Heidelberg University
Institute of Computer Science
Database Systems Research Group

Lecture: Complex Network Analysis

Prof. Dr. Michael Gertz

Assignment 8
Clustering and Modularity

`https://github.com/nilskre/CNA_assignments`

Team Member: Patrick Günther, 3660886,
Applied Computer Science
rh269@stud.uni-heidelberg.de

Team Member: Felix Hausberger, 3661293,
Applied Computer Science
eb260@stud.uni-heidelberg.de

Team Member: Nils Krehl, 3664130,
Applied Computer Science
pu268@stud.uni-heidelberg.de

Problem 8-1 European Power Grid

January 17, 2022

1 Lecture: Complex Network Analysis

Prof. Dr. Michael Gertz

Winter Semester 2021/22

1.1 Assignment 8 - Clustering and Modularity

1.1.1 Problem 8-1: European Power Grid

Students: Felix Hausberger, Nils Krehl, Patrick Günther

[56]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
```

[4]:
```
df = pd.read_csv('gridkit_europe.csv')
df.drop(columns=[col for col in df if col not in ["v_id_1", "v_id_2"]],␣
 ↪inplace=True)
df.head()
```

[4]:

	v_id_1	v_id_2
0	43193	23620
1	42022	13686
2	6913	48526
3	35422	28973
4	7864	63104

[16]:
```
# since it is an undirected graph, no parallel edges are added
G = nx.Graph()
G.add_edges_from(df.itertuples(index=False))

# remove self-loops
G.remove_edges_from(nx.selfloop_edges(G))

# stats
print(f"N = {G.number_of_nodes()}")
print(f"N = {G.number_of_edges()}")
```

```
N = 13844
N = 17277
```

2 1.

[24]:
```
def nth_moment(G, n):
    degrees = np.array(list(dict(G.degree).values()))
    return (sum(degrees**n)/len(G))

K = nth_moment(G, 2) / nth_moment(G, 1)
print(f"Molloy-Reed criterion: K = {np.round(K, 2)}")
```

```
Molloy-Reed criterion: K = 3.16
```

Since K is bigger than 2, the european power grid network should have a giant component. This makes complete sense in the case of a power grid network, since there shouldn't be fragmented infrastructure for power supply.

3 2.

[75]:
```
def size_largest_component(G):
    if G.number_of_nodes() > 0:
        return len(max(nx.connected_components(G), key=len))
    return 0

def average_size_components(G):
    if G.number_of_nodes() > 0:
        component_sizes = [len(component) for component in sorted(nx.
 ↪connected_components(G), key=len, reverse=True)]
        return sum(component_sizes) / len(component_sizes)
    return 0

print(f"Absolute size of largest component: {size_largest_component(G)}")
print(f"Relative size of largest component: {np.round(size_largest_component(G) /
 ↪ G.number_of_nodes(), 2)}")
```

```
Absolute size of largest component: 13478
Relative size of largest component: 0.97
```

Yes, the Molloy-Reed criterion gives the correct prediction as 97% of the nodes are inside a giant component.

4 3.

[53]:
```
print(f"f_c: {np.round(1 - (1 / (K - 1)), 2)}")
print(f"f_c_er: {np.round(1 - (1 / K), 2)}")
```

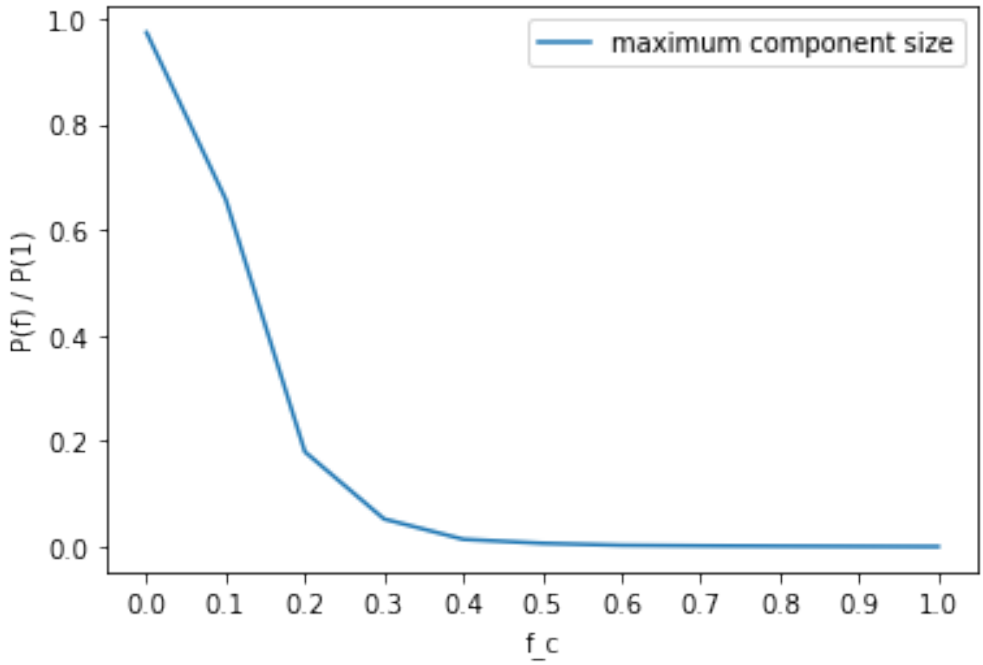
```
f_c: 0.54
f_c_er: 0.68
```

As $f_c < f_c^{ER}$ the european power grid network does not own enhanced robustness against random failures as its critical breakdown threshold is lower than expected for a random network with the same degree distribution.

5 4.

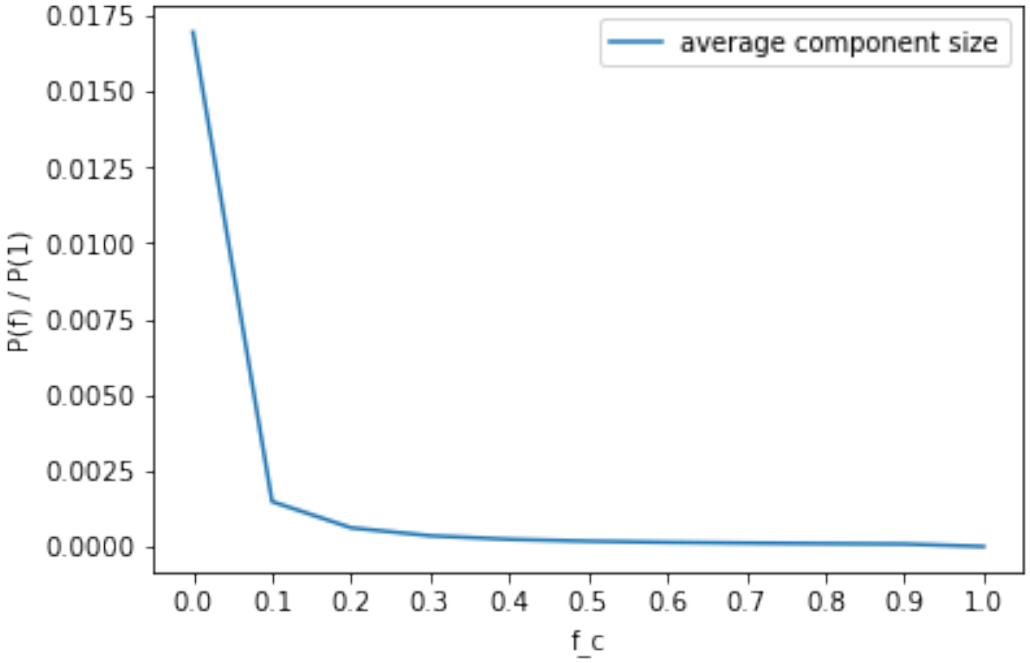
[71]:
```
def sample_random_failures(G, num_samples=10):
    N = G.number_of_nodes()

    averaged_largest_component_size = []
    averaged_average_component_size = []
    for f_c in np.arange(0, 1.1, 0.1):
        largest_component_size = []
        average_component_size = []
        for sample in range(num_samples):
            G_fc = G.copy()
            number_to_be_removed = int(np.round(f_c * N))
            nodes_to_be_removed = np.random.choice(G_fc.nodes(),␣
↪number_to_be_removed, replace=False)
            G_fc.remove_nodes_from(nodes_to_be_removed)
            largest_component_size.append(size_largest_component(G_fc))
            average_component_size.append(average_size_components(G_fc))

        averaged_largest_component_size.append(np.mean(np.
↪array(largest_component_size)))
        averaged_average_component_size.append(np.mean(np.
↪array(average_component_size)))
    return np.array(averaged_largest_component_size), np.
↪array(averaged_average_component_size)

max_component_sizes, average_component_sizes = sample_random_failures(G)
```

[72]:
```
N = G.number_of_nodes()
plt.plot(max_component_sizes / N, label="maximum component size")
plt.xticks(ticks=range(11), labels=(np.arange(0, 11, 1) / 10))
plt.legend()
plt.xlabel("f_c")
plt.ylabel("P(f) / P(1)")
plt.show()
```



One recognizes that at $f_c = 0.54$ the size of the largest component converges to 1 as predicted. This means for a power grid to lose its complete functionality to supply power around 50% of nodes need to randomly fail to achieve a collapse of the network.

[74]:
```
N = G.number_of_nodes()
plt.plot(average_component_sizes / N, label="average component size")
plt.xticks(ticks=range(11), labels=(np.arange(0, 11, 1) / 10))
plt.legend()
plt.xlabel("f_c")
plt.ylabel("P(f) / P(1)")
plt.show()
```



Already when 10% of all nodes randomly fail, one recognizes that the size of the largest component has drastically decreased and that also the average component size begins to converge to 1.

Problem 8-2 Hierarchical Clustering

For hierarchical clustering, we need to define the similarity of two disjoint clusters $X = \{x_1,\dots x_i\} \subset V$ and $Y = \{y_1,\dots,y_j\} \subset V$, based on the similarity of two nodes $s(v_i,v_j)$ with $v_i,v_j \in V$.

Popular linkage strategies for agglomerative hierarchical clustering include

	Similarity $s(X,Y)$	Distance $d(X,Y)$
Single-linkage clustering	$\max_{x\in X,y\in Y} s(x,y)$	$\min_{x\in X,y\in Y} d(x,y)$
Complete-linkage clustering	$\min_{x\in X,y\in Y} s(x,y)$	$\max_{x\in X,y\in Y} d(x,y)$

Given disjoint clusters X, X', and Y, we can alternatively compute the similarity $s(X \cup X',Y)$ using

	Similarity $s(X \cup X',Y)$	Distance $d(X \cup X',Y)$
Single-linkage clustering	$\max\{s(X,Y),s(X',Y)\}$	$\min\{d(X,Y),d(X',Y)\}$
Complete-linkage clustering	$\min\{s(X,Y),s(X',Y)\}$	$\max\{d(X,Y),d(X',Y)\}$

1. Discuss why Single-linkage and Complete-linkage swap their definitions (i.e. *min* and *max*) when used with a similarity rather than a distance function.

   After the merge process of nodes into communities, there are three approaches for calculating the new similarities: single-linkage (take the maximum), complete-linkage (take the minimum) and average-linkage. If the distance between two nodes is small (min), the similarity is high (max). And vice versa if the distance between two nodes is high (max), the similarity is low (min). Complete-linkage uses the maximum distance (which is the minimum similarity). Single-linkage uses the maximum similarity (which is the minimum distance).

2. Given the following similarity matrix x^o_{ij} , based on the topological overlap (cf. slide 9-22), perform agglomerative hierarchical clustering, using single-linkage.

$$x^o_{ij} = \begin{array}{c} \begin{matrix} A & B & C & D & E & F \end{matrix} \\ \begin{pmatrix} 0 & 1 & 1/2 & 1 & 1 & 0 \\ 1 & 0 & 1/3 & 1 & 1 & 0 \\ 1/2 & 1/3 & 0 & 1/3 & 1/3 & 1 \\ 1 & 1 & 1/3 & 0 & 1 & 0 \\ 1 & 1 & 1/3 & 1 & 0 & 0 \\ 1/2 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \end{matrix} \tag{1}$$

   Recall that for hierarchical clustering, you need to:

   - Find the maximum similarity $s(i,j)$,

- Merge the corresponding clusters,
- Update the similarity matrix by merging rows/columns i and j according to the linking strategy.

Begin by assigning each node to a separate cluster (each node is its own cluster), and repeat the above procedure until only one cluster remains. Write out the similarity matrix after each iteration and draw the resulting dendrogram. During each iteration, multiple sets may have the same maximum similarity. In this case, you may still only merge two clusters at a time. In this case, also prefer the merge of two smaller clusters over a bigger one, i.e., prefer to merge two clusters with cardinality 1 over the merge of clusters with size 2 and size 1, respectively.

---

single-linking strategy: take the minimum value

1. Merge A and B (maximum similarity 1)

$$x_{ij}^o = \begin{array}{c} \begin{array}{ccccc} A,B & C & D & E & F \end{array} \\ \begin{pmatrix} 0 & 1/3 & 1 & 1 & 0 \\ 1/3 & 0 & 1/3 & 1/3 & 1 \\ 1 & 1/3 & 0 & 1 & 0 \\ 1 & 1/3 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{array}{c} A,B \\ C \\ D \\ E \\ F \end{array} \tag{2}$$

Cluster: (A,B),C,D,E,F

2. Merge C and F (maximum similarity 1)

$$x_{ij}^o = \begin{array}{c} \begin{array}{cccc} A,B & C,F & D & E \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{array} \begin{array}{c} A,B \\ C,F \\ D \\ E \end{array} \tag{3}$$

Cluster: (A,B),(C,F),D,E

3. Merge D and E (maximum similarity 1)

$$x_{ij}^o = \begin{array}{c} \begin{array}{ccc} A,B & C,F & D,E \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \end{array} \begin{array}{c} A,B \\ C,F \\ D,E \end{array} \tag{4}$$

Cluster: (A,B),(C,F),(D,E)

4. Merge (A,B) and (D,E) (maximum similarity 1)

$$x_{ij}^{o} = \begin{array}{c} \begin{matrix} A,B,D,E & C,F \end{matrix} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{matrix} A,B,D,E \\ C,F \end{matrix} \end{array} \tag{5}$$

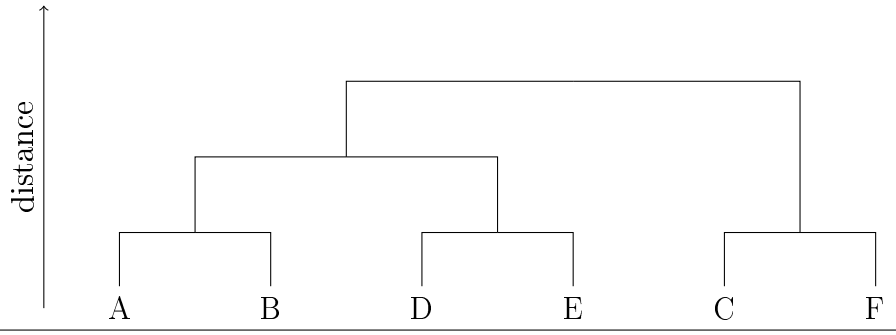
Cluster: (A,B,D,E),(C,F)

5. Merge (A,B,D,E) and (C,F) (maximum similarity 0)

results in one cluster containing all nodes

Cluster: (A,B,D,E,C,F)

Dendrogram:



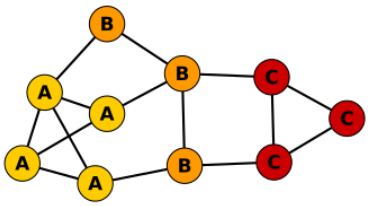
Problem 8-3 Modularity

Consider a simple, undirected graph G with L links between n nodes that are partitioned into n_c disjoint communities C. Let L_c be the number of links inside a given community c. Furthermore, let k_c be the sum of all degrees of nodes in community c (including edges that leave the community). Then the modularity M is defined as

$$M(G,C) := \sum_{c=1}^{n_c} (\frac{L_c}{L} - (\frac{k_c}{2L})^2) \tag{6}$$

1. Compute the modularity of the graph.

- $L = 15$
- $n = 10$
- $n_c = 3$



- $L_A = 5; L_B = 2; L_C = 3$
- $k_A = 3+3+3+4 = 13; k_B = 2+4+3 = 9; k_C = 3+2+3 = 8$

Plug the values into the given formula:

$$M(G,C) := (\frac{5}{15} - (\frac{13}{2*15})^2) + (\frac{2}{15} - (\frac{9}{2*15})^2) + (\frac{3}{15} - (\frac{8}{2*15})^2) = 0.318 \quad (7)$$

2. Give proof that for every partitioning of every simple graph, it always holds that $M \leq 1$.

For each component, its links within itself can be at most $L_c = L$. In this case, the component would hold all the links of the graph. For this component $\frac{L_c}{L}$ would be 1. Since there is the term $\frac{k_c}{2L}^2$ subtracted from that, the summand for this component in the equation cannot be greater than 1. All other components remaining in the graph cannot have any links, which means that these are just unconnected single nodes. This means that their summand in the equation (ignoring the division by zero) would be 0. The whole equation thus cannot be greater than 1.

For other cases where there are multiple components containing links, the total number of L_c for all components is still limited by L. Thus it has to hold that $\sum_{c=1}^{n_c}(\frac{L_c}{L}) \leq 1$. Since the other term in each summand $(\frac{k_c}{2L}^2)$ can only decrease the overall sum, $M \leq 1$ has to hold.

For the trivial case of all nodes being disconnected, $M = 0$.