CIC0199 - Teoria e Aplicação de Grafos - Projeto 1

Guilherme da Rocha Cunha - 221030007

2023-10-12

---

7. Components, Communities and Cliques

7.5 Learning exercises

7.5.2 Data exercises

For Exercises 4-10, load the `email_edgelist` and `email_vertices` data sets from the `onadata` package or download them from the internet. This data set represents a network of emails sent between members of a large research institution. The department of each member is included in the vertex data set. Create an undirected graph from this data.

```
# Bibliotecas e funções a serem utilizadas
import pandas as pd
import networkx as nx
from cdlib import algorithms
```

```
## Note: to be able to use all crisp methods, you need to install some additional packages:  {'graph_to
```

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.colors import ListedColormap, LinearSegmentedColormap
```

4. Determine the connected components of this network and reduce the network to its largest connected component. **Resposta:**

```
# Lê da planilha as arestas e vértices do grafo
tabela_arestas = pd.read_csv("https://ona-book.org/data/email_edgelist.csv")
tabela_vertices = pd.read_csv("https://ona-book.org/data/email_vertices.csv")

# Cria um grafo não-direcionado a partir das planilhas
grafo = nx.from_pandas_edgelist(
    tabela_arestas,
    source = "from",
    target = "to"
)
```

```python
# Obtem os componentes conexos do grafo
comp_conexos = nx.connected_components(grafo)

# Obtem os subgrafos a partir dos componentes conexos
subgrafos = [grafo.subgraph(componente).copy() for componente in comp_conexos]

# Inicializa o maior componente conexo e seu tamanho
maior_comp_conexo, maior_tam = None, 0

# Procura o maior componente conexo dentre todos os subgrafos
for subgrafo in subgrafos:
    if len(subgrafo) > maior_tam:
        maior_tam = len(subgrafo)
        maior_comp_conexo = subgrafo

print(f"Maior componente conexo do grafo: {maior_comp_conexo}")
```

```
## Maior componente conexo do grafo: Graph with 986 nodes and 16064 edges
```

5. Use the Louvain algorithm to determine a vertex partition/community structure with optimal modularity in this network. **Resposta:**

```python
# Usa o algoritmo de Louvain para determinar a estrutura de comunidade com modularidade ótima
comunidades_louvain = algorithms.louvain(grafo)
df_l = pd.DataFrame(comunidades_louvain.communities).transpose()
print(df_l)
```

```
##           0     1    2     3     4      5     6     7
## 0      10.0  14.0  2.0  13.0   7.0   60.0   0.0  15.0
## 1      16.0  41.0  3.0  23.0   8.0   61.0   1.0  45.0
## 2      20.0  51.0  4.0  24.0   9.0  103.0  17.0  46.0
## 3      21.0  53.0  5.0  25.0  11.0  104.0  18.0  97.0
## 4      22.0  64.0  6.0  26.0  12.0  122.0  52.0  98.0
## ..      ...   ...  ...   ...   ...    ...   ...   ...
## 241   969.0   NaN  NaN   NaN   NaN    NaN   NaN   NaN
## 242   984.0   NaN  NaN   NaN   NaN    NaN   NaN   NaN
## 243   989.0   NaN  NaN   NaN   NaN    NaN   NaN   NaN
## 244   999.0   NaN  NaN   NaN   NaN    NaN   NaN   NaN
## 245  1003.0   NaN  NaN   NaN   NaN    NaN   NaN   NaN
##
## [246 rows x 8 columns]
```

6. Compare the modularity of the Louvain community structure with that of the ground truth department structure. **Resposta:** Nota-se que estrutura de comunidade de Louvain é mais compacta e densa em comparação à estrutura de departamentos *ground truth*, onde por sua vez possui 42 subconjuntos (departamentos) com baixa modularidade, isto é, pouco densos.

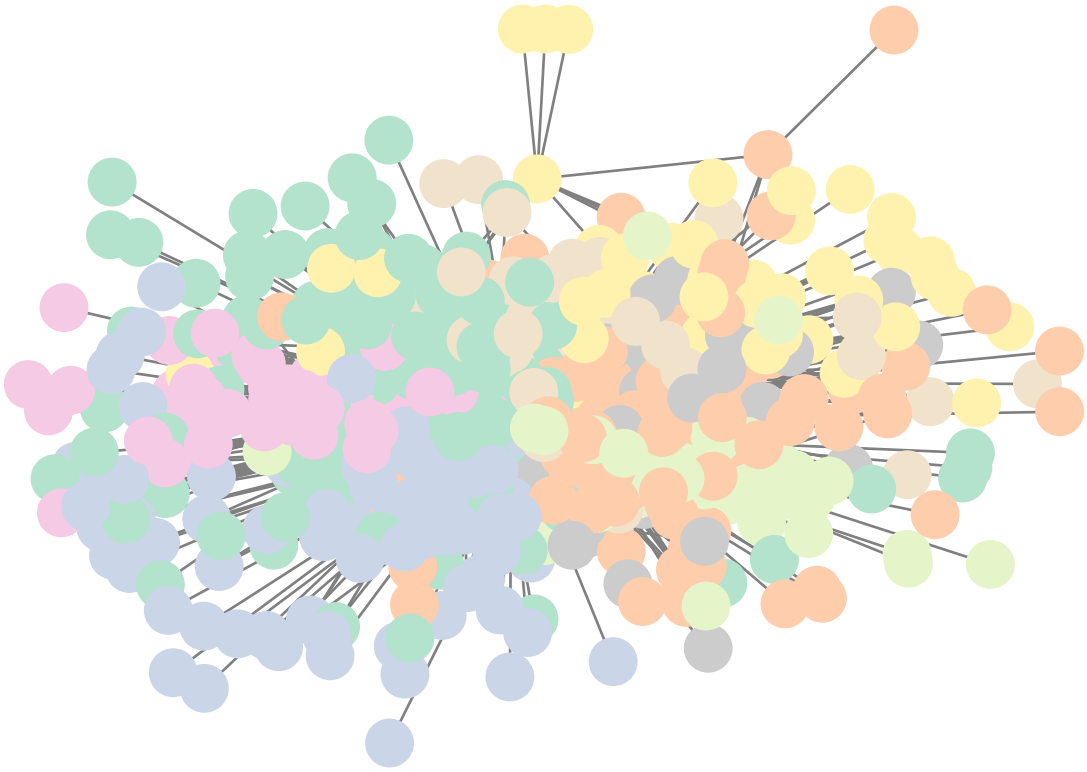
```python
departamentos = {}
dicio = {}
# Agrupa todos os vértices do grafo em seus respectivos departamentos
for i, row in tabela_vertices.iterrows():
```

```
    id = row["id"]
    dept = row["dept"]

    if dept not in departamentos.keys(): departamentos[dept] = [id]
    else: departamentos[dept].append(id)

    # Salva qual departamento o vertice pertence
    dicio[id] = dept

# Cria uma tabela de M colunas, onde M é o número de departamentos
matriz = []
for i in range(len(departamentos)): matriz.append([])
# Popula a matriz com seus respectivos valores
for dept, lista_id in departamentos.items():
    for id in lista_id:
        matriz[dept].append(id)

df_dept = pd.DataFrame(matriz).transpose()
print(df_dept)
```

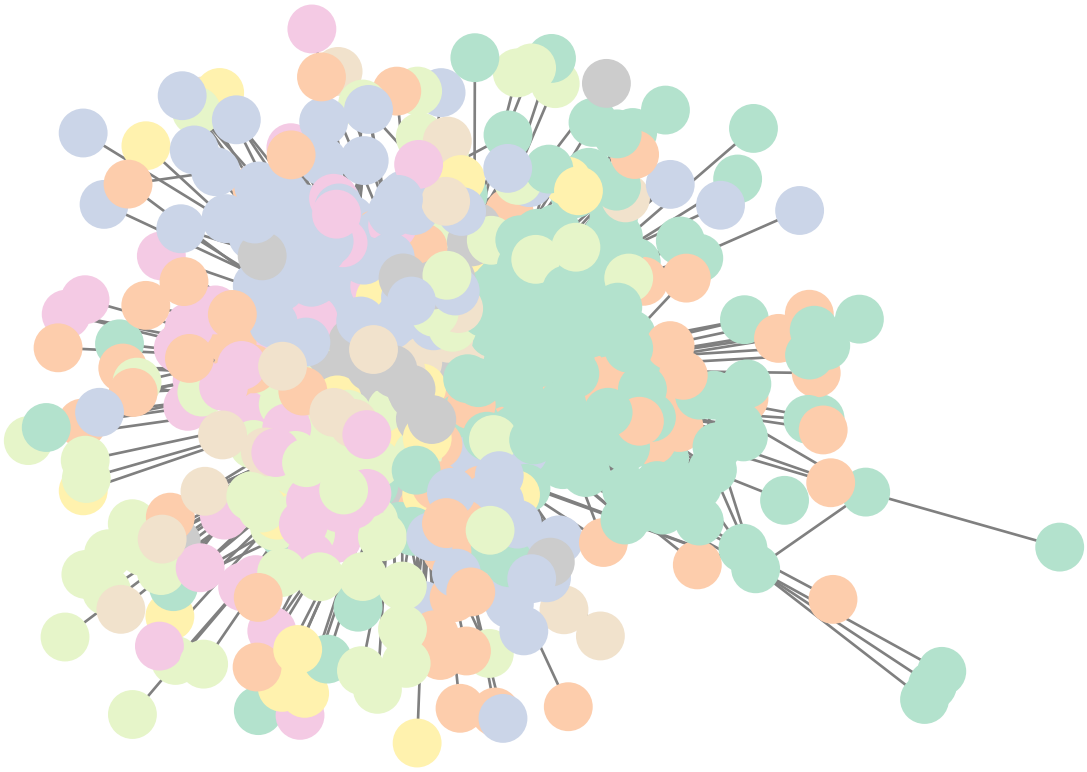
```
##         0     1      2      3       4   ...     37     38     39     40     41
## 0    122.0   0.0  134.0   77.0    14.0  ...   50.0  102.0  268.0  144.0  758.0
## 1    130.0   1.0  340.0   78.0    53.0  ...   70.0  106.0  331.0  186.0  941.0
## 2    148.0  17.0  482.0   79.0    65.0  ...   71.0  173.0  756.0  254.0    NaN
## 3    149.0  18.0  521.0  445.0    93.0  ...   84.0  300.0    NaN  492.0    NaN
## 4    156.0  73.0  553.0  483.0    95.0  ...  152.0  405.0    NaN    NaN    NaN
## ..     ...   ...    ...    ...     ...  ...    ...    ...    ...    ...    ...
## 104    NaN   NaN    NaN    NaN   959.0  ...    NaN    NaN    NaN    NaN    NaN
## 105    NaN   NaN    NaN    NaN   961.0  ...    NaN    NaN    NaN    NaN    NaN
## 106    NaN   NaN    NaN    NaN   965.0  ...    NaN    NaN    NaN    NaN    NaN
## 107    NaN   NaN    NaN    NaN   992.0  ...    NaN    NaN    NaN    NaN    NaN
## 108    NaN   NaN    NaN    NaN  1000.0  ...    NaN    NaN    NaN    NaN    NaN
##
## [109 rows x 42 columns]
```

7. Visualize the graph color-coded by the Louvain community, and then visualize the graph separately color-coded by the ground truth department. Compare the visualizations. Can you describe any of the Louvain communities in terms of departments? **Resposta:** Comparando as duas visualizações dos grafos, não é possivel descrever nenhuma comunidade de Louvain, devido à natureza não-ordenada dos vértices e seus vizinhos em relação aos seus respectivos departamentos.

```
# Visualização do grafo color-coded das comunidades de Louvain
comunidades = comunidades_louvain.to_node_community_map()
comunidades = [comunidades[v].pop() for v in grafo]

cor = cm.get_cmap("Pastel2", max(comunidades)+1)

np.random.seed(123)
nx.draw_spring(grafo, cmap = cor, node_color = comunidades, edge_color = "gray")
plt.show()
```



```
# Visualização do grafo color-coded dos departamentos "ground truth"
depts = [dicio[v] for v in grafo]

cor = cm.get_cmap("Pastel2", len(departamentos)+1)

nx.draw_spring(grafo, cmap = cor, node_color = depts, edge_color = "gray")
plt.show()
```



8. Create a dataframe containing the community and department for each vertex. Manipulate this dataframe to show the percentage of individuals from each department in each community. Try to visualize this using a heatmap or other style of visualization and try to use this to describe the communities in terms of departments. **Resposta:**

```
# Criação da tabela que indica a comunidade e departamente de cada vértice do grafo
matriz = []
com = comunidades_louvain.to_node_community_map()
for no in grafo:
    matriz.append([com[no].pop(), dicio[no]])

df_com_dept = pd.DataFrame(matriz, columns = ["Comunidade","Departamento"])
print(df_com_dept)
```

```
##      Comunidade  Departamento
## 0             6             1
## 1             6             1
## 2             2            21
## 3             2            21
## 4             2            21
## ..          ...           ...
## 981           1             4
## 982           2            21
## 983           6             1
## 984           0             6
```

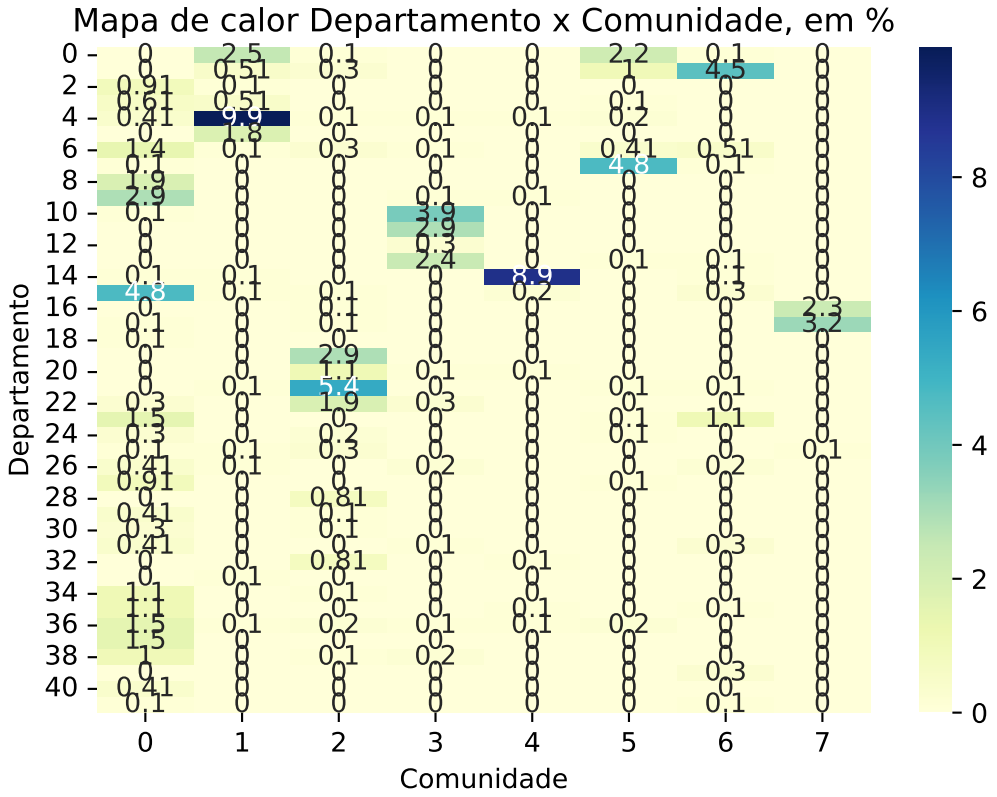
```
## 985           2            22
##
## [986 rows x 2 columns]
```

```python
# Criação da tabela que indica a porcentagem de cada departamento em cada comunidade
matriz = []
for i in range(df_l.shape[1]):
    matriz.append([0.0] * df_dept.shape[1])

# Leitura da tabela para calcular a porcentagem
for i,row in df_com_dept.iterrows():
    c = row["Comunidade"]
    d = row["Departamento"]
    matriz[c][d] += (1/df_com_dept.shape[0])*100

df_porcentagem = pd.DataFrame(matriz).transpose()
print(df_porcentagem)
```

```
##            0         1         2  ...         5         6         7
## 0   0.000000  2.535497  0.101420  ...  2.231237  0.101420  0.000000
## 1   0.000000  0.507099  0.304260  ...  1.014199  4.462475  0.000000
## 2   0.912779  0.101420  0.000000  ...  0.000000  0.000000  0.000000
## 3   0.608519  0.507099  0.000000  ...  0.101420  0.000000  0.000000
## 4   0.405680  9.939148  0.101420  ...  0.202840  0.000000  0.000000
## 5   0.000000  1.825558  0.000000  ...  0.000000  0.000000  0.000000
## 6   1.419878  0.101420  0.304260  ...  0.405680  0.507099  0.000000
## 7   0.101420  0.000000  0.000000  ...  4.766734  0.101420  0.000000
## 8   1.926978  0.000000  0.000000  ...  0.000000  0.000000  0.000000
## 9   2.941176  0.000000  0.000000  ...  0.000000  0.000000  0.000000
## 10  0.101420  0.000000  0.000000  ...  0.000000  0.000000  0.000000
## 11  0.000000  0.000000  0.000000  ...  0.000000  0.000000  0.000000
## 12  0.000000  0.000000  0.000000  ...  0.000000  0.000000  0.000000
## 13  0.000000  0.000000  0.000000  ...  0.101420  0.101420  0.000000
## 14  0.101420  0.101420  0.000000  ...  0.000000  0.101420  0.000000
## 15  4.766734  0.101420  0.101420  ...  0.000000  0.304260  0.000000
## 16  0.000000  0.000000  0.101420  ...  0.000000  0.000000  2.332657
## 17  0.101420  0.000000  0.101420  ...  0.000000  0.000000  3.245436
## 18  0.101420  0.000000  0.000000  ...  0.000000  0.000000  0.000000
## 19  0.000000  0.000000  2.941176  ...  0.000000  0.000000  0.000000
## 20  0.000000  0.000000  1.115619  ...  0.000000  0.000000  0.000000
## 21  0.000000  0.101420  5.375254  ...  0.101420  0.101420  0.000000
## 22  0.304260  0.000000  1.926978  ...  0.000000  0.000000  0.000000
## 23  1.521298  0.000000  0.000000  ...  0.101420  1.115619  0.000000
## 24  0.304260  0.000000  0.202840  ...  0.101420  0.000000  0.000000
## 25  0.101420  0.101420  0.304260  ...  0.000000  0.000000  0.101420
## 26  0.405680  0.101420  0.000000  ...  0.000000  0.202840  0.000000
## 27  0.912779  0.000000  0.000000  ...  0.101420  0.000000  0.000000
## 28  0.000000  0.000000  0.811359  ...  0.000000  0.000000  0.000000
## 29  0.405680  0.000000  0.101420  ...  0.000000  0.000000  0.000000
## 30  0.304260  0.000000  0.101420  ...  0.000000  0.000000  0.000000
## 31  0.405680  0.000000  0.000000  ...  0.000000  0.304260  0.000000
## 32  0.000000  0.000000  0.811359  ...  0.000000  0.000000  0.000000
## 33  0.000000  0.101420  0.000000  ...  0.000000  0.000000  0.000000
## 34  1.115619  0.000000  0.101420  ...  0.000000  0.000000  0.000000
```

```
## 35  1.115619  0.000000  0.000000  ...  0.000000  0.101420  0.000000
## 36  1.521298  0.101420  0.202840  ...  0.202840  0.000000  0.000000
## 37  1.521298  0.000000  0.000000  ...  0.000000  0.000000  0.000000
## 38  1.014199  0.000000  0.101420  ...  0.000000  0.000000  0.000000
## 39  0.000000  0.000000  0.000000  ...  0.000000  0.304260  0.000000
## 40  0.405680  0.000000  0.000000  ...  0.000000  0.000000  0.000000
## 41  0.101420  0.000000  0.000000  ...  0.000000  0.101420  0.000000
##
## [42 rows x 8 columns]
```

```python
# Visualização da tabela usando um heatmap
import seaborn as sns
sns.heatmap(df_porcentagem, annot = True, cmap = "YlGnBu")
plt.title("Mapa de calor Departamento x Comunidade, em %")
plt.xlabel("Comunidade")
plt.ylabel("Departamento")
plt.show()
```



9. Find the largest clique size in the graph. How many such largest cliques are there? What do you think a clique represents in this context? **Resposta:** Nesse contexto, os cliques podem representar os departamentos presentes no grafo.

```python
# Encontra o clique máximo do grafo
max_clique = nx.graph_clique_number(grafo)
print(f"Clique máximo: {max_clique}")
```

```
## Clique máximo: 18
```

```
# Obtém todos os cliques do grafo
cliques = nx.find_cliques(grafo)
cliques_maximais = sorted(cliques, key = len, reverse = True)

# Procura pelos cliques de tamanho do clique máximo
num_max_clique = 0
for clique in cliques_maximais:
    if len(clique) == max_clique: num_max_clique += 1
print(f"Quantidade de cliques de tamanho {max_clique}: {num_max_clique}")
```

```
## Quantidade de cliques de tamanho 18: 56
```

10. Try to visualize the members of these cliques in the context of the entire graph. What can you conclude?
Resposta: Ao analisar a visualização dos cliques, percebe-se que os cliques nesse contexto parecem descrever o quão distante os vértices do "centro" do grafo/ponto de referência.

```
# Para cada clique do grafo, é dado um identificador para cada vértice pertencente aquele clique
cliques = {}
id_clique = 0
for clique in cliques_maximais:
    add = False
    for v in clique:
        if v not in cliques:
            cliques[v] = id_clique
            add = True

    if add: id_clique += 1

# Visualização dos cliques do grafo
clq = [cliques[v] for v in grafo]

cor = cm.get_cmap("Pastel2", id_clique+1)

nx.draw_spring(grafo, cmap = cor, node_color = clq, edge_color = "gray")
plt.show()
```