Data mining. Project Report - Analysis of KRS data.

Authors: Jacek Budny, Bartłomiej Jamiołkowski, Konrad Krzemiński

Have you ever considered what your neighbors do for living, or how wealth is distributed in society? If not, you have an up-to-date opportunity to dive into this topic deeper. By using multiple Data Mining techniques, we are going to uncover some mysteries of Polish business that will make you astonished. Still interested? Let's begin!

The whole project idea emerged during project classes of Data Mining course on AGH University in Cracow in spring 2024, where three ambitious students decided to join their forces to analyze the KRS data. At this point, you might be concerned what the KRS is.

What is Polish National Court Register, KRS?

Shortly speaking, the KRS register is a source of data about entities that operate in Poland. It mostly contains numerous information about organizations and people who are somehow connected with each other. They are obligated by Polish law to provide this data to the KRS. In theory, every person has the right to have insight into provided information. In reality, without advanced programming knowledge, it is very difficult to gather and analyze such data. The KRS owns website called register.io where some details can be found: For example:

- Legal form of entity;
- Business areas;
- Headquarters;
- Branches;
- Time when company was established;
- Bankruptcy;
- Data of entity representatives and more.

Purpose of our project

By taking this information into account, our team decided to discover how entities and people are connected. A task seemed to be simple, but in reality it turned out to be a very complex problem. First, we had to gather necessary data in the required format. Then, one of the colleagues had to create different types of graphs showing connections between connections. Finally, our specialist from Web Development area created a website where users can manually check mentioned graphs and find what they are looking for.

Web scraping KRS

In order to build different graphs that display connections between people and entities, some data structure has been proposed. As an example, a table can be seen below so you can imagine how collected data looks like.

KRS	OrganizationName	LegalForm	Representatives	RepresentativesID	ConnectedOrganizations
1	A	public limited company	John Smith	75475	C:3, D:10, F:100
1	A	public limited company	Katrine Johns	31414	
2	B	public partnership	Amy Brown	75536	D:65
2	B	public partnership	George Taylor	52543	A:1
3	C	Individual activity	John Lee	70874	E:36

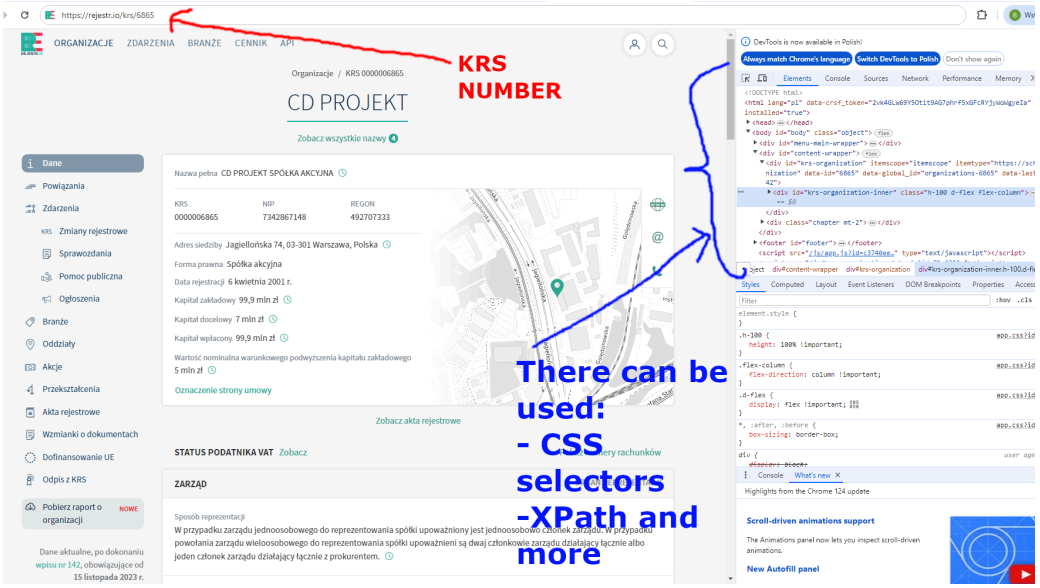
Variables:

- KRS - KRS is a unique number for an entity;
- OrganizationName - organization name is not unique;
- LegalForm - Polish legal form of entity;
- Representatives - Firstname and Surname of person who represents organization;
- RepresentativesID - every representative has a unique ID, because their names can occur many times, e.g. John Smith;
- ConnectedOrganizations - other organizations connected with a representative. This column can sometimes be empty or has values similar to Python dictionary. It is necessary to bound organization name with KRS number, because organization names are not unique.

At this point, we are moving to our web scraper. It was written in Python programming language with usage of a tool called Selenium. Selenium provides WebDriver for different types of web browsers. For example webdriver.Chrome() is for Chrome browser, and therefore it is used in this project to obtain website elements.

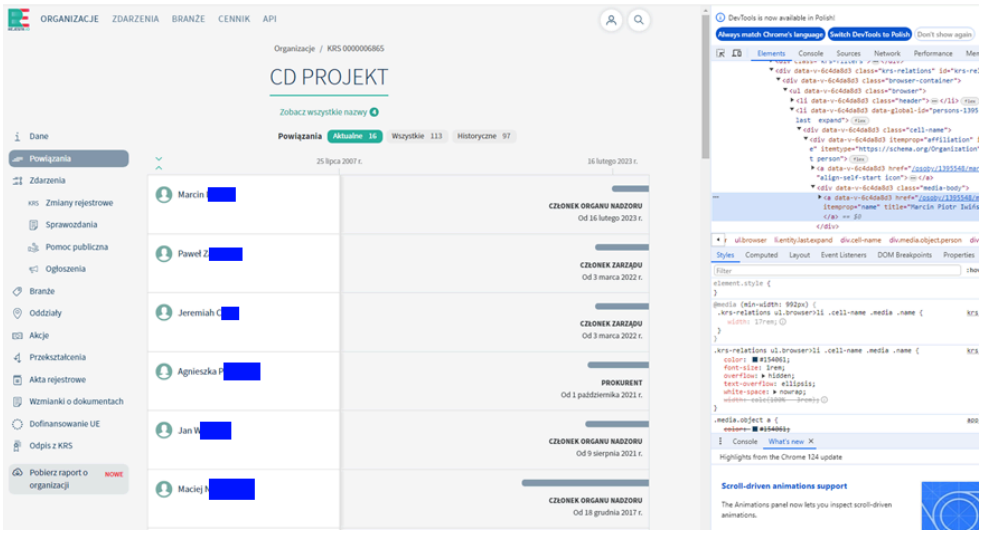
```python
import numpy as np
from selenium.common.exceptions import NoSuchElementException
from selenium import webdriver
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.common.by import By
import pandas as pd
import csv


Bartłomiej Jamiołkowski
def initialize_driver() -> tuple:
    options = webdriver.ChromeOptions()
    driver_org = webdriver.Chrome(ChromeDriverManager().install(), options=options)
    driver_rep = webdriver.Chrome(ChromeDriverManager().install(), options=options)
    driver_con = webdriver.Chrome(ChromeDriverManager().install(), options=options)
    return driver_org, driver_rep, driver_con
```

The main frame of a program is a loop which iterates over a range of values specified by a user (start_num and end_num). These values are possible KRS numbers. We do not know real numbers of Polish entities, so web scraper has to check them one by one. It constructs the URL for the given KRS number and navigates to the webpage. Then, web scraper checks if the webpage exists by verifying the title. If checked number does not exist or company was removed from KRS register, web scraper omits such web pages and do not scrape data. Otherwise, it stars searching elements on existing web page by using XPath, CSS selectors, which are reported to be quite efficient methods. Below is presented a picture how it looks like in theory.



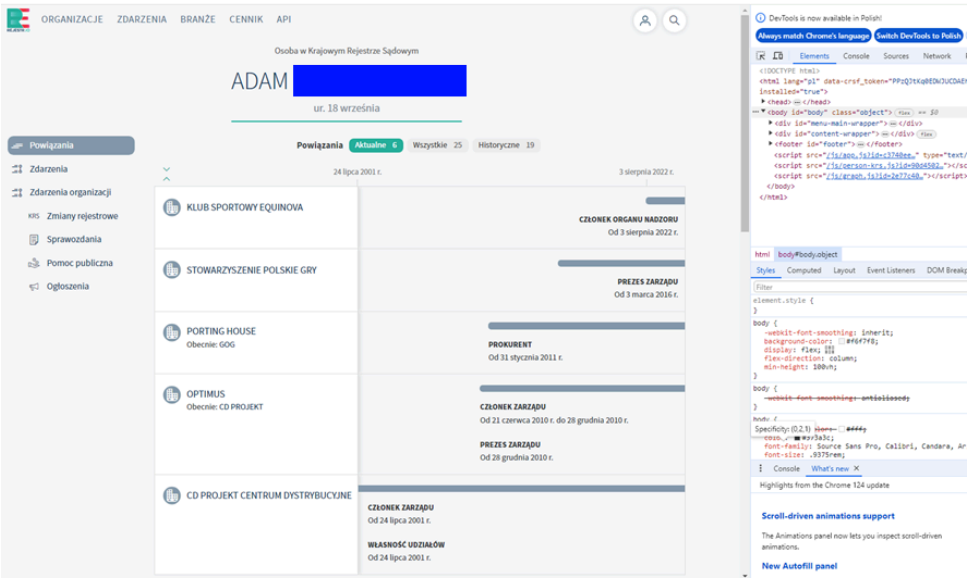

```python
def scrape_krs_data(start_num: int, end_num: int) -> pd.DataFrame:
    driver_organizations, driver_representatives, driver_connections = initialize_driver()
    krs_dataset = pd.DataFrame(columns=['KRS', 'OrganizationName', 'LegalForm', 'Representatives', 'RepresentativesID',
                                        'ConnectedOrganizations'])

    for krs_num in range(start_num, end_num):
        print(f'KRS: {krs_num}')
        url = 'https://rejestr.io/krs/' + str(krs_num)
        driver_organizations.get(url)

        if driver_organizations.title != 'Nie znaleziono strony | Rejestr.io':
            try:
                organization_name = driver_organizations.find_element(By.XPATH, '//div[@class="h1_wrapper"]/h1/a').text
            except NoSuchElementException:
                organization_name = np.nan

            try:
                legal_form = driver_organizations.find_element(By.CSS_SELECTOR,
                                                               "div[data-name='forma_prawna'] .val.text-normalize span").text
            except NoSuchElementException:
                legal_form = np.nan

            try:
                link_to_representatives = driver_organizations.find_element(By.LINK_TEXT, 'Powiązania').get_attribute(
                    "href")
            except NoSuchElementException:
                link_to_representatives = None
```

At this point the program extracts information such as organization name, legal form, and link to representatives. Then, it navigates to the page containing representatives using the extracted link.




```python
if link_to_representatives:
    driver_representatives.get(link_to_representatives)
    representatives = driver_representatives.find_elements(By.CSS_SELECTOR,
                                                           "div[class='media object person']")
    representatives_id = driver_representatives.find_elements(By.CSS_SELECTOR, 'li[data-global-id]')
    list_of_representatives_id = []
    count = 0

    for id in representatives_id:
        rep_id = id.get_attribute("data-global-id")
        rep_id = ''.join(filter(str.isdigit, rep_id))
        list_of_representatives_id.append(rep_id)

    for representative in representatives:
        krs_dataset_row = dict.fromkeys(
            ['KRS', 'OrganizationName', 'LegalForm', 'Representatives', 'RepresentativesID',
             'ConnectedOrganizations'])
        krs_dataset_row['KRS'] = krs_num
        krs_dataset_row['LegalForm'] = legal_form
        krs_dataset_row['OrganizationName'] = '"' + organization_name + '"'
        krs_dataset_row['Representatives'] = representative.text
        krs_dataset_row['RepresentativesID'] = list_of_representatives_id[count]

        link_to_connections = None
        try:
            link_to_connections = driver_representatives.find_element(By.LINK_TEXT,
                                  representative.text).get_attribute("href")
        except NoSuchElementException:
            pass
```

The Program scrapes representatives data (first name and last name) and their unique IDs. For each representative, it collects their connected organization's information. However, at this stage sometimes can appear difficulties. Foremost, some foreign representatives do not have their profile web pages on rejestr.io what is not common and rise concerns about their identity. Therefore, the program can scrape only their first name and last name, but not possible connections with other entities. What is more, organizations names are not updated in a list of representatives connections. As a result, the web scraper has to update these names on its own.




```python
if link_to_connections:
    list_of_connections = []
    list_of_connections_krs = []
    driver_connections.get(link_to_connections)
    connections = driver_connections.find_elements(By.CSS_SELECTOR,
                                                   "div[class='media object krs org']")
    connections_krs = driver_connections.find_elements(By.CSS_SELECTOR, 'li[data-global-id]')

    for connection_krs in connections_krs:
        org_num = connection_krs.get_attribute("data-global-id")
        org_num = ''.join(filter(str.isdigit, org_num))
        if int(org_num) != krs_num:
            list_of_connections_krs.append(org_num)

    for connection in connections:
        if "Obecnie: " in connection.text:
            actual_connection_name = connection.text.split("Obecnie: ")[1].strip()
            if len(connections) == 1 and actual_connection_name == organization_name:
                krs_dataset_row['ConnectedOrganizations'] = np.nan
            elif len(connections) > 1 and actual_connection_name == organization_name:
                continue
            elif len(connections) > 1 and actual_connection_name != organization_name:
                list_of_connections.append(actual_connection_name)
        else:
            if len(connections) == 1 and connection.text == organization_name:
                krs_dataset_row['ConnectedOrganizations'] = np.nan
            elif len(connections) > 1 and connection.text == organization_name:
                continue
            elif len(connections) > 1 and connection.text != organization_name:
                list_of_connections.append(connection.text)
```

```
                    krs_dataset_row['ConnectedOrganizations'] = str(
                        dict(zip(list_of_connections, list_of_connections_krs))).replace('{', '').replace('}', '')
                else:
                    krs_dataset_row['ConnectedOrganizations'] = np.nan

                krs_dataset = pd.concat([krs_dataset, pd.DataFrame([krs_dataset_row])], ignore_index=True)
                count += 1

    driver_organizations.quit()
    driver_representatives.quit()
    driver_connections.quit()
    return krs_dataset


if __name__ == '__main__':
    start_num = 30000
    end_num = 35000
    krs_dataset = scrape_krs_data(start_num, end_num)
    krs_dataset.to_csv(f'data/krs_data_{start_num}_{end_num}.csv', encoding='utf-8-sig', index=False, sep=',',
                       quotechar='"', quoting=csv.QUOTE_ALL)
```

Results are stored in pandas data frame and saved to CSV file when the program stop checking a given range of possible KRS numbers.

Analysis of business linkage graphs

Our research on the structure of business links uses the methods and tools of graph theory. Linkage graphs, which represent relationships between entities such as companies and persons, can provide interesting insights into the structure of business networks. In this section, we will focus on the analysis of the linkage graph, with a presentation of the methods used to conduct the research.

Introduction to link graph analysis

We began our analysis by creating a two-color graph, in which nodes of one color represent companies, nodes of the other color represent persons, and edges represent the connections between them. This approach allows us to model complex relationships in business networks and better understand their structure.

Tools

For this, we used the NetworkX library. NetworkX is a universal tool in the world of network analysis, which allows for basic graph operations and advanced structural analysis. We used the pandas library to import the data, and matplotlib to present the graphs.

Data conversion

We converted the data stored in the CSV file into a Graph object.

```python
G = nx.Graph()
person_color = 'red'
company_color = 'blue'
```

```python
import re

def add_company_from_row(graph, row):
  krs_num = int(row[0])
  name = row[1]
  legal_form = row[2]
  if not graph.has_node(krs_num):
    graph.add_node(krs_num, name=name, legal_form=legal_form,
                   color=company_color)

  return krs_num

def add_person_from_row(graph, row):
  person_id = "P" + str(row[4])
  name = row[3]
  if not graph.has_node(person_id):
    graph.add_node(person_id, color=person_color, name=name)

  return person_id

def add_relationship(graph, node1, node2):
  if not graph.has_edge(node1, node2):
    graph.add_edge(node1, node2)

def add_connected_org(graph, person_id, row):
  connected_org = row[5]
  if pd.isna(connected_org):
    return

  pattern = r"'(.*?)': '(\d*?)'"

  # Find all matches
  matches = re.findall(pattern, connected_org)

  result = {match[0]: int(match[1]) for match in matches}
  for name, company_id in result.items():
    if not graph.has_node(company_id):
      graph.add_node(company_id, name=name, legal_form=None,
                     color=company_color)

    add_relationship(graph, company_id, person_id)
```

```python
for index, row in df.iterrows():
    company_id = add_company_from_row(G, row)
    person_id = add_person_from_row(G,row)

    add_relationship(G, company_id, person_id)

    add_connected_org(G, person_id, row)
```

By iterating over successive rows from the dataset, corresponding to a particular person, we create connections between that person and the companies to which he or she is connected. In this way, by adding vertices and edges, a graph of links is created.

Basic structural analysis

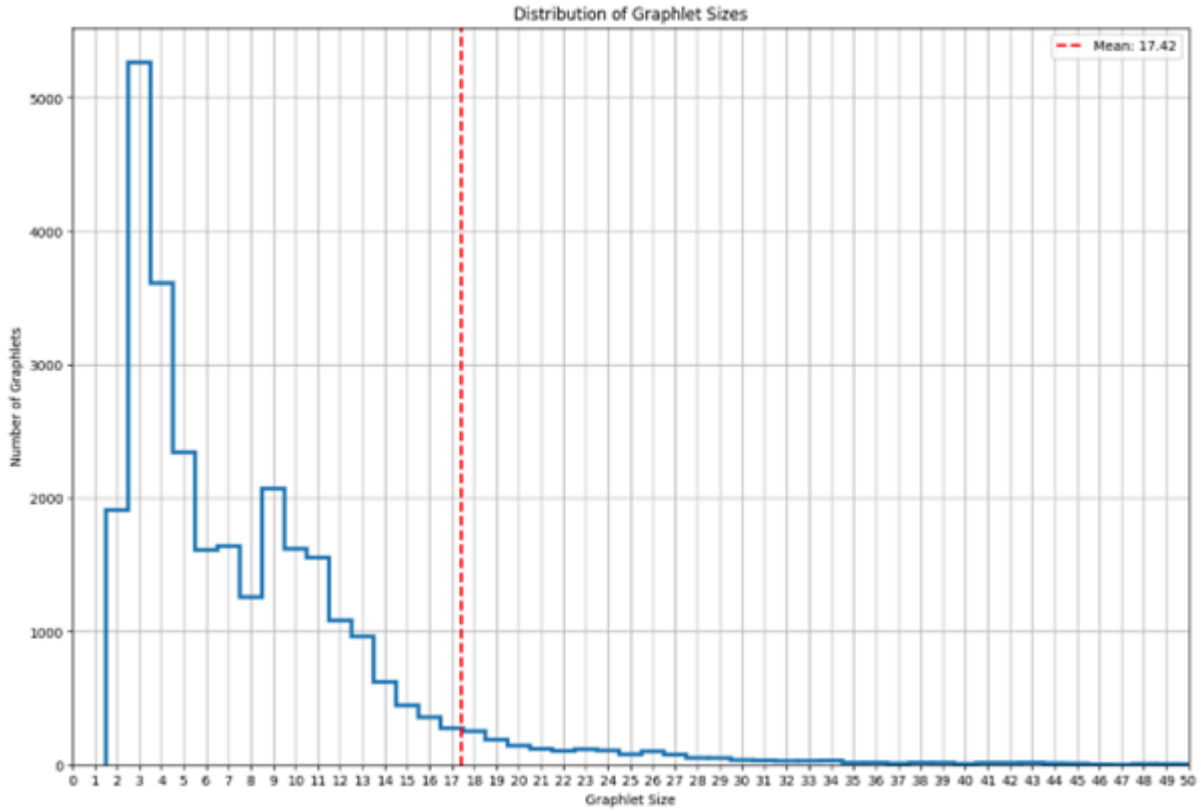
To begin with, it is useful to know some basic statistical properties of a graph for further analysis. A graph consists of 496349 vertices and 532702 edges. The graph includes 150336 companies and 346013 people. The graph has 28488 connected components. Size of ten biggest subgraphs is: [265328, 342, 197, 132, 131, 110, 105, 101, 90, 87]. The size of the largest subgraph seems surprising, and we will analyze it later in the article. The highest degree of the vertex is 350.

Exploring the structure of the graph further, it is worth focusing on analyzing the distribution of graphlet sizes. For this, we used the nx.connected_components(G) method from the NetworkX library, which allows us to identify successive sets of connected vertices. This method uses the Breadth-First Search (BFS) algorithm, which is one of the basic graph search techniques.

To calculate the size distribution of subgraphs, we first create a list of sizes, and then draw a histogram of these values.

```
# Counting graphlets
graphlet_sizes = [len(c) for c in sorted(nx.connected_components(G),
                                         key=len, reverse=True)]

mean_size = np.mean(graphlet_sizes)
std_size = np.std(graphlet_sizes)

plt.figure(figsize=(15, 10))
plt.hist(graphlet_sizes,
         bins=range(min(graphlet_sizes), max(graphlet_sizes) + 1, 1),
         histtype='step', align='left', edgecolor='C0', linewidth=3)

plt.xlabel('Graphlet Size')
plt.ylabel('Number of Graphlets')
plt.title('Distribution of Graphlet Sizes')

plt.xticks(range(0, 51, 1))
plt.xlim((0, 50))
plt.grid(True)
plt.legend()

plt.axvline(mean_size, color='r', linestyle='dashed',
            linewidth=2, label=f'Mean: {mean_size:.2f}')

plt.show()
```

The distribution of subgraph sizes is shown in the figure below:



In the main graph, most subgraphs have sizes ranging from 2 to 13. There are the most subgraphs with a size of 3. The average subgraph size is 17.42.

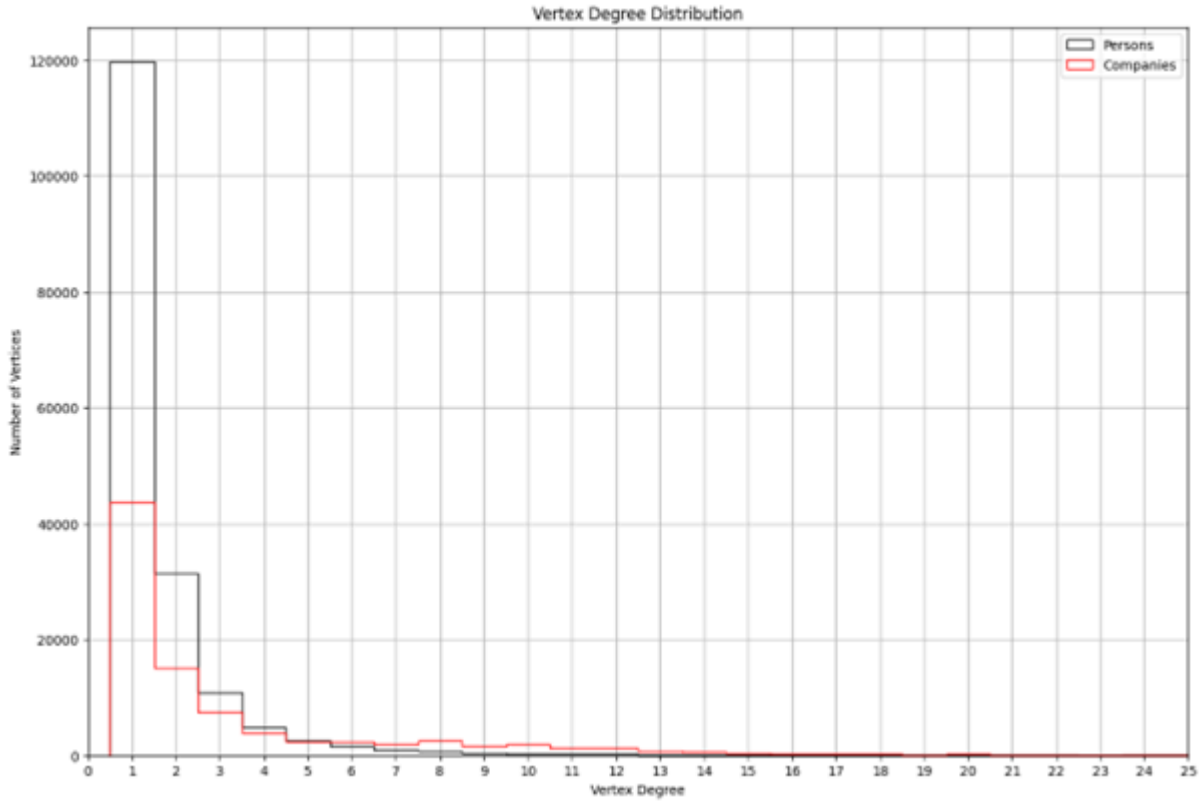
Analysis of the largest subgraph

Due to the unusual size of the largest subgraph, it is worth focusing on its deeper analysis to better understand the data.

To find out if the creation of this graph is not related to some error, it is worth looking for whether there are vertices connected to many other vertices. For this purpose, you can determine the maximum degree of a vertex, in addition, we will distinguish between people and companies.

```
person_degrees = [degree for node, degree in subgraph.degree()
                  if subgraph.nodes[node]['color'] == person_color]
company_degrees = [degree for node, degree in subgraph.degree()
                   if subgraph.nodes[node]['color'] == company_color]
```

Calculations show that the maximum degree of the vertex corresponding to a person is 350 and the company is 158.
After checking the vertex with the highest degree, we concluded that this is the case of a person associated with 350 companies.
To learn more about the structure of the graph, it is useful to plot the degree distribution of the vertices.

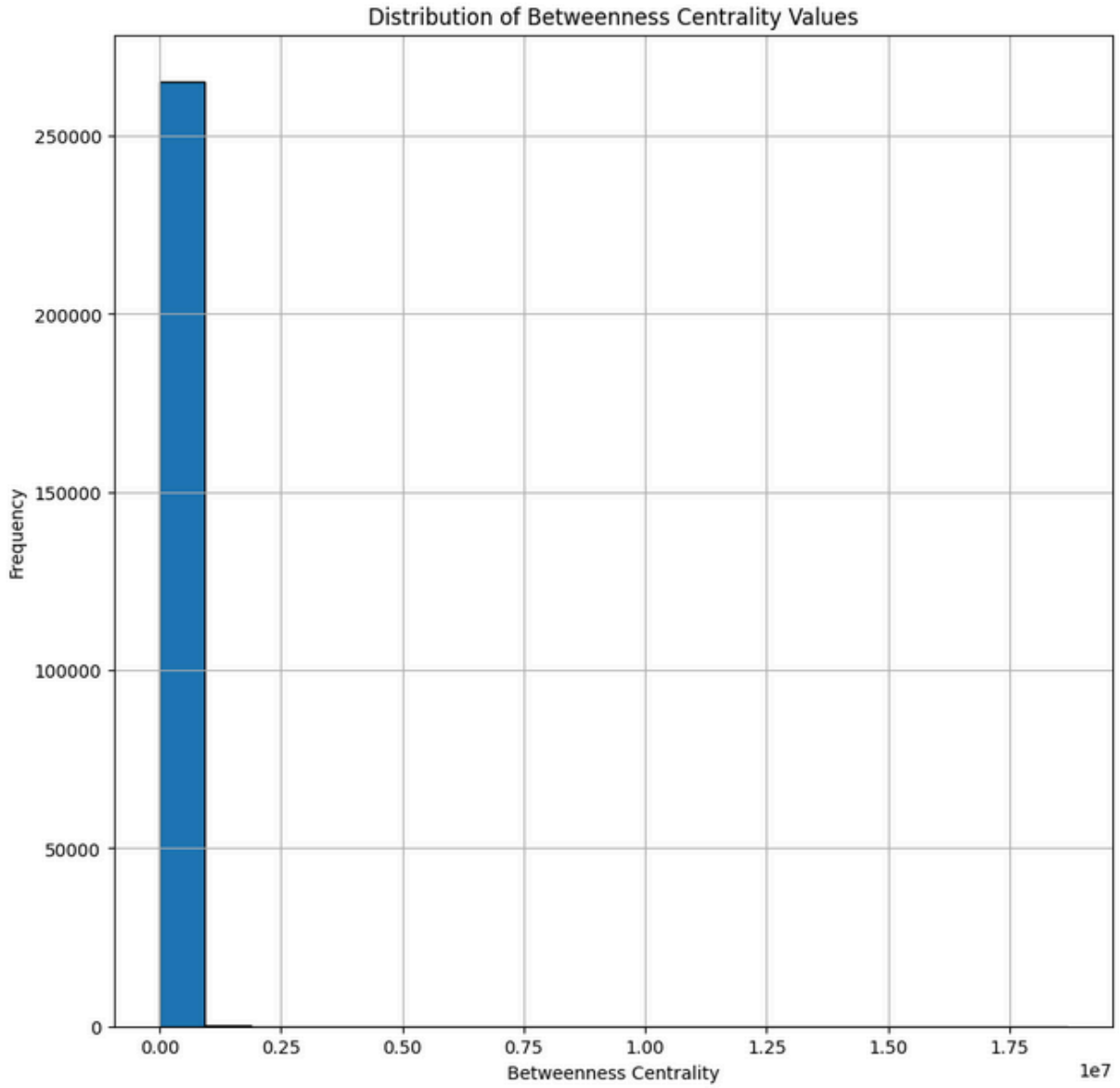


The plot shows that there are most vertices with low degree. This suggests that this is a graph in which the vertices have few neighbors and the graph merges into a larger whole through the presence of a relatively small number of vertices with high degree.

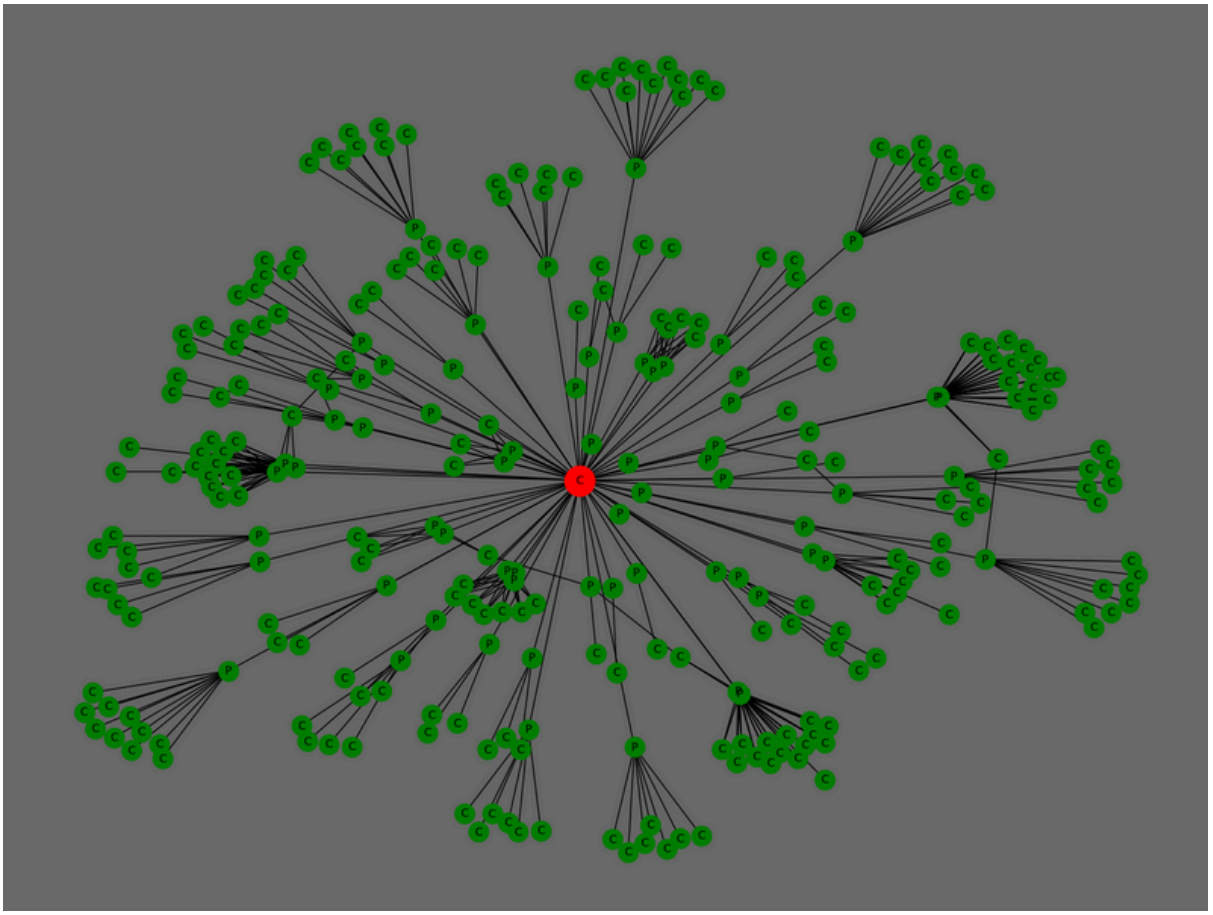
And here are the other basic properties of this graph:
Number of nodes: 265328
Number of edges: 319695
Average degree: 2.41
Density: 9.08e-06

Betweenness centrality

Going further in the analysis of the largest subgraph we analyzed betweenness centrality Betweenness centrality is a measure used in network analysis to quantify the importance of a node (or an edge) within a network. It does this by looking at the number of shortest paths that pass through the node.



The betweenness centrality graph gives not much information unfortunately. For a deeper analysis, we drew the graph with the largest betweenness centrality along with second-order neighbors.



As you can see a vertex is directly connected to many other vertices.

Pattern analysis

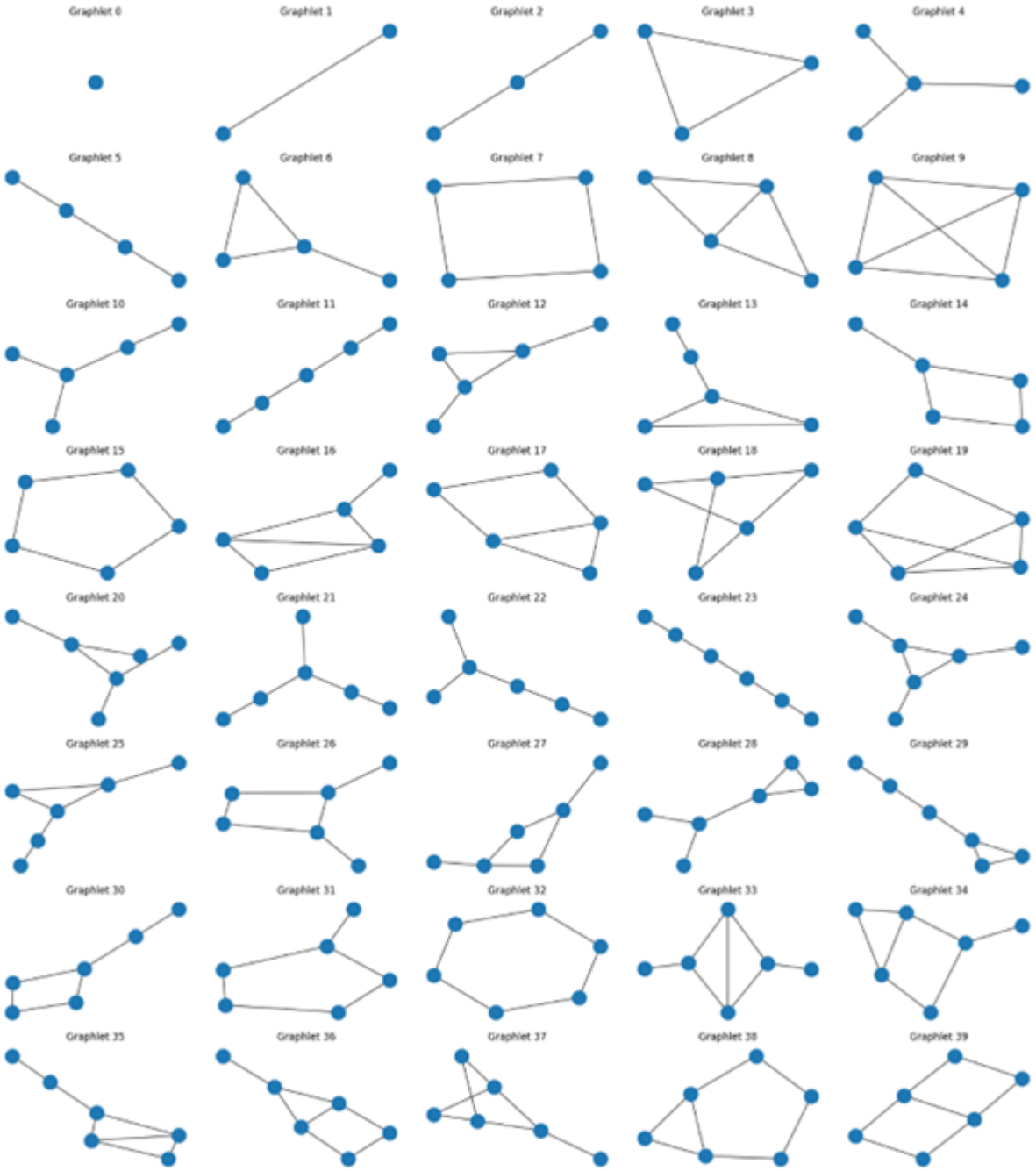
Continuing our analysis, we focused on studying the frequency of graphelets in the linkage graph. Graphelets are small repeating subgraphs in a larger network. They allow us to study local patterns and structures in the network. In the context of a company's linkage graph, which consists of many small subgraphs that are not connected to each other, graphlet analysis can help understand how small units affect the overall structure of the network.

To perform this analysis, we used the graphlet atlas built into the Networx library. The nx.graph_atlas_g() method returns a list of all graphs up to seven vertices, sorted by number of vertices, number of edges and several other properties. For our analysis, we chose graphs whose

- the number of vertices is less than seven,
- the degree of vertices is less than four
- which are consistent graphs.

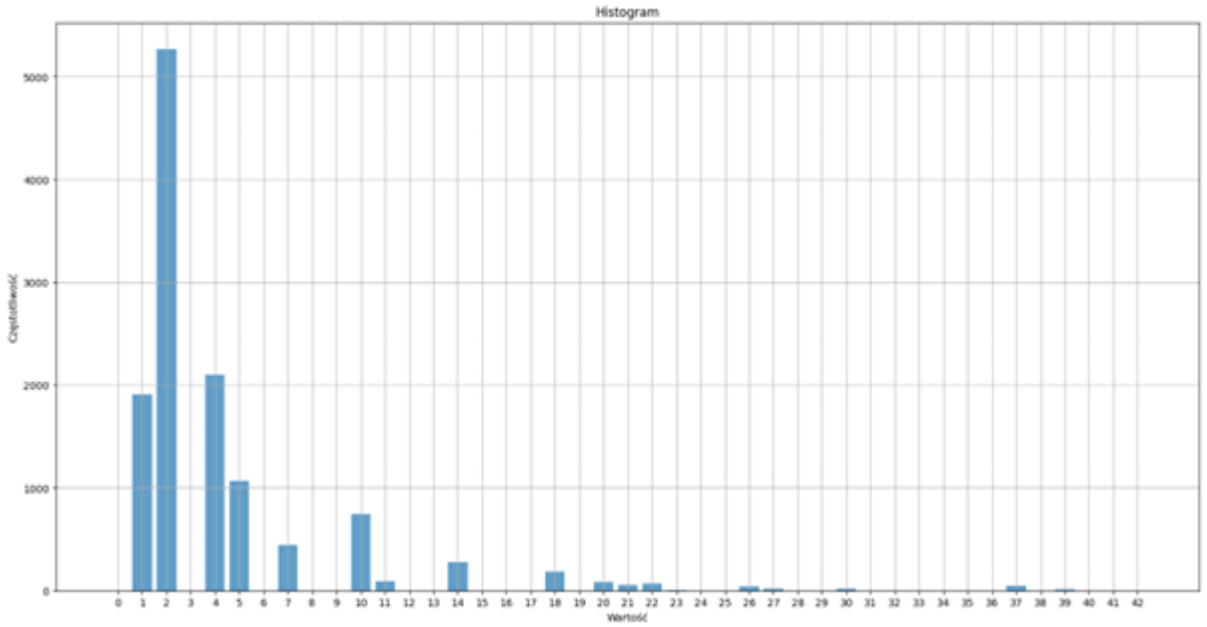
The selection process allowed us to focus on the most representative and important subgraphs.

Some of the graphlets included in the analysis are shown in the figure below:

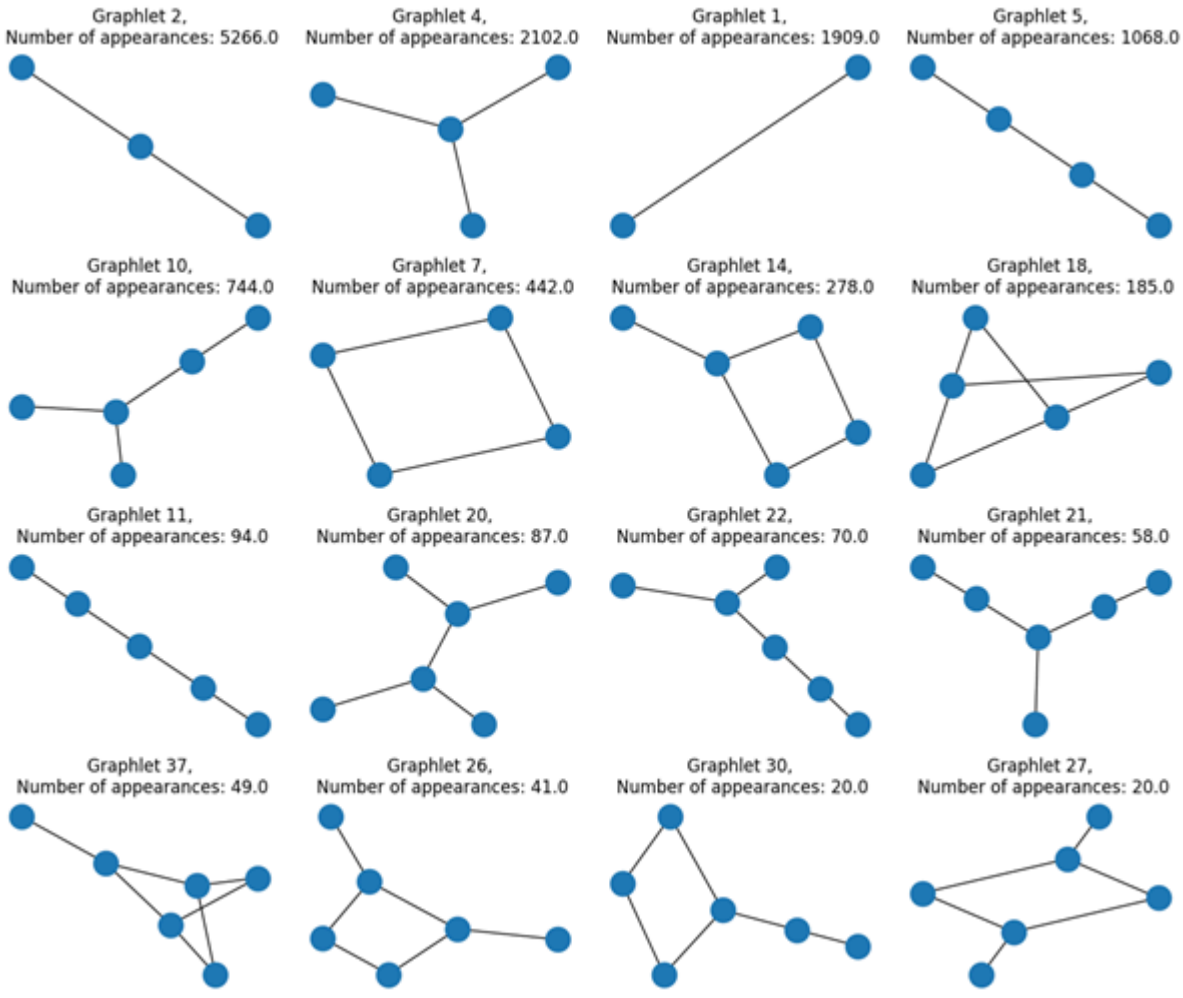


After building a list of graphlets to be included in the analysis, we began searching the linkage graph to examine the number of occurrences of the selected graphlets. A key aspect of this process was the use of the nx.is_isomorphic(G1, G2) method, which allows two graphs to be compared in terms of their structure.

As a result of this analysis, we obtained the distribution of graphlets shown in the figure below.

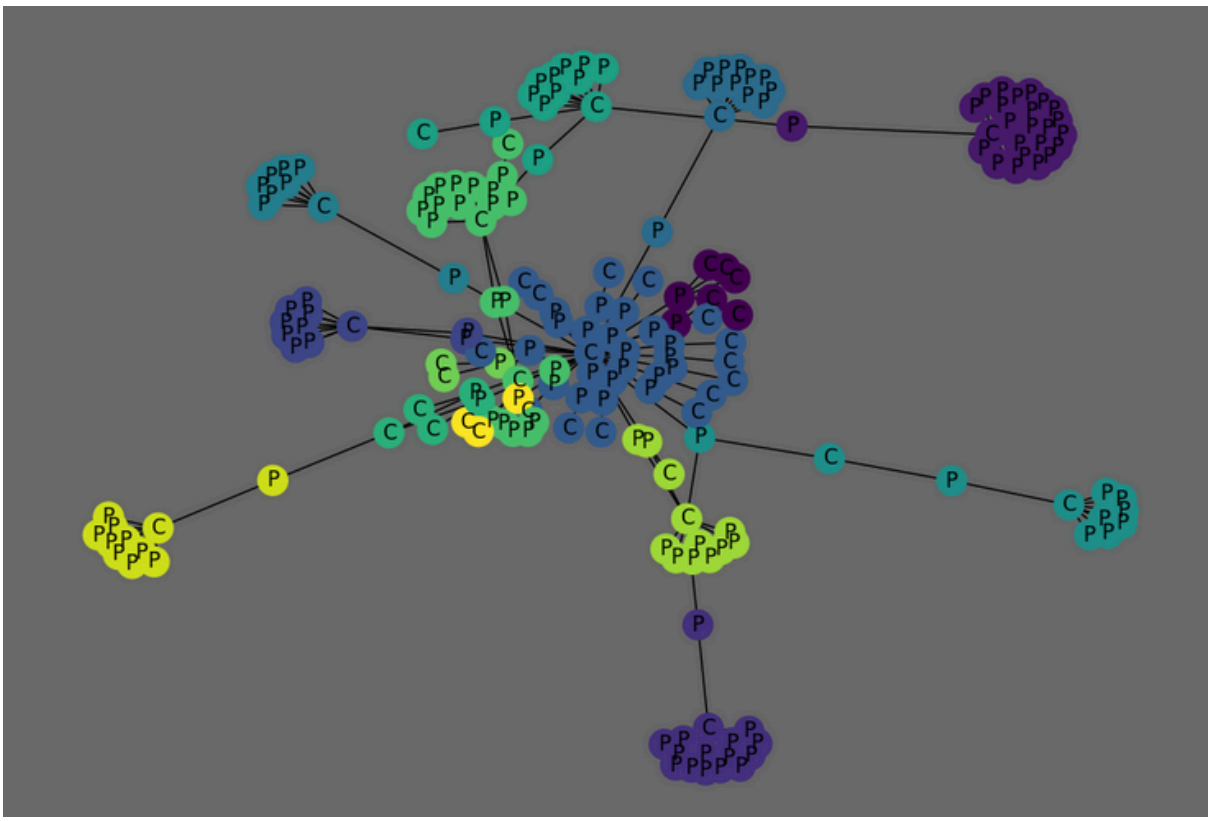


And visualization of the most common graphlets:



Community analysis

For the smaller subgraph, we are able to perform a community occurrence analysis using the nx.algorithms.community.girvan_newman(G) method. Which uses the Girvan Newman algorithm. This algorithm detects communities by progressively removing edges from the original network. The connected components of the remaining network are the communities. As a result of the algorithm, we are able to determine communities and then draw them on a plot

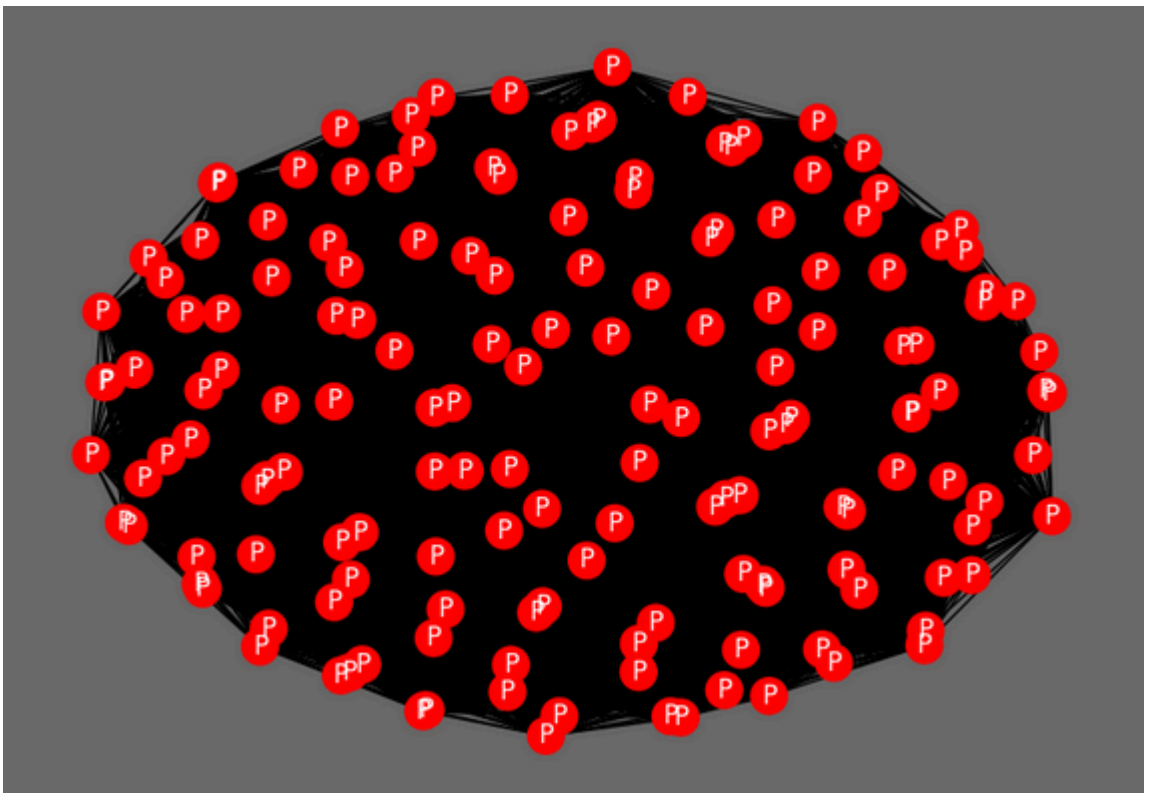


As you can see in the example subgraph, and as our analysis shows, a common case is the clustering of people around some company that has a connection to other companies by one or more people.

Person - person graph

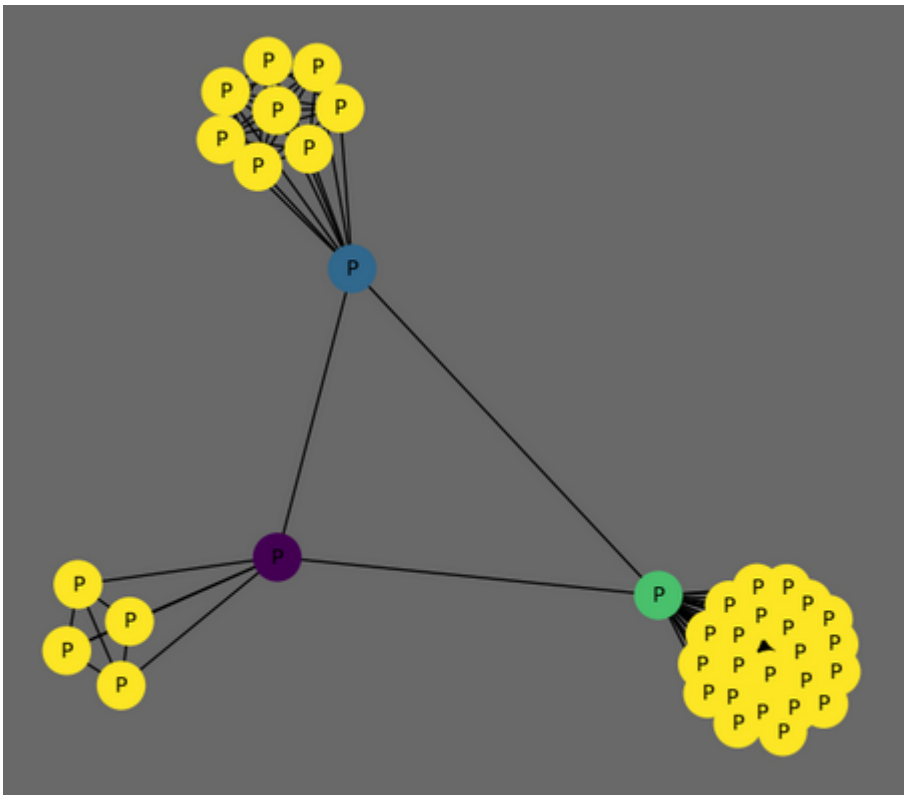
At the end of the analysis, we constructed a person-to-person graph.

In this graph you can find many cliques, they are formed when many people are related to a certain organization.



The largest clique has 158 vertices.

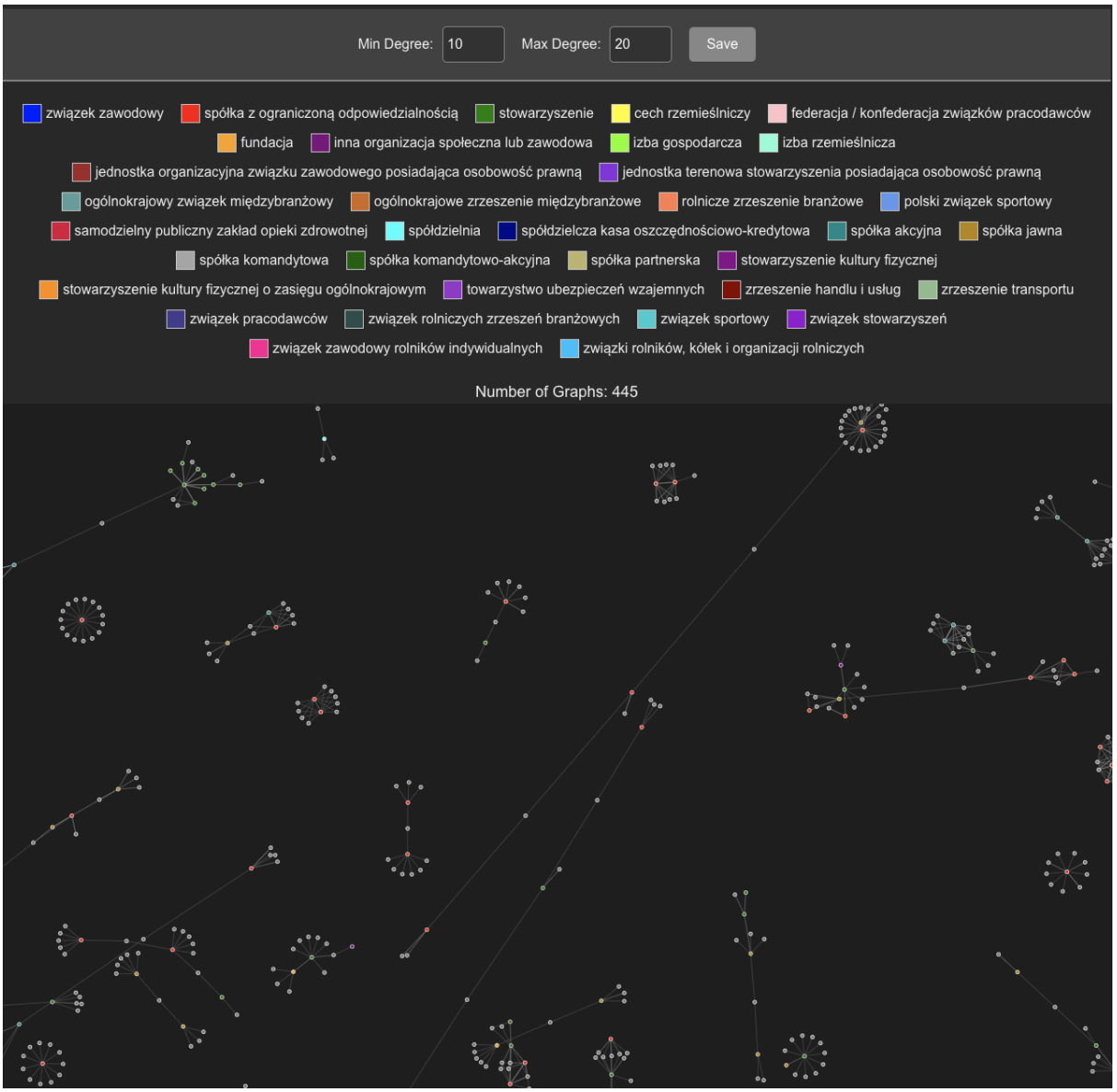
For a certain subgraph, we can also show the distribution of the clustering coefficient. The clustering coefficient is a measure used in network analysis to quantify the degree to which nodes in a graph tend to cluster together. It provides insights into the local density of connections in a network.



High values of this coefficient have vertices that are strongly connected with each other.

Subgraphs visualization

In order to visualize the links between the companies, we have developed a web application that provides this functionality. The application was written in typescript using React.js. It displays a set of subgraphs with a given range of the number of vertices. The vertices are the organizations and the edges are the people connecting them. Clicking on a vertex will display a modal with the name of the company. The functionality of coloring vertices according to the type of organization is also implemented. It is worth mentioning that uncoloured vertices are those that have not been scrapped and appear in connected organizations. The number of subgraphs for a given visualization is also shown. Below is a sample output of our application:



Selected visualizations

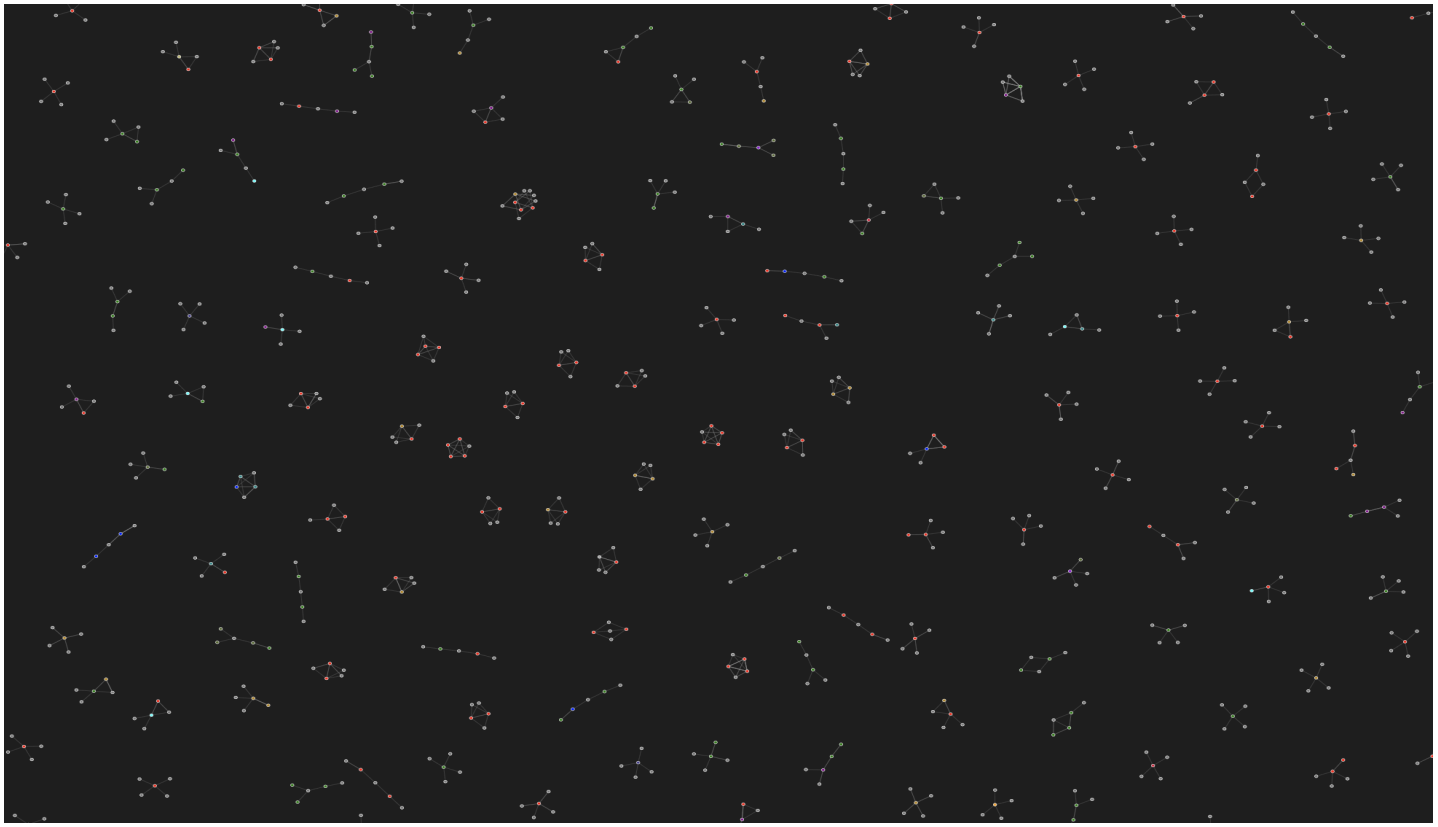
1) **Graphs with the number of nodes 3**

It is easy to see that a line structure is much more common than a triangle structure.

2) **Graphs with the number of nodes 4**

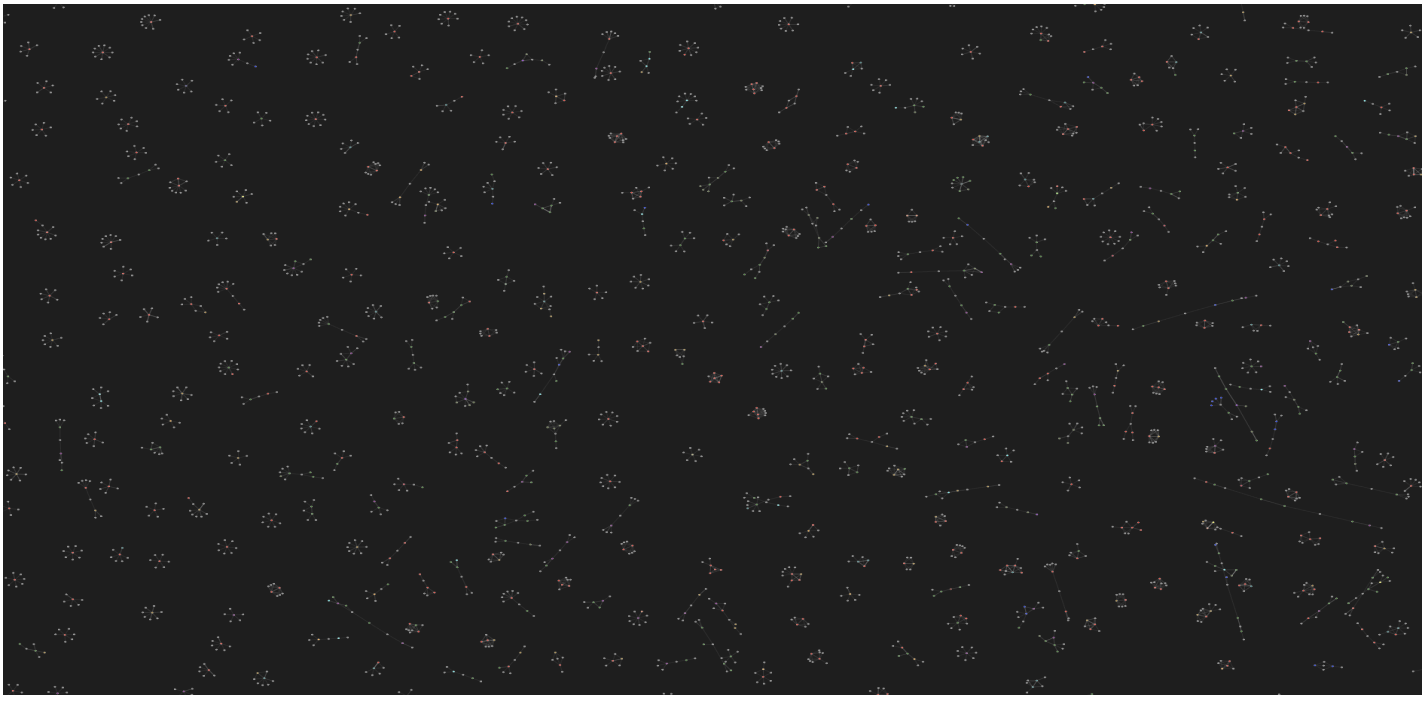
It can be seen that graphlets with a centre point and in the shape of a line predominate.

3) Graphs with the number of nodes 5



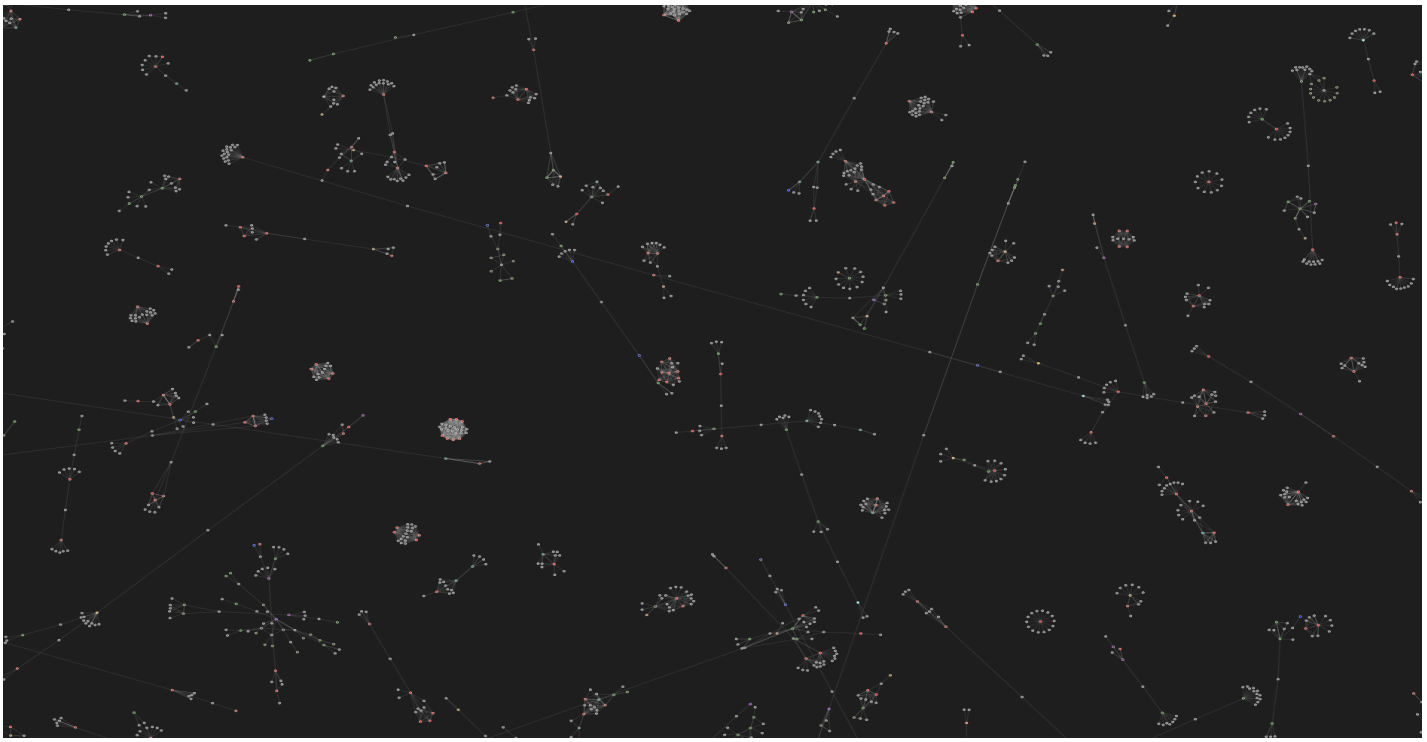
Instead, here we see a considerable variety of graphemes, making it hard to overtly state which type of subgraph is the most common.

4) Graphs with the number of nodes 6 - 10



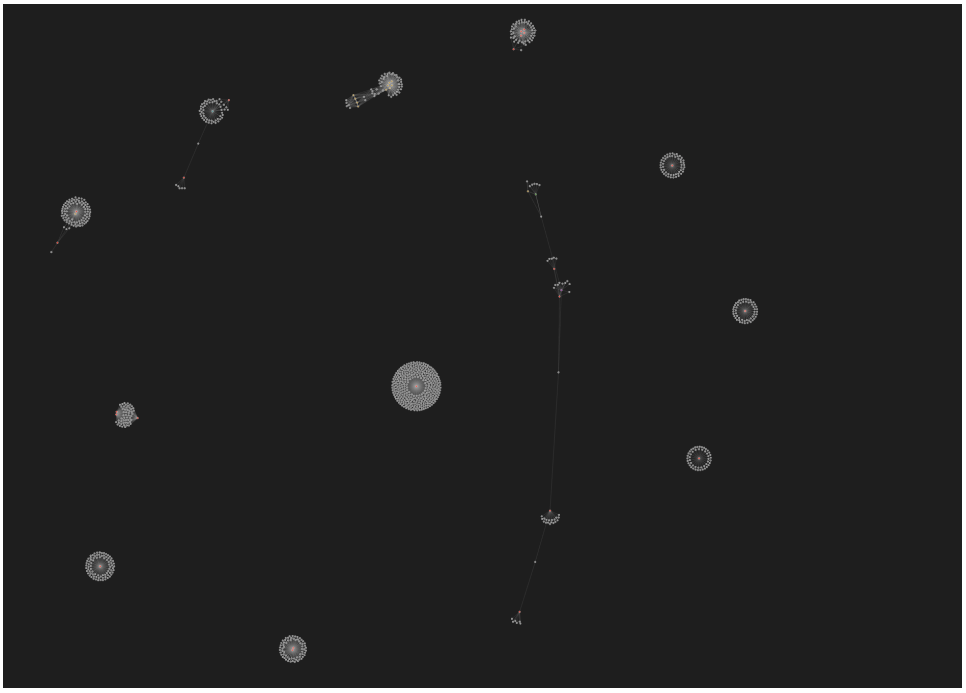
Although the variety of shapes is considerable, there is already a trend towards graphlets focused on a single point.

5) Graphs with the number of nodes 11 - 30



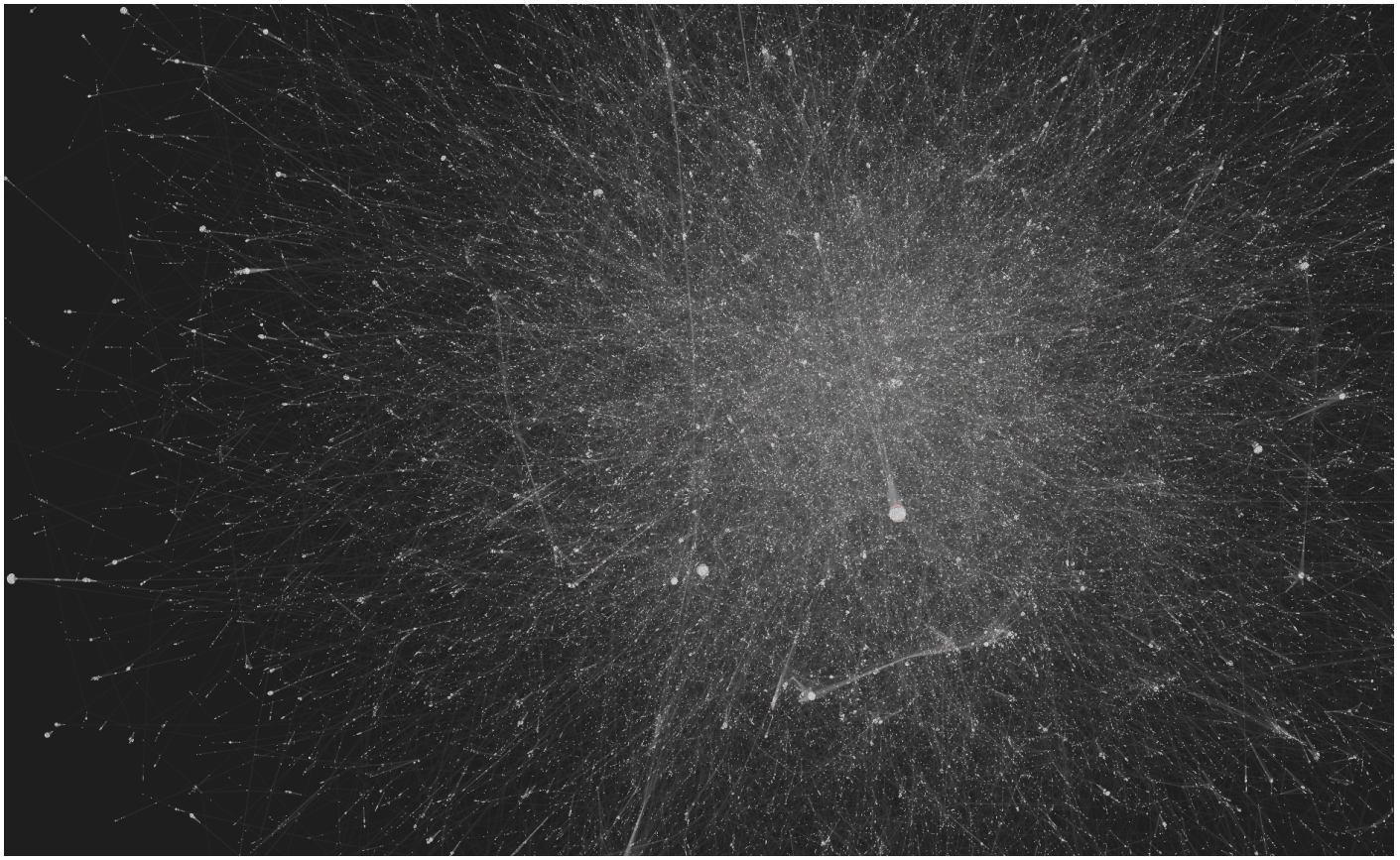
We see many different shapes, it is hard to find a specific pattern

6) Top 2 - 13 the largest subgraphs



These are graphs with a size of 50 - 342 nodes. It can be seen that graphs with a given centre point dominate.

7) Largest graph



It is interesting to note that there is a graph consisting of a thousand nodes. Its visualisation does not show a clear level of where it comes from.