medir a eficiência das conexões em um grafo. Quando é próximo de 6, diz-se que a rede exibe características de ”seis graus de separação”.

[18]:
```
N = 1000  # Número de nós
Z = 2     # Grau médio
target_avg_path_length = 6  # Alvo para

# Valor inicial de p
p = 0.01

# Tolerância para a diferença entre    e o alvo
tolerance = 0.1

while True:
    # Construir o grafo
    grafo = construir_rede_mundo_pequeno(N, Z, p)

    # Calcular
```

```
    avg_path_length = FindAveragePathLength(grafo)

    # Exibir o valor atual de p e
    #print(f'p = {p:.4f},    = {avg_path_length:.4f}')

    # Verificar se atingimos o alvo
    if abs(avg_path_length - target_avg_path_length) < tolerance:
        print(f'\nAlcançamos o alvo de    = {target_avg_path_length} com p = {p:
↪.4f}')
        break

    # Ajustar o valor de p para a próxima iteração
    p += 0.01
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[18], line 13
      9 tolerance = 0.1
     11 while True:
     12     # Construir o grafo
---> 13     grafo = construir_rede_mundo_pequeno(N, Z, p)
     15     # Calcular
     16     avg_path_length = FindAveragePathLength(grafo)

NameError: name 'construir_rede_mundo_pequeno' is not defined
```

02.3. Média do tamanho dos caminhos

[ ]:
```
# Parâmetros
N = 100  # Número de nós
Z = 2    # Grau médio
p = 0.1  # Probabilidade de reconfiguração

# Número de execuções
num_execucoes = 10

# Calcular    para diferentes execuções
resultados_d = []

for _ in range(num_execucoes):
    # Construir a rede
    grafo = build_network(N, Z, p)

    # Calcular
    d_medio = FindAveragePathLength(grafo)
    resultados_d.append(d_medio)
```

```
    # Exibir o valor de    para esta execução
    #print(f'Execução:    = {d_medio:.4f}')

# Exibir o valor médio de    e a variabilidade
media_d = np.mean(resultados_d)
desvio_padrao_d = np.std(resultados_d)

print(f'\nValor médio de   : {media_d:.4f}')
```

02.4. Arestas longas

[ ]:
```
N*p*Z/2
```

02.5. Flutuações nas distâncias É esperado observar flutuações nas distâncias médias entre pares de nós em diferentes execuções devido à natureza estocástica da construção da rede de mundo pequeno.

Essas flutuações são comuns em redes geradas aleatoriamente, e o desvio padrão das distâncias médias entre diferentes execuções fornece uma medida da variabilidade dessas flutuações. Quanto maior o desvio padrão, maior a variabilidade nas distâncias médias entre execuções.

[ ]:
```
print(f'Desvio padrão de   : {desvio_padrao_d:.4f}')
```

02.6. Comprimento médio entre os caminhos d(p), dividido por d(p=0)

[ ]:
```
# Parâmetros
N = 50    # Número de nós
Z = 2     # Grau médio

# Gerar valores de p igualmente espaçados em escala logarítmica
valores_p = np.logspace(-3, 3, num=100)

# Calcular comprimento médio do caminho normalizado para diferentes valores de p
d_norm_list = []

for p in valores_p:
    grafo = build_network(N, Z, p)
    d_norm = get_d_norm(grafo)
    d_norm_list.append(d_norm)

# Deslocar valores de p por um fator de 100
valores_p_deslocados = valores_p * 100
```

[ ]:
```
# Plotar o gráfico semi-log
plt.semilogx(valores_p_deslocados, resultados_d_norm, marker='o', linestyle='-')
plt.xlabel('p (deslocado por um fator de 100)')
plt.ylabel('Comprimento Médio do Caminho Normalizado ($d(p) / d(p=0)$)')
plt.title('Comprimento Médio do Caminho Normalizado em Função de p')
```

```
plt.grid(True)
plt.show()
```

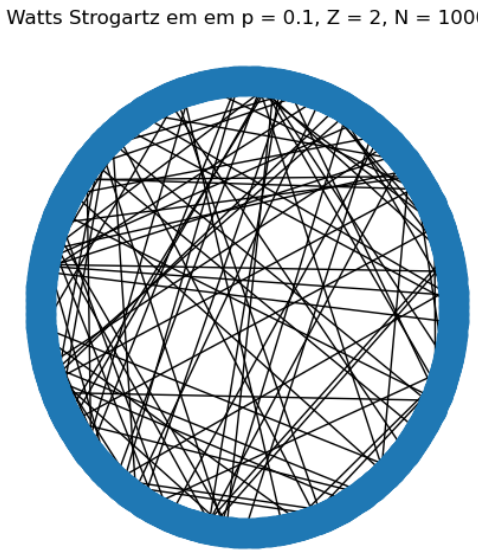
Quando p é muito pequeno, a probabilidade de reconfiguração é baixa, o que significa que a maioria das arestas permanece inalterada. Nesse caso, a estrutura da rede se aproxima de uma rede regular, onde os nós têm arestas que se conectam aos seus vizinhos mais próximos, formando um arranjo regular e ordenado.

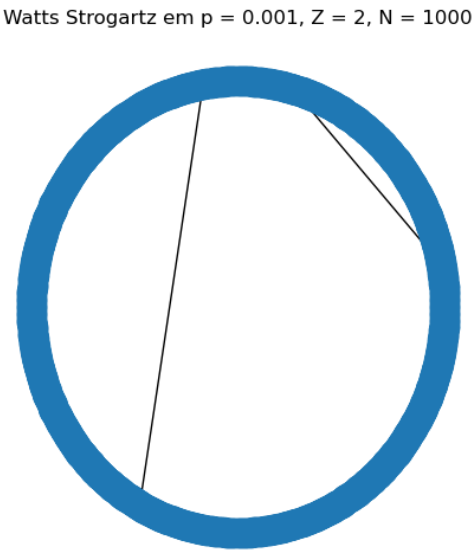
Nas redes regulares, o comprimento médio do caminho é relativamente grande em comparação com redes de mundo pequeno. Portanto, ao normalizar em relação a p=0 (onde a rede é regular), o comprimento médio do caminho será próximo a 1 para p pequeno.Essa geometria teria, então, um grafo circular como:

[ ]:
```
# Parâmetros
N = 50    # Número de nós
Z = 2     # Grau médio
p = 0.1

graph = build_network(N, Z, p)
nx.draw_circular(graph)
plt.title(f'Rede de Mundo Pequeno (p = {p})')
plt.show()
```

03.1. Grafo para geometria de Watts e Strogatz

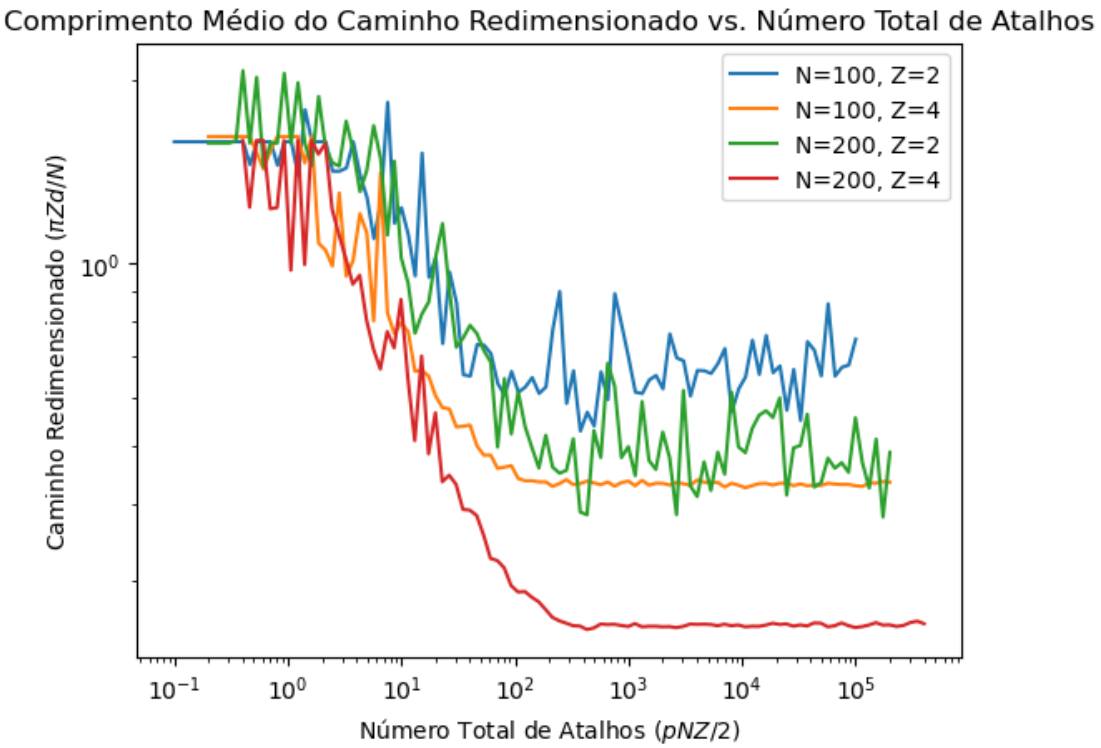
[19]:
```
p1 = 0.1
p2 = 0.001

graph1 = nx.watts_strogatz_graph(N, Z, p1)
graph2 = nx.watts_strogatz_graph(N, Z, p2)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Rede
plt.subplot(1, 2, 1)
nx.draw_circular(graph1)
plt.title(f'Watts Strogartz em em p = {p1}, Z = {Z}, N = {N}')

# Histograma
plt.subplot(1, 2, 2)
nx.draw_circular(graph2)
plt.title(f'Watts Strogartz em p = {p2}, Z = {Z}, N = {N}')

plt.show()
```





O grafo Watts Strogartz em p = 0.001 se parece mais com o grafo gerado pela rede mini mundo, o que faz sentido porque ambos compartilham todos os parâmetros de entrada, porque ambos contiinuam tendo conexões entre não vizinhos, mesmo que raras. **03.2. Comprimento médio do caminho redimensionado**

[20]:
```
N_vec = [100, 200]
Z_vec = [2, 4]
p_range = np.logspace(-3, 3, 100)

# Plotar o gráfico
for N in N_vec:
    for Z in Z_vec:
        redim_lengths = [get_redim_lengths(N, Z, p) for p in p_range]
        plt.loglog(p_range * N * Z / 2, redim_lengths, label=f'N={N}, Z={Z}')

plt.xlabel('Número Total de Atalhos ($pNZ/2$)')
plt.ylabel('Caminho Redimensionado ($\pi Z d / N$)')
plt.title('Comprimento Médio do Caminho Redimensionado vs. Número Total de␣
 ↪Atalhos')
plt.legend()
plt.show()
```



04. Redes reais @inproceedingsnr-aaai15, title = The Network Data Repository with Interactive Graph Analytics and Visualization, author=Ryan A. Rossi and Nesreen K. Ahmed, booktitle = AAAI, url=http://networkrepository.com, year=2015 "The graph "karate" contains the network of friendships between the 34 members of a karate club at a US university, as described by Wayne Zachary in 1977. If you use these data in your work, please cite W. W. Zachary, An information flow model for conflict and fission in small groups, Journal of Anthropological Research 33, 452-473 (1977)."

[21]:
```
karate_network = np.array(mmread("soc-karate/soc-karate.mtx").toarray())
```

[22]:
```
karate_network
```

[22]:
```
array([[0., 1., 1., …, 1., 0., 0.],
       [1., 0., 1., …, 0., 0., 0.],
       [1., 1., 0., …, 0., 1., 0.],
       …,
       [1., 0., 0., …, 0., 1., 1.],
       [0., 0., 1., …, 1., 0., 1.],
       [0., 0., 0., …, 1., 1., 0.]])
```

[23]:
```
graph = nx.DiGraph(karate_network)
```

04.1. Distância média

[35]:
```
avg_path = FindAveragePathLength(graph)
```

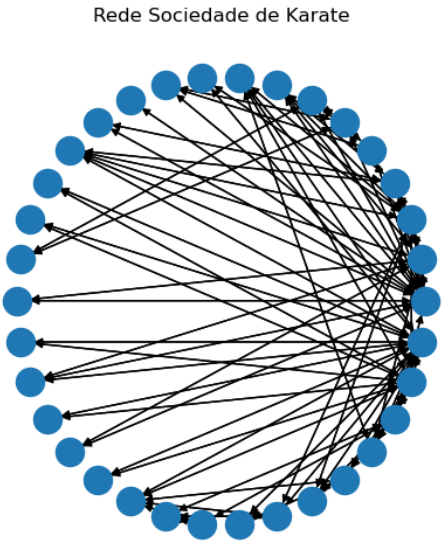
[36]:
```
avg_path
```

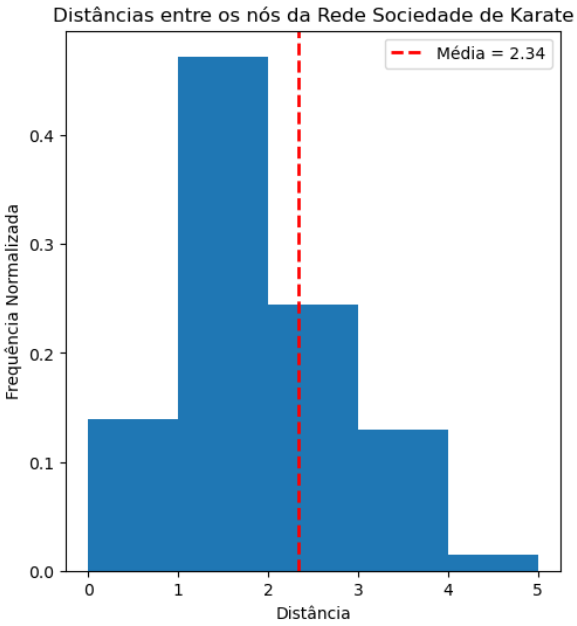
[36]:
```
2.337370242214533
```

[ ]:
```

```

[37]:
```
distances = [len(nx.shortest_path(graph, source, target)) - 1
             for source in graph.nodes()
             for target in graph.nodes() if source != target]
```

[40]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Rede
plt.subplot(1, 2, 1)
nx.draw_circular(graph)
plt.title(f'Rede Sociedade de Karate')

# Histograma
plt.subplot(1, 2, 2)
histogram, bin_edges = np.histogram(distances, bins=np.arange(min(distances),␣
 ↪max(distances)+2)-0.5, density=True)
plt.bar(bin_edges[:-1], histogram, width=1)
plt.axvline(average_path, color='red', linestyle='dashed', linewidth=2,␣
 ↪label=f'Média = {round(avg_path, 2)}')
plt.xlabel('Distância')
plt.ylabel('Frequência Normalizada')
plt.title('Distâncias entre os nós da Rede Sociedade de Karate')
plt.legend()
plt.show()
```


Rede Sociedade de Karate


Distâncias entre os nós da Rede Sociedade de Karate
Média = 2.34
Frequência Normalizada
0.0
0.1
0.2
0.3
0.4
0
1
2
3
4
5
Distância

[ ]:

[ ]: