University of Szeged

Department of Computational Optimization

Embedded network structures of transaction graphs

Master Thesis

Written by:

Daniella Nikov

Advisor:

András Pluhár

Associate Professor

Szeged

2022

Table of contents

1 Description of topic

Study and analyze graph structures, implement clustering and community detecting algorithms. The application has to be able to handle directed and undirected graphs as well. The edges can carry weight information as attributes. The application should be covered with unit tests.

2 Abstract

A prototype console application has been implemented to detect social networks and clustering transactional type of the graphs. Several algorithms have been implemented to run against the given databases, such as Girvan-Newman, Markov chain, etc. The application is able to map a graph (also known as network) with any given delimiter, in any text format furthermore there is an option to connect to database if necessary. The results of the algorithms are plotted and displayed after the run. The coloring logic has been implemented in a fairly naive and greedy colormap. The application is scalable and modularized.

3 Introduction

World Wide Web, blogging platforms, instant messaging and Facebook can be characterized by the interplay between rich information content, the millions of individuals and organizations who create and use it, and the technology that supports it. This thesis will cover the processing of recent research on the structure and analysis of large social and transaction networks and on models and algorithms that abstract their basic properties. Unusual ways have been explored how to practically analyze large scale network data and how to reason about it through models for network structure. Topics include methods for network community detection and their connection with transactional graphs. [1]

Community detection and analysis is an important methodology for understanding the organization of various real-world networks and has applications in problems as diverse as consensus formation in social communities. Currently used algorithms that identify the community structures in large-scale real-world networks require a priori information such as the number and sizes of communities or are computationally expensive. I intend to rely more on algorithms, which use the network structure as their guide instead of this priori information. Finding community structures in networks is another step towards understanding the complex systems they represent [2]. Social networks are represented by people as nodes and their relationships by edges therefore they contain more triangles than a random graph with similar edge density or degree properties. In contrast technological or transaction graphs contain fewer triangles and often display tree-like structures [15].

I gained knowledge about some of the prerequisites of social graph studies, for example algorithms of community detection have been used, along with the extracted data from an earlier implemented software, the Sixtep program, as well.

Based on the research written in article [15] and some earlier studies I intend to give an explanation about the implementation of algorithms for generalized coloring of transactional graphs. As the study goes, I follow the heuristics and possible solutions for the clustering problem.

There was an attempt to find the embedded pattern in the provided transaction-like graphs.

4 Used Technologies

- Python, PyEnv, PyCharm
- NetworkX
- Sixtep software
- Git
- Graph Databases

4.1 Python, PyEnv, PyCharm

Python is a programming language that makes possible to work more quickly and integrate the designed system more effectively. Perfect choice to develop proof of concepts, make prototypes. It can be easy to pick up whether the developer makes a first time project or has experience with other languages. The well documented packages help the programmer along the learning way and the gains in productivity can bee seen almost immediately. It also reduces maintenance costs of the targeted application. The packages used in this thesis are developed under open source license, making it freely usable and distributable not only educational purposes but even for commercial use. [16]

PyEnv makes it possible to switch between multiple versions of Python. It is unobstructive and follows the UNIX tradition of single-purpose tools. PyEnv-virtualenv is a PyEnv plugin that provides features to manage virtualenvs for Python on UNIX-like systems.

PyCharm is a Python IDE with intelligent code completion, on-the-fly error checking, quick fixes, built in support for Numpy, Matplotlib and other scientific libraries while offering graphs, array viewers and much more.

These tools have been chosen for thesis purposes not only because they are open source, but the documentations and tutorials are in good quality. The tools have wide range of support layer and make the development ready to start. Perfect choice when the scope is widely changes during the development and each features have to be prototyped at first.

4.2 Git

Git is a free and open source distributed version control system designed to handle everything from small to very large project with speed and efficiency. It has wide range of supported features, like branching and merging strategy, distribution, data assurance, staging area, however during the app development for the thesis, it has only been used for version control.

4.3 NetworkX

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. It supports a variety of features, provides classes for graph objects, generators to create standard graphs, IO routines for reading in existing datasets, algorithms to analyze the resulting networks and some basic drawing tools.

Most of the NetworkX API is provided by functions which take a graph object as an argument. Methods of the graph object are limited to basic manipulation and reporting. This provides modularity of code and documentation. It also makes it easier to learn about the package in stages.

In NetworkX, a graph is represented by a collection of edges that are pairs of nodes. Attributes are often associated with nodes and/or edges. NetworkX graph objects come in different flavors depending on the properties of the network, like is the graph directed, or are multiple edges allowed. Based on that, the basic graph classes are Graph, Directed Graph, Multi Graph, Multi-Directed Graph. Graph objects can be created by importing data from pre-existing (usually file) sources. In this thesis, graph algorithms provided by NetworkX include shortest path and breadth first search and clustering algorithm. While NetworkX in not designed as a network drawing tool, it provides a simple interface to drawing packages and some simple layout algorithms. In the application drawing has been done using this package and the Matplotlib Python package. The basic drawing functions essentially place the nodes on a scatterplot using the positions which have been provided via a dictionary or the positions are computed with a layout function. The edges are represented as lines between the dots. NetworkX uses dictionaries as the basic network data structure. This allows fast lookup with reasonable storage for large sparse networks. [4]

- Data structures for graphs, digraphs, and multigraphs
- Many standard graph algorithms
- Network structure and analysis measures
- Generators for classic graphs, random graphs, and synthetic networks

- Nodes can be "anything" (e.g., text, images, XML records)
- Edges can hold arbitrary data (e.g., weights, time-series)
- Open source 3-clause BSD license
- Well tested with over 90% code coverage
- Additional benefits from Python include fast prototyping, easy to teach, and multi-platform

Algorithms:

Bipartite module provides functions and operations for Graph and DiGraph classes. The algorithms are not imported into the NetworkX namespace at the top level, so it has to be added manually. The convention is to use a node attribute with value 0 or 1 to identify the sets each node belongs to. The functions in this package do not check that the node set is actually correct nor that the input graph is actually bipartite.

For bipartition computing, the Kernighan-Lin bipartition algorithm is a good choice. This algorithm partitions a network into two sets by iteratively swapping pairs of nodes to reduce the edge cut between the two sets.

Finding the largest clique in a graph is NP- complete problem, so the find_cliques() function probably has an exponential running time, however it returns all maximal cliques in an undirected graph.

K-Clique. A k-clique community is the union of all cliques of size k that can be reached through adjacent k-cliques.

Modularity-based communities: greed_modularity_communities()

Greedy modularity maximization begins with each node in its own community and repeatedly joins the pair of communities that lead to the largest modularity until no further increase in modularity is possible.

4.4 Sixtep software

The software has been released in 2007 by network theory researchers and CRM advisors. It is able to load a graph and visualize it. After adding a graph, an attribute file can be attached to it, provide information about the nodes with it. For example, if the nodes are representing people, such information like name, age, city can be added.

Several algorithms are implemented to clusterize or detect communities such as the Newman-Girvan algorithm or the Markov chain model. The UI representation of the graph is user friendly; the location of the nodes can be easily modified by clicking one by one or select a targeted area. Several built-in functions help to make the graph more interpretable. The user can select unique modules like clusters or communities and display only the selected ones.

3 algorithms have been used in this thesis to classify the nodes:

- Markov chain clustering
- Maximized modularity
- Community detection

The source code of the software cannot be accessed; therefore, the software is not expandable, cannot be further developed. However, it is possible to get the calculated cluster and community data with the export function. The exported data needed further processing, since the data structure was not in the right format to manufacture the NetworkX Graph object. An import function processes the data from the export then a factory function provides the Graph object that is needed. After that, a colormap is assigned to the nodes by the exported labels. The result can be compared with the built in NetworkX functions via modularity and/or the number of the used colors.

Sadly, there is no information provided in the software about the edges between the clusters, however valuable data can be found while exploring that area. See more at graph condensation.

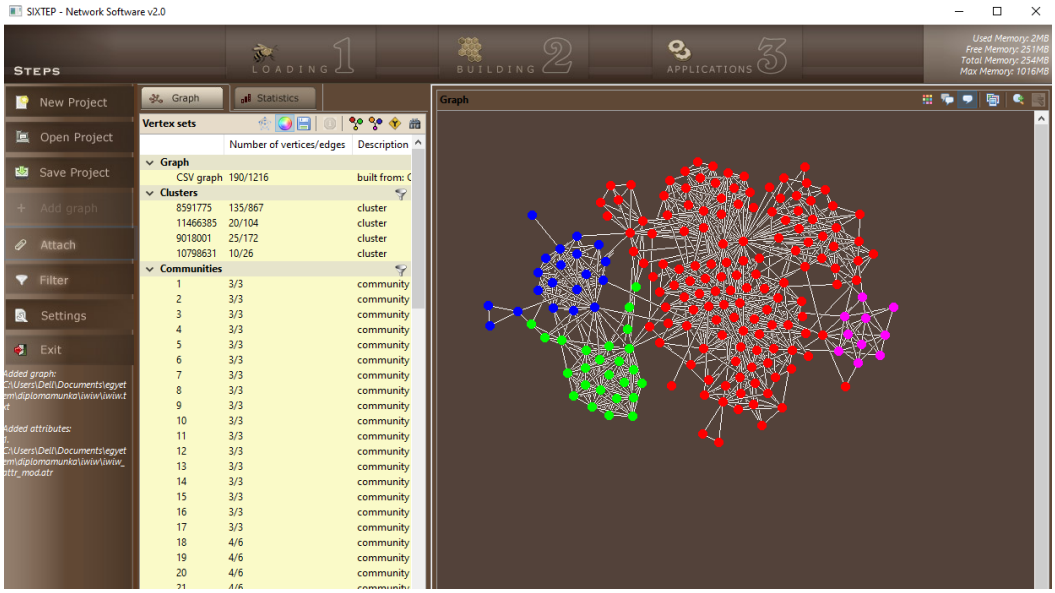


Figure 4.1 Clustering result in Sixtep software

4.5 Graph Databases

A graph database is defined as a specialized, single-purpose platform for creating and manipulating graphs. Graphs contain nodes, edges and properties, all of which are used to represent and store data in a way that relational databases are not equipped to do. Graph analytics is another commonly used term and it refers specially to the process of analyzing data in a graph format using data points as nodes and relationships as edges. Graph analytics requires a database that can support graph formats. This could be a dedicated database, or a converged database that supports multiple data models, including graphs.

There are two popular models of graph databases: property graphs and RDF graphs. The property graph focuses on analytics and querying, while the RDF graph emphasizes data integration. Both types of graphs consist of a collection of points (nodes) and the connection between those points (edges). But there are differences as well.

Property Graphs.

These types of graphs are used to model relationships among data and they enable query and data analytics based on these relationships. A property graph has nodes that can contain detailed information about a subject and edges that denote the relationship between nodes. The nodes and edges can have attributes, called properties, with which they are associated. Because they are so versatile, property graphs are used in a broad range of industries and sectors, such as finance, manufacturing, public safety, retail and many others.

RDF graphs.

RDF stand for Resource Description Framework. These graphs conform to a set of W3C standard designed to represent statements and are best for representing complex metadata and master data. They are often used for linked data, data integration and knowledge graphs. They can represent complex concepts in a domain, or provide rich semantics and inferencing on data. In the RDF model a statement is represented by three elements: two nodes connected by an edge reflecting the subject, predicate and object of a sentence – this is known as an RDF triple. Every node and edge is identified by a unique URI, or Unique resource Identifier. The RDF model provides a way to publish data in a standard format with well-defined semantics, enabling information exchange. Government statistics agencies, pharmaceutical companies and healthcare organizations have adopted RDF graphs widely.

How graph databases work.

Graphs and graph databases provide graph models to represent relationships in data. They allow users to perform "traversal queries" based on connections and apply graph algorithms to find patterns, paths, communities, influencers, single points of failure, and other relationships, which enable more efficient analysis at scale against massive amounts of data. The power of graphs is in analytics, the insights they provide, and their ability to link disparate data sources.

When it comes to analyzing graphs, algorithms explore the paths and distance between the vertices, the importance of the vertices, and clustering of the vertices. For example, to determine importance algorithms will often look at incoming edges, importance of neighboring vertices, and other indicators.

Graph algorithms—operations specifically designed to analyze relationships and behaviors among data in graphs—make it possible to understand things that are difficult to see with other methods. When it comes to analyzing graphs, algorithms explore the paths and distance between the vertices, the importance of the vertices, and clustering of the vertices. The algorithms will often look at incoming edges, importance of neighboring vertices, and other indicators to help determine importance. For example, graph algorithms can identify what individual or item is most connected to others in social networks or business processes. The algorithms can identify communities, anomalies, common patterns, and paths that connect individuals or related transactions.

Advantages of graph databases.

The graph format provides a more flexible platform for finding distant connections or analyzing data based on things like strength or quality of relationship. Graphs let you explore and discover connections and patterns in social networks, IoT, big data, data warehouses, and also complex transaction data for multiple business use cases including fraud detection in banking, discovering connections in social networks, and customer 360. Today, graph databases are increasingly being used as a part of data science as a way to make connections in relationships clearer.

Because graph databases explicitly store the relationships, queries and algorithms utilizing the connectivity between vertices can be run in subseconds rather than hours or days. Users don't

need to execute countless joins and the data can more easily be used for analysis and machine learning to discover more about the world around us.

Graph databases are an extremely flexible, extremely powerful tool. Because of the graph format, complex relationships can be determined for deeper insights with much less effort. Graph databases generally run queries in languages such as Property Graph Query Language (PGQL). The example below shows the same query in PGQL and SQL.

Because graphs emphasize relationships between data, they are ideal for several different types of analyses. In particular, graph databases excel at:

- Finding the shortest path between two nodes
- Determining the nodes that create the most activity/influence
- Analyzing connectivity to identify the weakest points of a network
- Analyzing the state of the network or community based on connection distance/density in a group

5 Graphs, Data sources

5.1 Social

5.1.1 IWIW

Number of nodes: 190

Number of edges: 1216

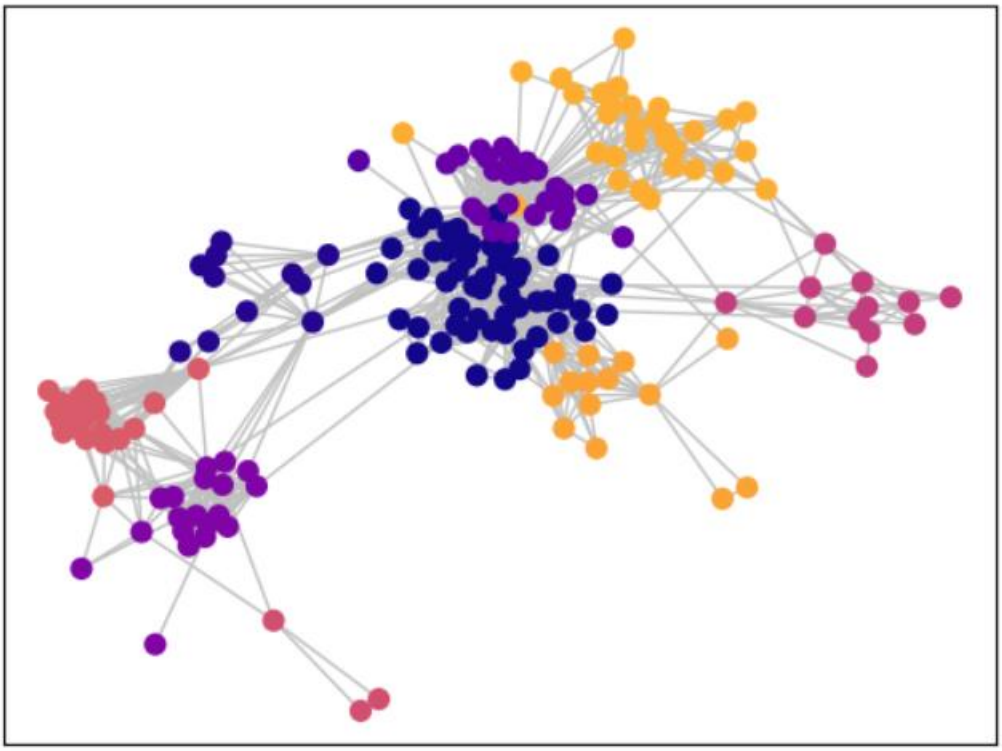


Figure 5.1 IWIW social connection graph

The graph above represents the connections of a person on the IWIW social platform. This is the only data which has not been anonymized. The nodes represent people and the edges the friendships. An attribute file is available for this graph, holding information about names, locations and other valuable information. With this data, a possibility was given to check the efficiency of the implemented community detector algorithms. The results showed that the communities appeared in certain period of time.

5.1.2 Facebook

This dataset is collected from the website of Stanford University. Despite the data is anonymized, this source opened a possibility to handle large social type of graphs and examine the structure of them. Based on what is known about the IWIW graph, the same patterns can be deduced in other anonymized graphs like the ones collected from Facebook or Twitter.

The datasets from Stanford are using a wide range of delimiters to separate the nodes of the edges, so most of them are not eligible to open with the Sixtep software that has been provided to use during the research. The application had to be implemented that way to be able to use any delimiter given by the user. This parameter comes from outside, and not depends on the implementation.

Number of nodes: 128

Number of edges: 100

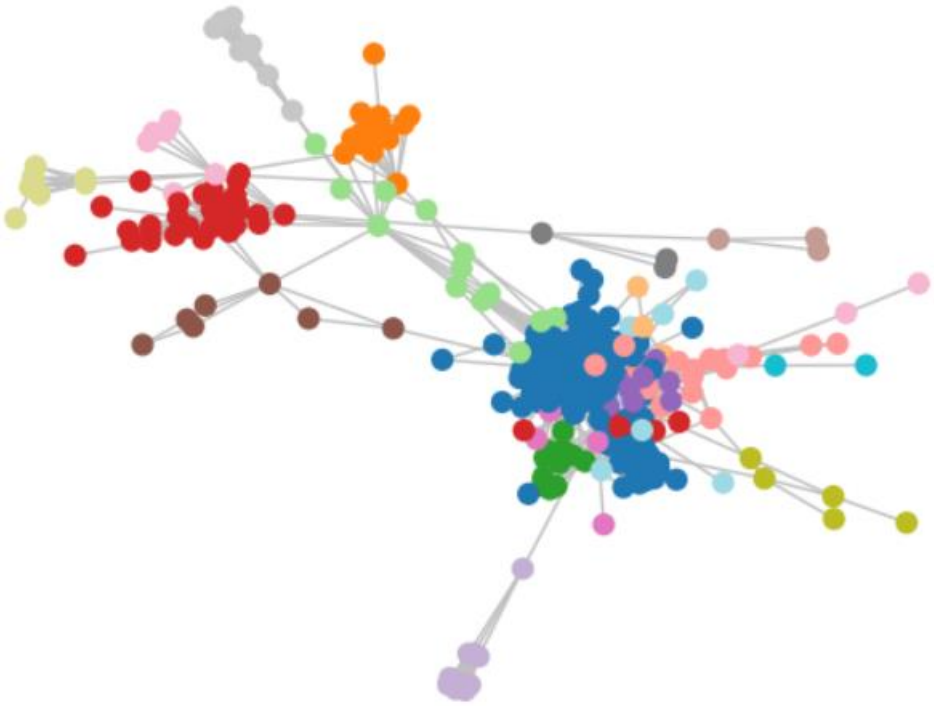


Figure 5.2 Newman-Girvan run on the slice of a FB graph

5.1.3 Zachary

A social network of a karate club was studied by Wayne W. Zachary. The network became a popular example of community structure in networks after its use by Michelle Girvan and Mark Newman. It captures 34 members of a karate club, documenting links between pairs of members who interacted outside the club. During the study a conflict arose between the administrator and instructor, which led to the split of the club into two. Half of the members formed a new club around the instructor; members from the other part found a new instructor or gave up karate. Based on collected data Zachary correctly assigned all but one member of the club to the groups they actually joined after the split. The coloring of the graph represents the two new community. [5]

Number of nodes: 34

Number of edges: 78

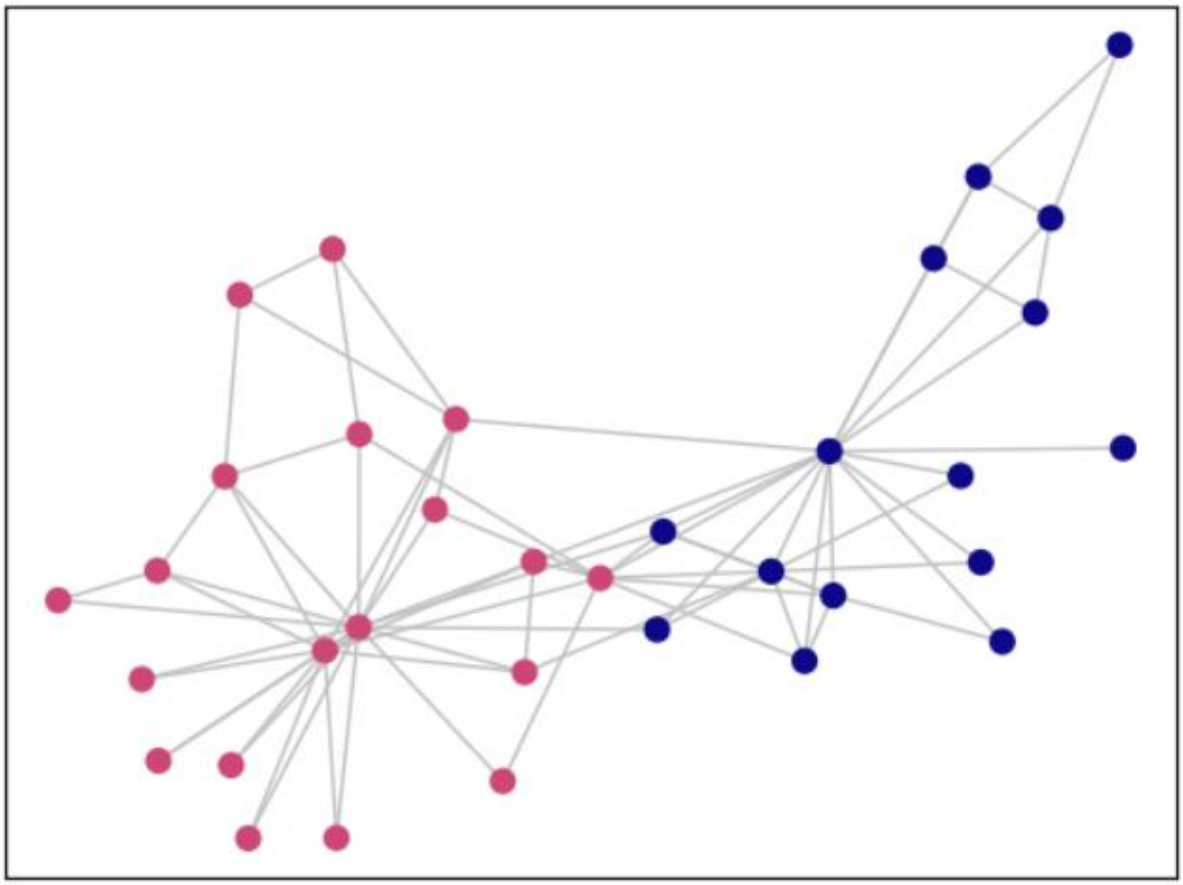


Figure 5.3: The well-known karate club community is divided into 2 main part due to a conflict of interest.

5.2 Transaction

5.2.1 OTP graph

This graph has been provided by the Department of Computational Optimization from the University of Szeged and despite the data is covered with a total anonymization, it is strictly confidential. The graph originally has been cut into 3 pieces and even smaller slices were made from the main partitions to make the detection easier even to the naked eye. It is not possible to decide obviously whether the graph is social or transactional type, there are areas with each kind. See layers after condensation.

Main partitions:

Relevant first quarter year:

- Nodes: 34992
- Edges: 72440

Relevant second quarter year:

- Nodes: 37008
- Edges: 79098

Relevant third quarter year:

- Nodes: 39155
- Edges: 86110

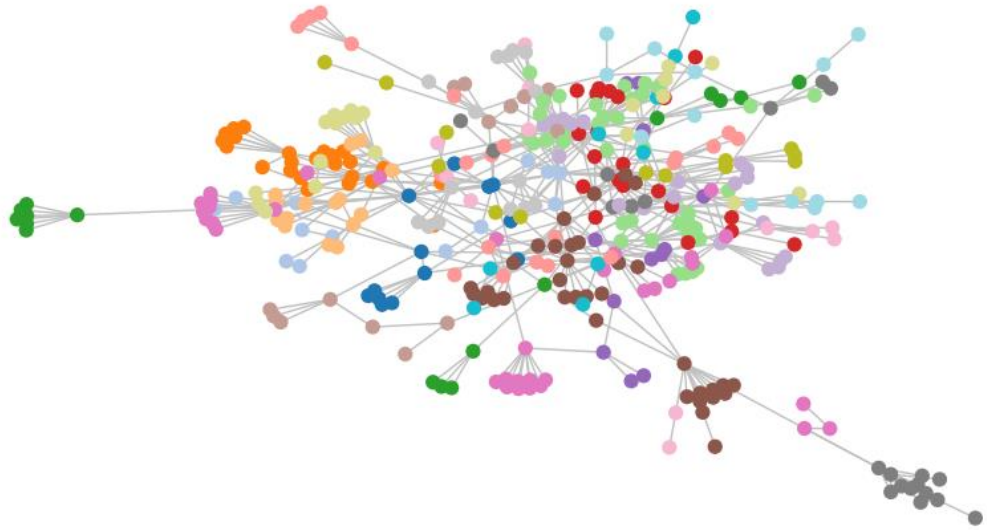


Figure 5.4 Largest connected components clustered by MCL in a slice of OTP first quarter graph

5.2.2 Word-Graph

The graph based on a word association game: People have been asked what comes to mind about the given words. It is a fairly large graph with a wide range of network structures. It is similar to random graphs, however both social and transaction patterns can be detected while observing the structures of inner layers.

6 Algorithms, experiences

6.1 Community detection algorithms

6.1.1 Newman-Girvan

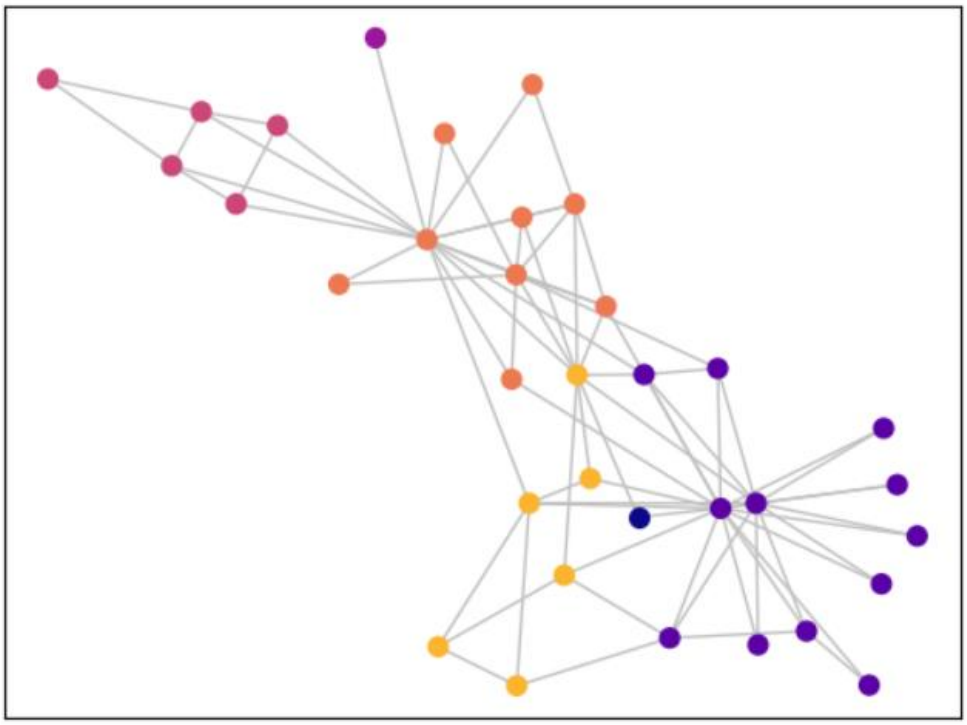


Figure 6.1 NG communities in zachary-graph

The Girvan-Newman algorithm for the detection and analysis of community structure relies on the iterative elimination of edges that have the highest number of shortest paths between nodes passing through them. By removing edges from the graph one-by-one, the network breaks down into smaller pieces, so-called communities. The algorithm was introduced by Michelle Girvan and Mark Newman. The idea was to find which edges in a network occur most frequently between other pairs of nodes by finding edges betweenness centrality. The edges joining communities are then expected to have a high edge betweenness. The underlying community structure of the network will be much more fine-grained once the edges with the highest betweenness are eliminated which means that communities will be much easier to spot. [3]

In the NetworkX implementation the Girvan-Newman algorithm can be divided into four main steps:

1. For every edge in a graph, calculate the edge betweenness centrality.
2. Remove the edge with the highest betweenness centrality.
3. Calculate the betweenness centrality for every remaining edge.

4. Repeat steps 2-4 until there are no more edges left.

6.1.2 Maximized modularity

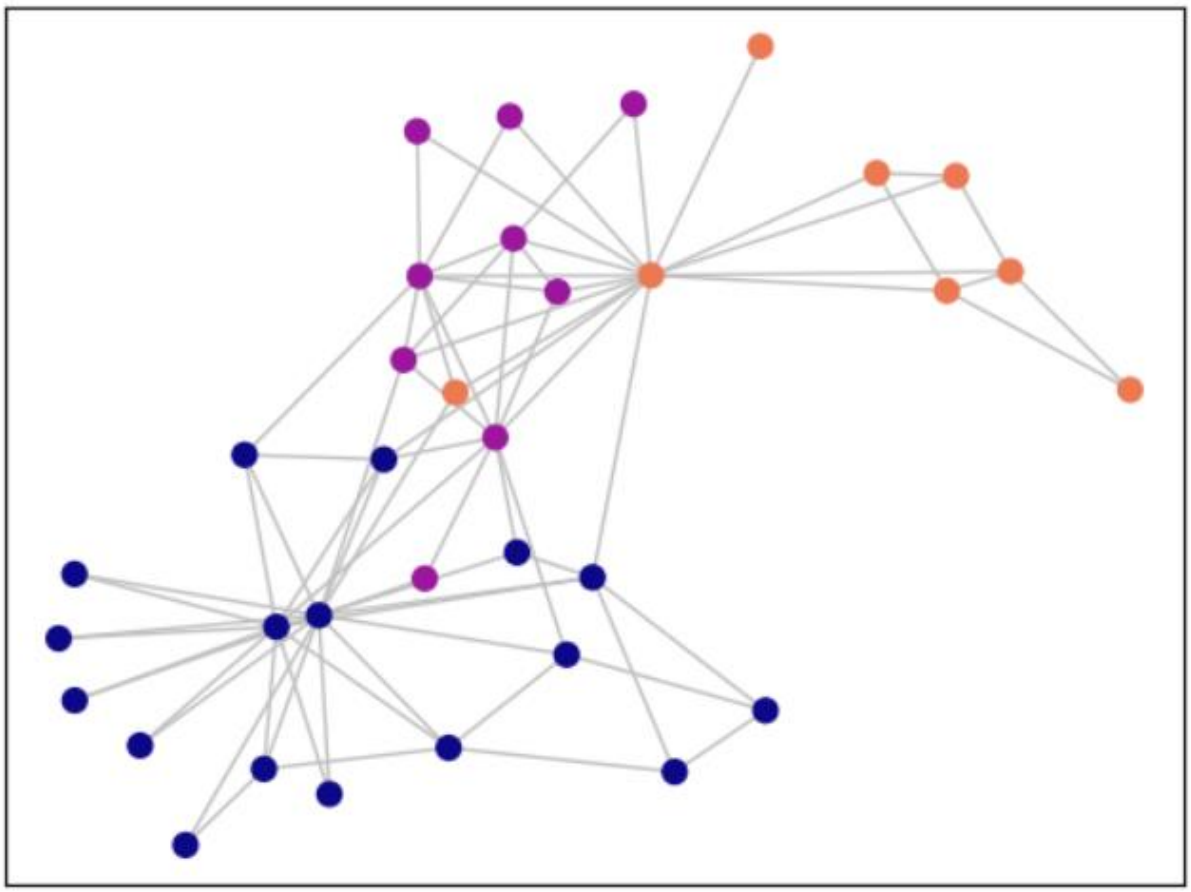


Figure 6.2 Maximized modularity in zachary-graph

This algorithm finds communities in graph using Clauset-Newman-Moore greedy modularity maximization. This method currently does not consider edge weights. Greedy modularity maximization begins with each node in its own community and joins the pair of communities that most increases modularity until no such pair exists. [3]

Source code:

https://networkx.org/documentation/stable/_modules/networkx/algorithms/community/modularity_max.html

Modularity

One of the most sensitive detection methods is optimization of the quality function known as modularity over the possible divisions of a network, but direct application of this method using,

for instance, simulated annealing is computationally costly. A community structure in a network corresponds to a statistically surprising arrangement of edges, can be quantified using the measure known as modularity [9]. The modularity is, up to a multiplicative constant, the number of edges falling within groups minus the expected number in an equivalent network with edges placed at random. The modularity can be either positive or negative, with positive values indicating the possible presence of community structure [10].

Modularity is defined in [11] as

$$Q = \frac{1}{2m}\sum_{ij}\left(A_{ij} - \gamma\frac{k_i k_j}{2m}\right)\delta(c_i, c_j)$$

Where m is the number of edges, A is the adjacency matrix of G, k_i is the degree of i, γ is the resolution parameter, and $\delta(c_i, c_j)$ is 1 if i and j are in the same community, else 0.

According to [12] (and verified some algebra) this can be reduced to

$$Q = \sum_{c=1}^{n}\left[\frac{L_c}{m} - \gamma\left(\frac{k_c}{2m}\right)^2\right]$$

Where the sum iterates over all communities c, m is the number of edges, L_c is the number of intra-community links for community c, k_c is the sum of degrees of the nodes in community c, and γ is the resolution parameter.

The resolution parameter sets an arbitrary tradeoff between intra-group edges and inter-group edges. More complex grouping patterns can be discovered by analyzing the same network with multiple values of gamma and then combining the results [13]. That said, it is very common to simply use gamma=1. More on the choice of gamma is in [14].

This NetworkX version has been used in the community detector repository. The parameters are the following: G represent the NetworkX graph. Communities are a list or iterable of set of nodes. These node sets must represent a partition of G's nodes. Weight is an edge attribute that holds the numerical value used as a weight. It is an optional parameter, if the value is None or an edge does not have that attribute, then that edge has weight 1. Resolution is an optional parameter as well. If resolution is less than 1, modularity favors larger communities. Greater than 1 favors smaller communities. The function returns Q, the modularity of the partition. In case of the communities are not a partition of G, the function raises NotAPartition exception.

The second formula is the one actually used in calculation of the modularity. For directed graphs the second formula replaces k_c with $k_c^{in} k_c^{out}$.

6.1.3 Cliques

Clique problem.

In computer science, the clique problem is the computational problem of finding cliques (subsets of vertices, all adjacent to each other, also called complete subgraphs) in a graph. It has several different formulations depending on which cliques, and what information about the cliques should be found. Common formulations of the clique problem include finding a maximum clique (a clique with the largest possible number of vertices), finding a maximum weight clique in a weighted graph, listing all maximal cliques (cliques that cannot be enlarged) and solving the decision problem of testing whether a graph contains a clique larger than a given size.

The clique problem arises in the following real-world setting. Consider a social network, where the graph's vertices represent people, and the graph's edges represent mutual acquaintance. Then a clique represents a subset of people who all know each other, and algorithms for finding cliques can be used to discover these groups of mutual friends. Along with its applications in social networks, the clique problem also has many applications in bioinformatics, and computational chemistry.

Most versions of the clique problem are hard. The clique decision problem is NP-complete (one of Karp's 21 NP-complete problems). The problem of finding the maximum clique is both fixed-parameter intractable and hard to approximate. Listing all maximal cliques may require exponential time as there exist graphs with exponentially many maximal cliques. Therefore, much of the theory about the clique problem is devoted to identifying special types of graphs that admit more efficient algorithms, or to establishing the computational difficulty of the general problem in various models of computation.

To find a maximum clique, one can systematically inspect all subsets, but this sort of brute-force search is too time-consuming to be practical for networks comprising more than a few dozen vertices. Although no polynomial time algorithm is known for this problem, more efficient algorithms than the brute-force search are known. For instance, the Bron–Kerbosch

algorithm can be used to list all maximal cliques in worst-case optimal time, and it is also possible to list them in polynomial time per clique. [19]

Clique maximization:

For each node n, a maximal clique for n is a largest complete subgraph containing n. The largest maximal clique is sometimes called the maximum clique.

This function returns an iterator over cliques, each of which is a list of nodes. It is an iterative implementation, so should not suffer from recursion depth issues.

This function accepts a list of nodes and only the maximal cliques containing all of these nodes are returned. It can considerably speed up the running time if some specific cliques are desired.

A list output of the function find_cliques(G) has been used to obtain all maximal cliques. However, in the worst-case scenario, the length of this list can be exponential in the number of nodes in the graph. This function avoids storing all cliques in memory by only keeping current candidate node lists in memory during its search.

This implementation is based on the algorithm published by Bron and Kerbosch (1973) [6], as adapted by Tomita, Tanaka and Takahashi (2006) [7] and discussed in Cazals and Karande (2008) [8].

This algorithm ignores self-loops and parallel edges, since cliques are not conventionally defined with such edges.

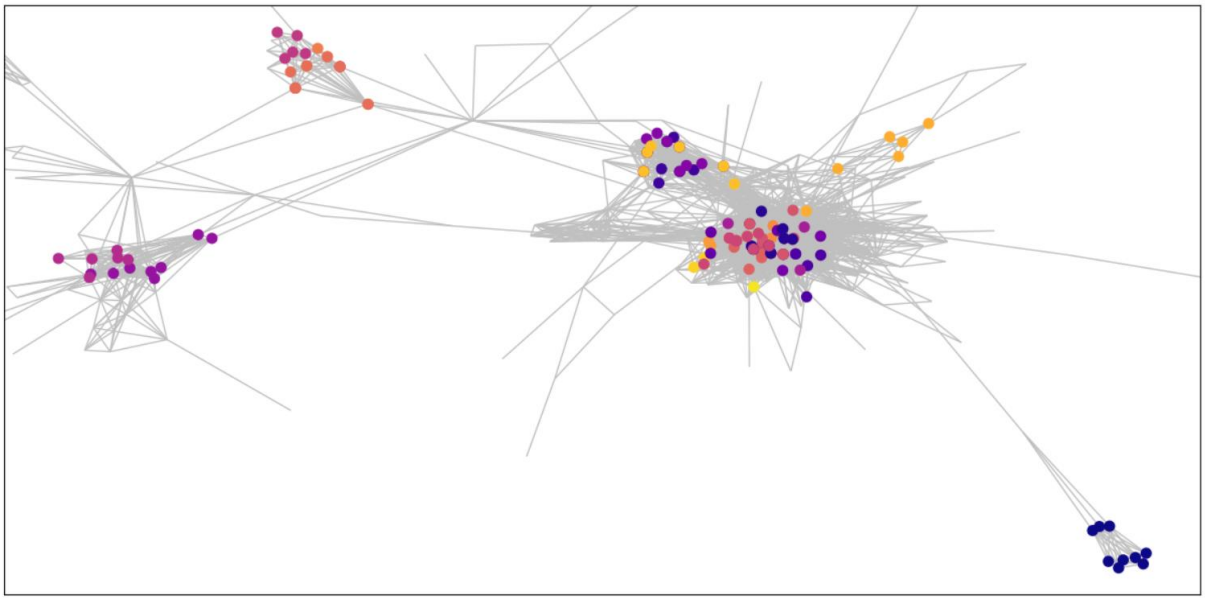


Figure 6.3 It is important to determine the size of the cliques should be detected. For example, a clique with size 3 has lower importance than 5 or above.

6.2 Clustering algorithms on transaction networks

6.2.1 Markov-chain

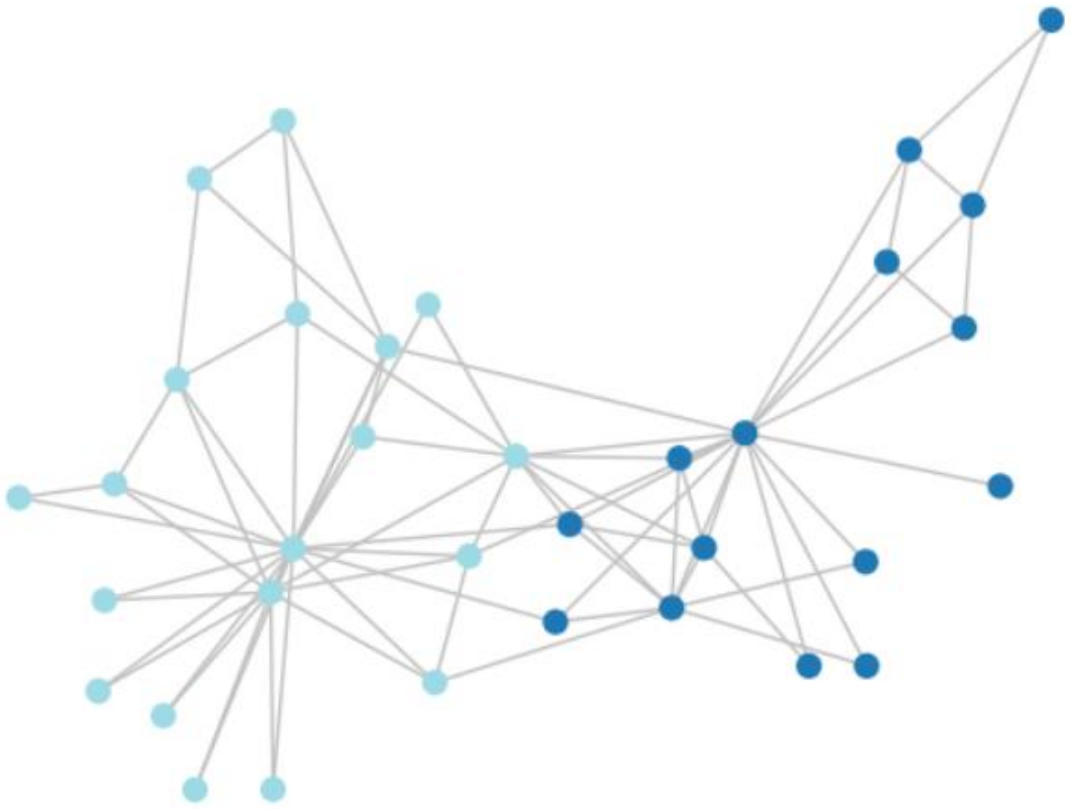


Figure 6.4 MCL in zachary-graph

The MCL algorithm is short for the Markov Cluster Algorithm, a fast and scalable unsupervised cluster algorithm for graphs (also known as networks) based on simulation of (stochastic) flow in graphs. The algorithm was invented/discovered by Stijn van Dongen at the Centre for Mathematics and Computer Science (also known as CWI) in the Netherlands. [4]

6.2.2 H-avoiding coloring

Embeddedness of bipartite subgraphs in transaction networks

Certain bipartite graphs, for example pollinator networks or trade networks suggest the presence of different structures, like the notion of embeddedness. The vertices of each color class can be ordered and the smaller ranked vertex neighbourhood contains the neighbourhood of any higher ranked one. A binary matrix *A* is fully nested if its rows and columns can be reordered such that

the ones are in echelon form. Let G_A be the bipartite graph whose adjacency matrix is A. Then A being fully nested is equivalent to G_A satisfying embeddedness.

Let X (the columns) and Y (the rows) be the bipartition of G_A. The matrix A and the graph G_A are each said to be k-nested with respect to X if X can be partitioned as $X_1, \ldots, X_k$ such that all subgraphs spanned by (X_i, Y) are fully nested for $i = 1, \ldots, k$. The quantity of interest for any G_A is smallest k for which G_A is k-nested.

Given a "forbidden graph" H and an integer k, an H-avoiding k-coloring of a graph G is a k-coloring of the vertices of G such that no maximal H-free subgraph of G is monochromatic.

A monochromatic graph is a colored graph (either vertex-colored or edge-colored, depending on the context) in which each of the vertices or edges is assigned the same color.

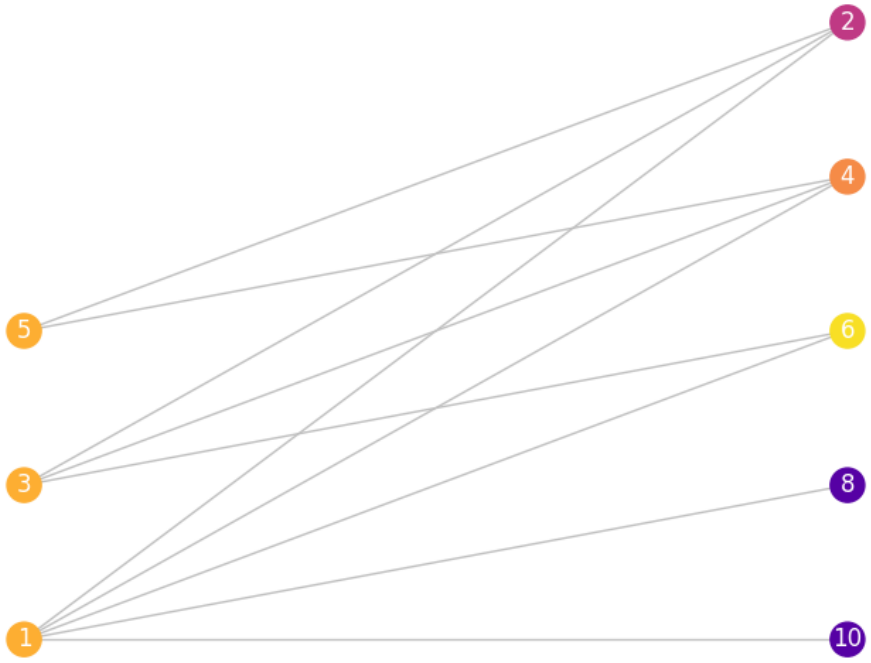


Figure 6.5. Proper 2K2 avoiding coloring of a bipartite graph

Colormap: {'1': 0.82, '2': 0.44999999999999996, '4': 0.72, '6': 0.94, '8': 0.15000000000000002, '10': 0.15000000000000002, '3': 0.82, '5': 0.82}

A new kind of clustering of general (that is, not necessarily bipartite) transaction graphs has been presented via a certain class of proper colorings. The clusters are the color classes, since no edges are desired inside a cluster. The structure of the edges is restricted between the pairs of classes. The above examples suggest that in some cases there should be a fully nested or, equivalently, embeddedness relation among any two-color classes. This notion is generalized to an arbitrary host graph G and a forbidden bipartite subgraph H as follows.

Definition 1. Fix a bipartite graph H. A proper coloring of a graph G is an H-avoiding coloring if the union of any two-color classes spans an induced H-free graph. Let $\chi_H(G)$ be the minimum number of colors in an H-avoiding coloring of G.

Observation 2. For any graphs H and G, $\chi(G) \leq \chi_H(G)$. If G is H-free, then $\chi(G) = \chi_H(G)$

The computation of $\chi_H(G)$ is NP-hard for some graphs and polynomially computable for others. The most interesting case is, when $H = 2K_2$, gives back embeddedness as described above. For these generalized chromatic numbers some theoretical extremal results have been derived as well as results on complexity.

6.3 Other algorithms

6.3.1 Condensation

Condensed graphs

Each community/cluster has been contracted to single node. If there is a path between the modules, the algorithm puts an edge between the nodes. The algorithm works well, if the source graph is strongly connected. Works well on iwiw graph with girvan clustered communities. Clusters classified by MCL failes in condensation function. No vertices displayed. Works well with greedy modularity as well.

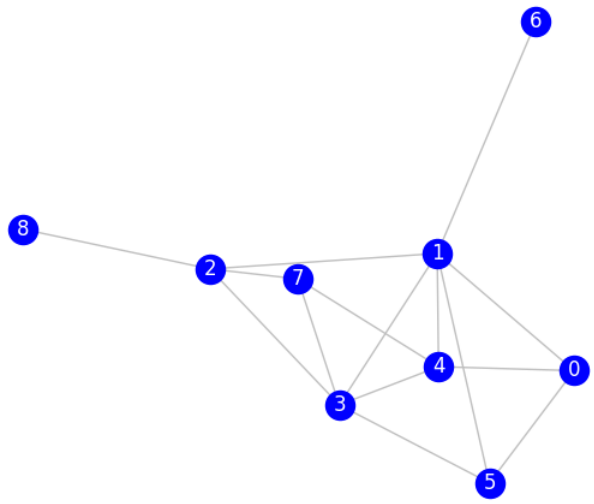


Figure 6.6

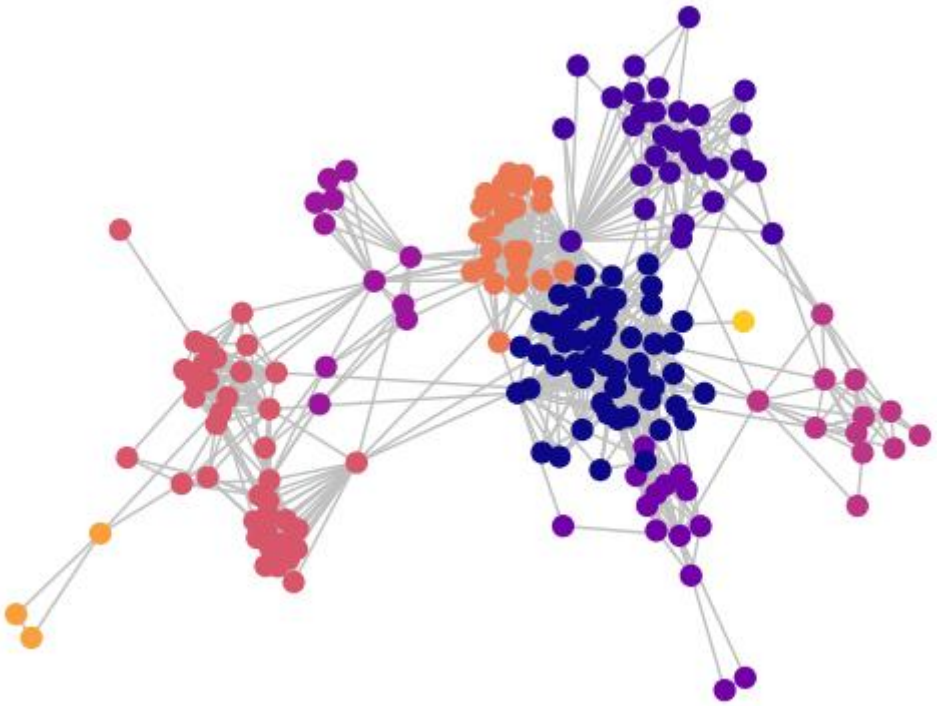


Figure 6.7 Communities in IWIW graph clustered by girvan

6.3.2 Small-worldness

A small-world network is a type of mathematical graph in which most nodes are not neighbors of one another, but the neighbors of any given node are likely to be neighbors of each other and most nodes can be reached from every other node by a small number of hops or steps. Specifically, a small-world network is defined to be a network where the typical distance L between two randomly chosen nodes (the number of steps required) grows proportionally to the logarithm of the number of nodes N in the network, that is:

$$L \propto Log\ N$$

while the clustering coefficient is not small. In the context of a social network, this results in the small world phenomenon of strangers being linked by a short chain of acquaintances. Many empirical graphs show the small-world effect, including social networks, wikis such as Wikipedia, gene networks, and even the underlying architecture of the Internet. [19]

Small-world functions are used for estimating the small-worldness of graphs. A small world network is characterized by a small average shortest path length and a large clustering

coefficient. Small-worldness is commonly measured with a coefficient, which compares the average clustering coefficient and shortest path length of a given graph against the same quantities for an equivalent random or lattice graph.

The coefficient is defined as:

$$\sigma = \frac{C/C_r}{L/L_r}$$

Where C and L are respectively the average clustering coefficient and average shortest path length of the graph. [3]

The graph is commonly classified as small-world if $\sigma > 1$.

6.3.3 Colormap

NetworkX draw() function needs a dictionary that describes which node to be colored by which color. There are predefined palettes provided by the MatPlotLib Python package.

Classes of colormaps:

Colormaps are often split into several categories based on their function (see, e.g., [Moreland]):

- Sequential: change in lightness and often saturation of color incrementally, often using a single hue; should be used for representing information that has ordering.
- Diverging: change in lightness and possibly saturation of two different colors that meet in the middle at an unsaturated color; should be used when the information being plotted has a critical middle value, such as topography or when the data deviates around zero.
- Cyclic: change in lightness of two different colors that meet in the middle and beginning/end at an unsaturated color; should be used for values that wrap around at the endpoints, such as phase angle, wind direction, or time of day.
- Qualitative: often are miscellaneous colors; should be used to represent information which does not have ordering or relationships.

Figure 6.8 source: https://matplotlib.org/stable/gallery/color/colormap_reference.html

Sequential:

For the Sequential plots, the lightness value increases monotonically through the colormaps. Some of the L^*values in the colormaps span from 0 to 100 (binary and the other grayscale), and others start around $L^* = 20$. Those that have a smaller range of L^* will accordingly have a smaller perceptual range. Note also that the L^* function varies amongst the colormaps: some are approximately linear in L^* and others are more curved. [20]

7 Measuring, results

Experiences:

The NetworkX implementations of the algorithms have been used for the python script. The code of the algorithms can be found in the coloring.py file. The colormap and the coloring logic is self-implemented and it can be found in the Utils.py file.

The Newman-Girvan algorithm is fairly slow on medium sized graphs, but the result is more accurate and all the nodes are classified.

7.1 Modularities, used colors

7.1.1 Network: Facebook, 0.edges dataset

Application	Community detector app			
Algo	NG	NG-comm*	Markov	Maximized modularity

Nr. of colors	2	30	24	8
Modularity	0.3615	(6) 0.2523 (20) 0.3729 (30) 0.4136 (35) 0.4129 (40) 0.4111 (50) 0.4043	-0.8080	**0.4429**

Application	Sixtep		
Algo	communities	Markov	Maximized modularity
Nr. of colors	208	7	8
Modularity	Cannot be calculated**	0.3306	**0.3891**

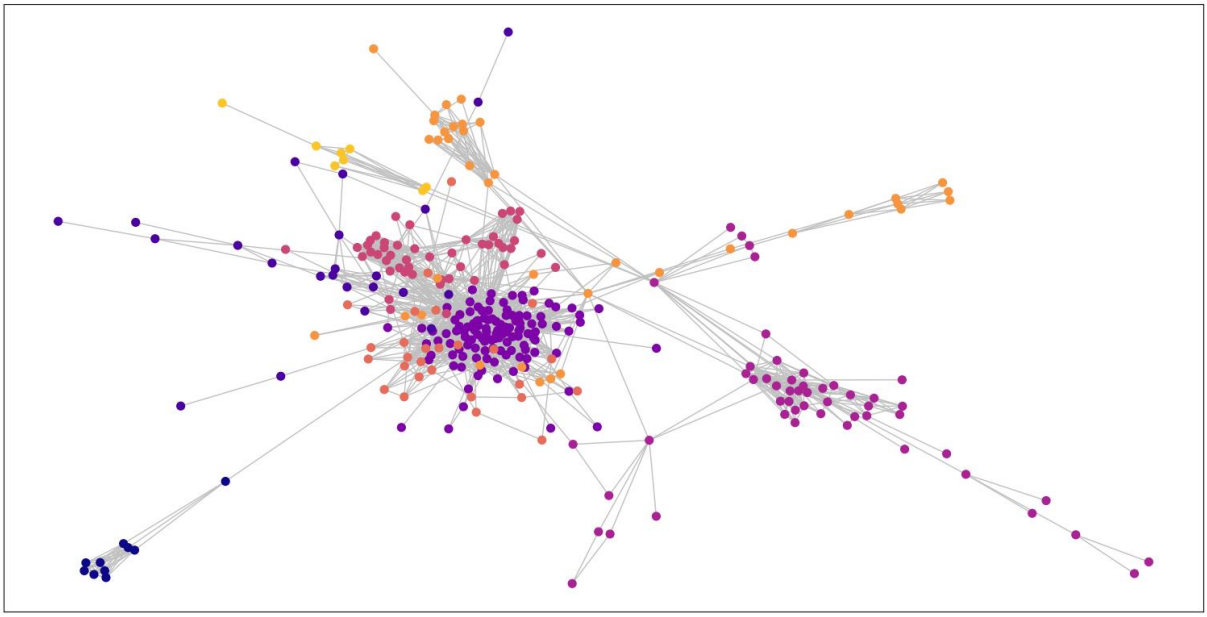


Figure 7.1: Facebook communities detected by greedy modularity algorithm

Number of nodes: 324

Number of edges: 2514

The value of the number of edges is different from the value given by the Sixtep software. The reason behind this is that the edges are represented in both direction in the text file. In conclusion, the value is double.

* The numbers before the modularity values represent the number of communities the algorithm took.

** The communities that has been provided by the Sixtep software are not a valid partition of the graph, therefore the NetworkX modularity function cannot be run against the data.

7.1.2 Network: transaction graphs of the OTP database

Application	Community detector app					
	Newman-Girvan		Markov		Greedy modularity	
	# colors	Modularity	# colors	Modularity	# colors	Modularity
OTP 100	10	0.6981	14	0.5413	9	0.7042
First 600 edges of first quarter year largest cc	19	**0.7570**	80	0.5725	15	0.7632
First quarter year 1k	19	0.7977	113	0.6182	20	0.7981
Middle slice	-	-	3810	0.5509	92	0.8227
total	-	-	0.3900		-	
Second quarter year	-	-	0.3794		-	
Third quarter year	-	-	0.3690		-	

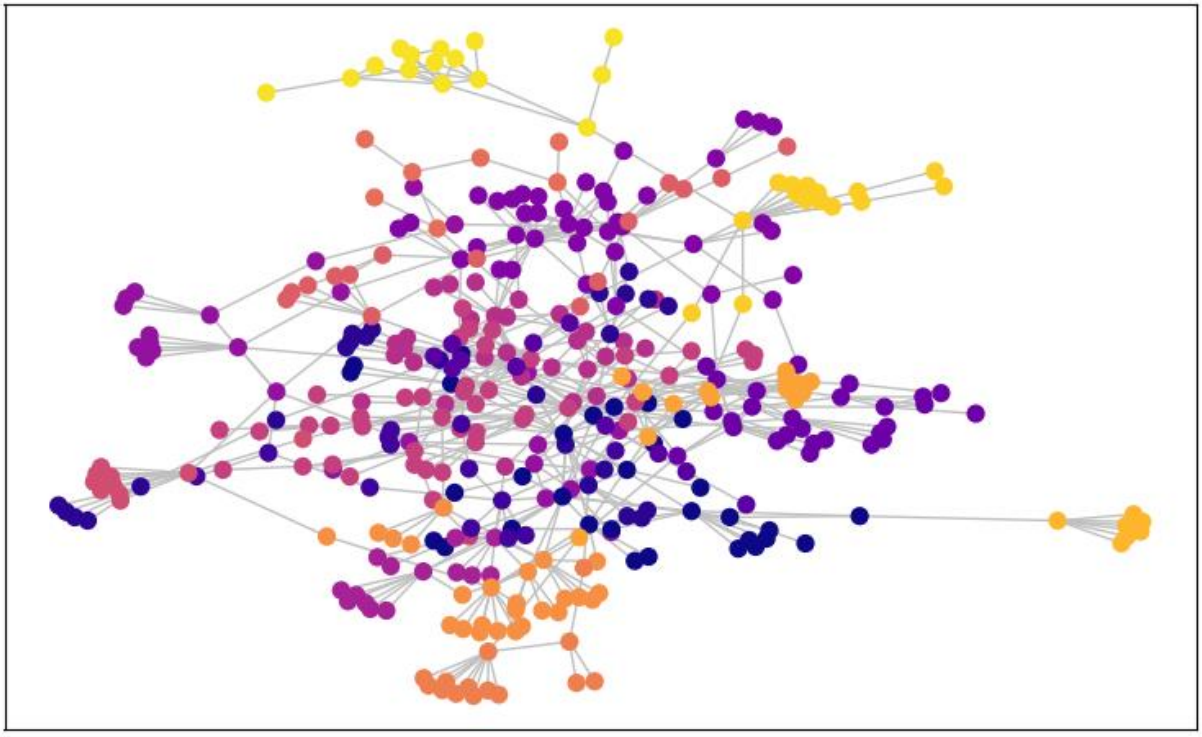
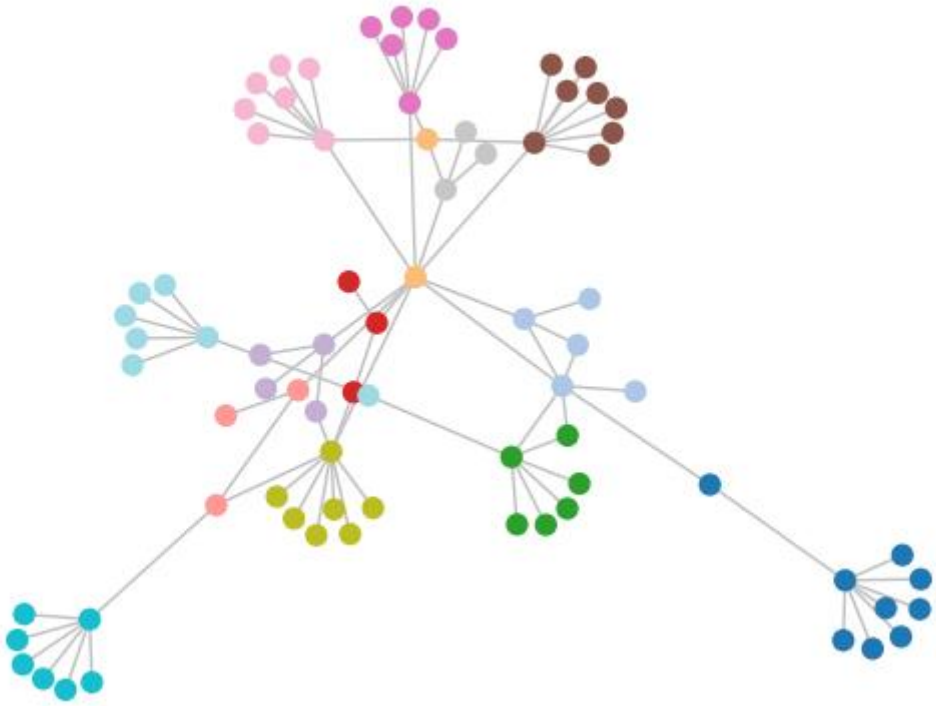


Figure 7.2 First 600 edges of first quarter year largest cc, 19 with 0.7570 mod



Application	Sixtep

	communities	Markov		Greedy modularity	
	# colors	# colors	Modularity	# colors	Modularity
OTP 100	2	9	0.6128	12	0.7580
First 600 edges of first quarter year largest cc	39	25	0.7346	19	0.7697
First quarter year 1k	56	39	0.7751	25	0.7965
Middle slice	862	1587	0.7632	879	0.8349
First quarter year	3583	1688	0.5792	-	
Second quarter year	3814	1587	0.5764	-	
Third quarter year	4303	1738	0.5707	-	

First quarter year:

- Number of nodes: 34992
- Number of edges: 72440, 76668 in Sixtep

Second quarter year:

- Number of nodes: 37008
- Number of edges: 79098, 83926 in Sixtep

Third quarter year:

- Number of nodes: 39155
- Number of edges: 86110, 91862 in Sixtep

OTP midslice of a first quarter year:

- Number of nodes:18032
- Number of edges: 25686

First 1K edges of first quarter year largest cc:

- Number of nodes:675

- Number of edges: 1000

First 600 edges of first quarter year largest cc:

- Number of nodes: 398
- Number of edges: 600 in Sixtep

First 100 edges of first quarter year largest cc:

- Number of nodes: 96
- Number of edges: 100

Is the OTP graph directed?

Usually, the difference between the value of the edges represents the direction of the graph. X -> Y and Y -> Y show that the nodes are mutually connected to one another. Difference between edges caused by graph vs. largest cc.The smaller OTP graph has been represented by 96 nodes, the first quarter year by 675 nodes and the middle slice is about 18032 nodes. The smaller one is a subset of the first quarter year. The total amount of nodes in this dataset is 34992. The graph of the second quarter year contains 37008 and the third one has 39155. At this size there is not much sense to visualize the graphs in 2D. No information can be gained with naked eyes or manual clustering.

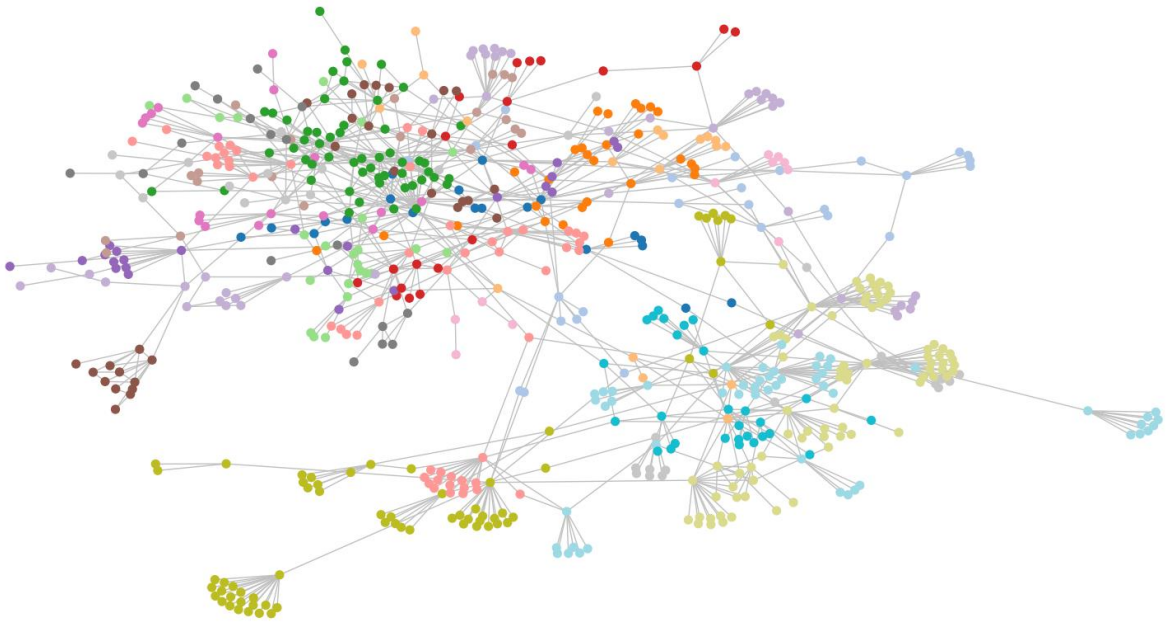


Figure 7.3. First quarter year of the OTP transactions clustered by Markov chain algorithm. 675 nodes have been colorized.

7.1.3 Wordgraph

Application	Community detector app					
	Newman-Girvan		Markov		Greedy modularity	
	# colors	Modularity	# colors	Modularity	# colors	Modularity
wordgraph	-	-	3786	0.2241	106	0.5349
1200	17	**0.8663**	52	0.7264	19	0.8661

Application	Sixtep				
	communities	Markov		Greedy modularity	
	# colors	# colors	Modularity	# colors	Modularity
wordgraph	null	236	0.4303	114	0.5303

- Number of nodes: 11381
- Number of edges: 31784

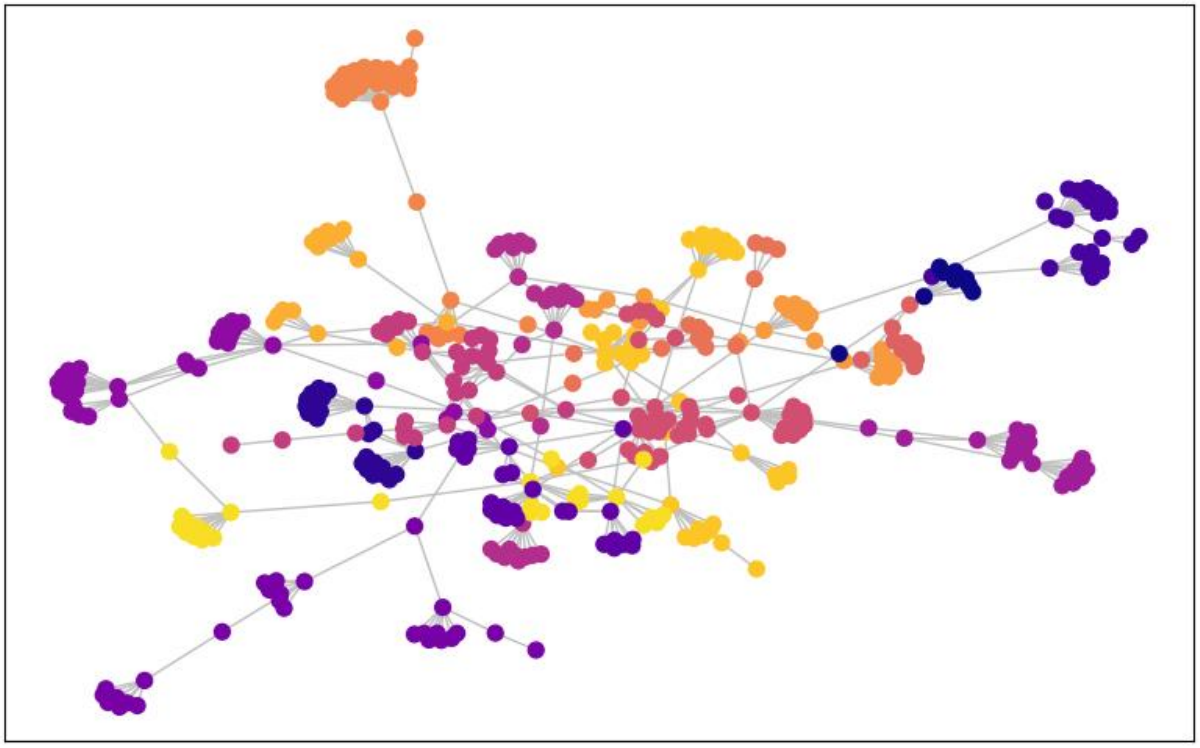


Figure 7.4 slice of wordgraph, colored by NG

OTP transaction leveling by condensation

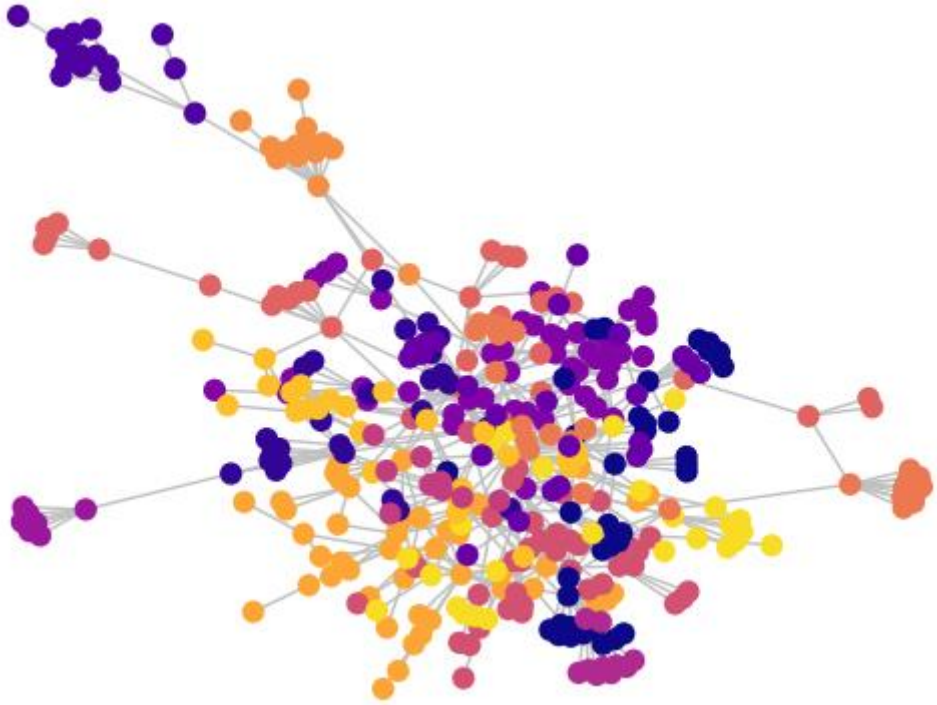


Figure 7.5 a slice of the OTP transactions from the first quarter year

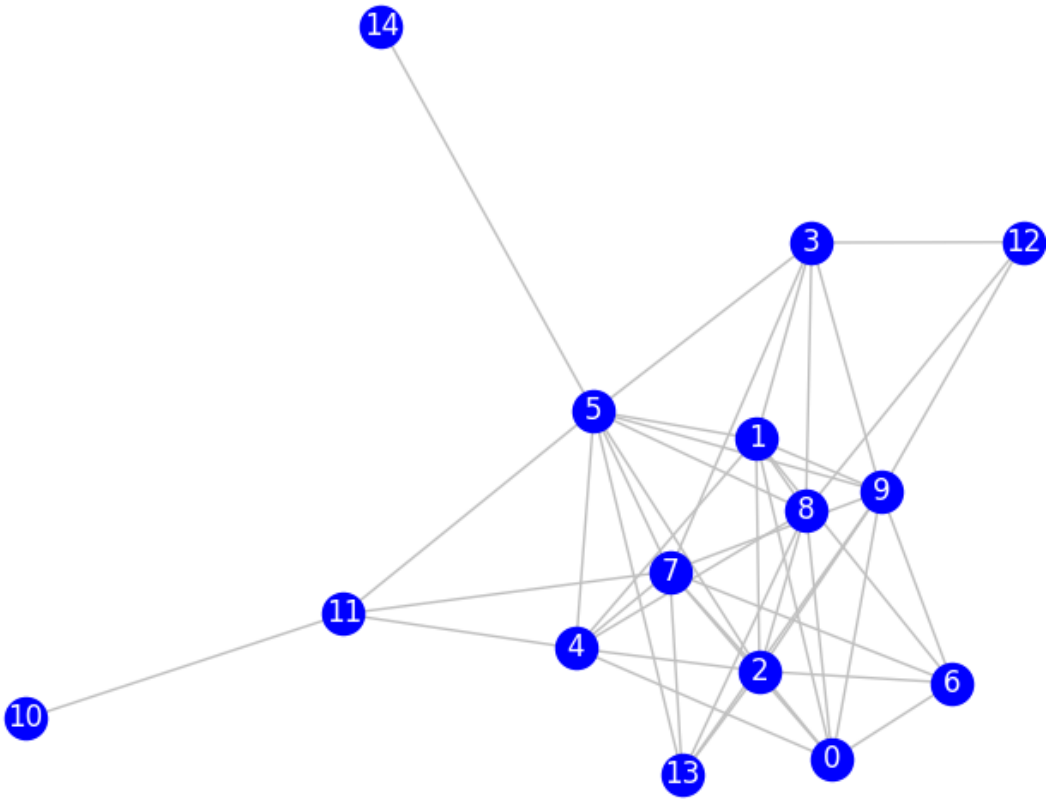


Figure 7.6 …and the condensated one

This is interesting because technical graphs usually represent tree-like structure (like said before), but this seems more like a social graph. Some of the layers in the otp graph look like communities. Originally the suppliers supposed to be in a competitive relation by restrict the edges between them.

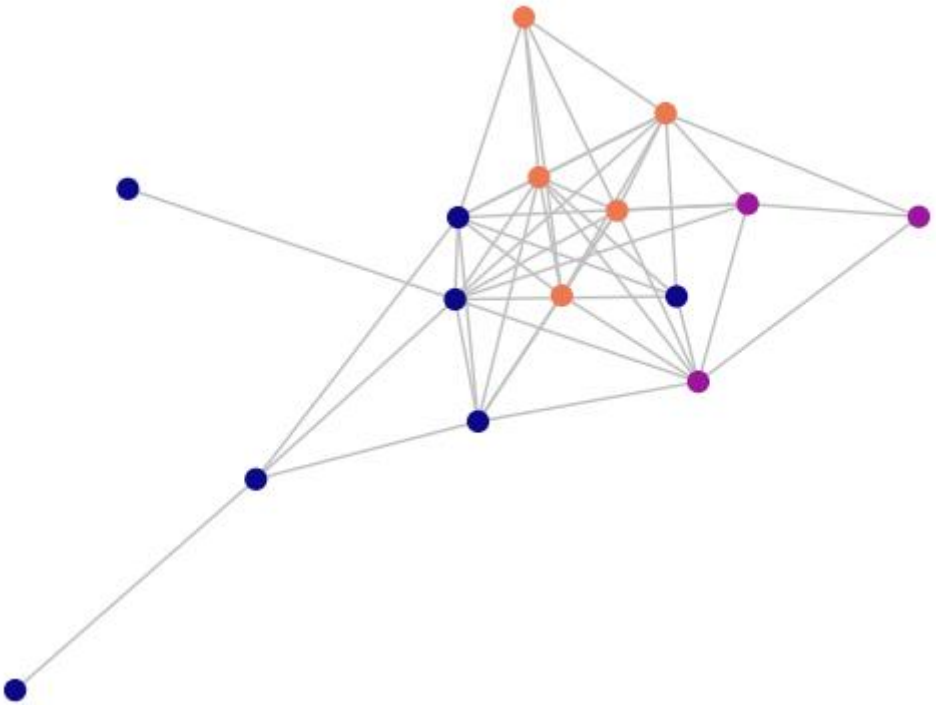


Figure 7.7 coloring of a condensed graph

But based on this, some cooperation can be supposed.

The modularity of the original graph is 0.7632.

7.2 Patterns

The following patterns are present in the word association graph.

7.2.1 Supplier – Customer

In this structure only a few nodes are present with a high edge number originated from them and many nodes are displayed with low degree number. In the word association graph, the meaning of the pattern is that there are words that cover a wide range of topic, for example: TOOL. It can represent basically anything like an IT tool, Neo4J Aura (graph database in cloud), or a mechanical tool like hammer. In a transactional environment, it could represent a

classic transactional pattern with a supplier – customer relationship, however since the OTP graph is anonymized, the necessary information is not provided to resolve the exact meaning.

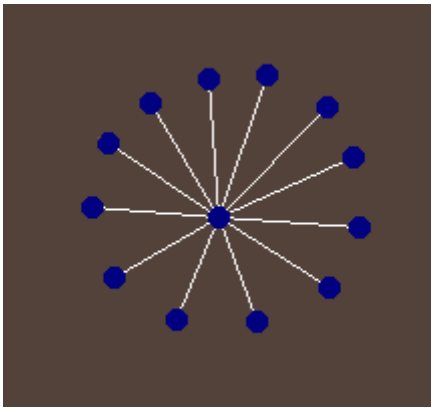


Figure 7.8

The following pattern has been unable to be identified, however it is quite common both in social and in transaction networks as well.

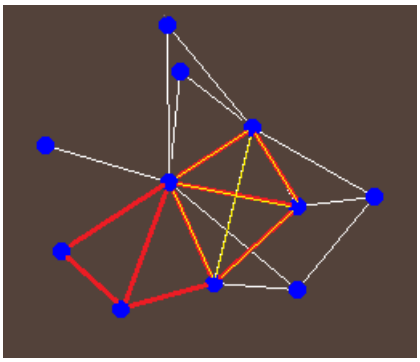


Figure 7.9

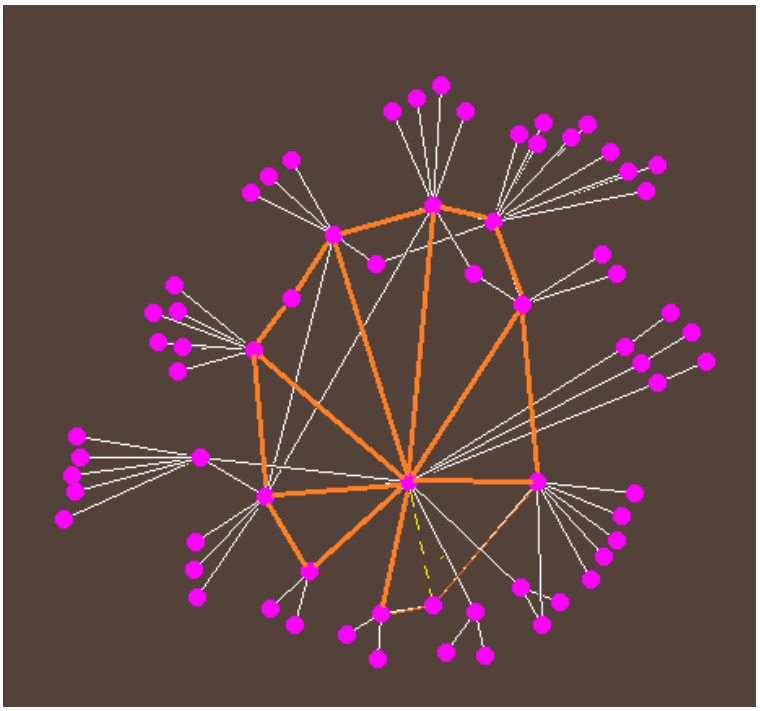
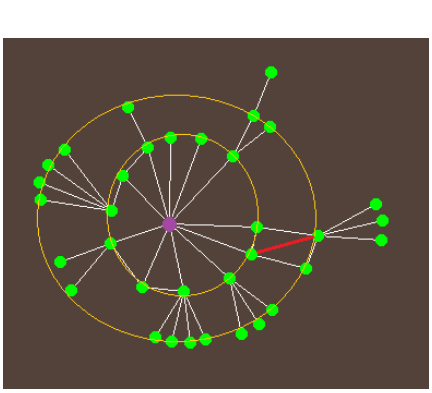
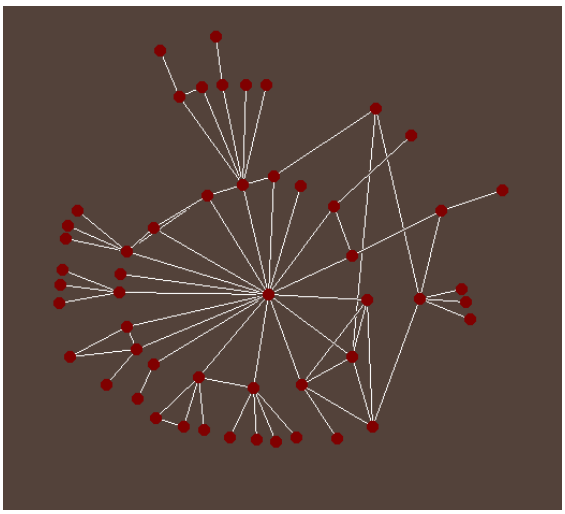


Figure 7.10

If the structure is extended, it appears that the so-called supplier vertices are connected with edges. As I looked through the graph, a circular, tree-like structure appeared.





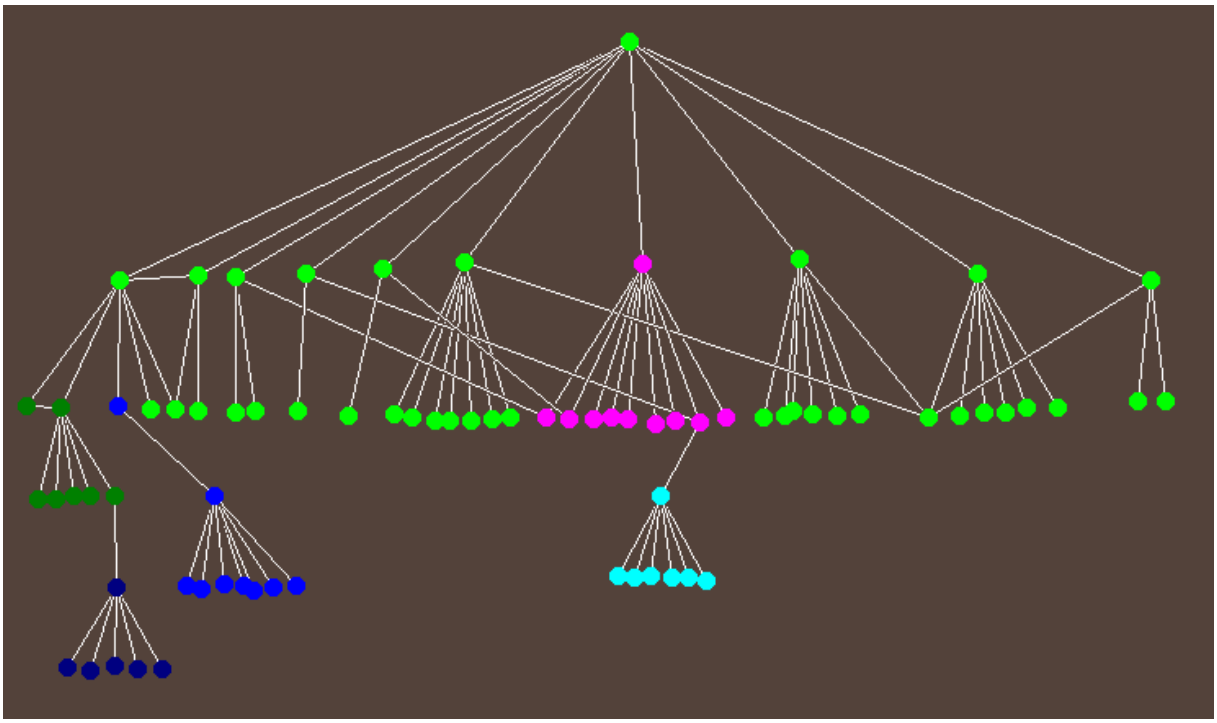


Figure 7.11

If we consider one supplier and the customers as one cluster, we can observe smaller communities among them. In the previous image we can see several triangles, but I discovered larger ones as I increased the number of layers I observed at the same time. Since the graph I am working with is anonymized, we do not have the necessary information to resolve the graph, thus I am still not sure what this structure represent.

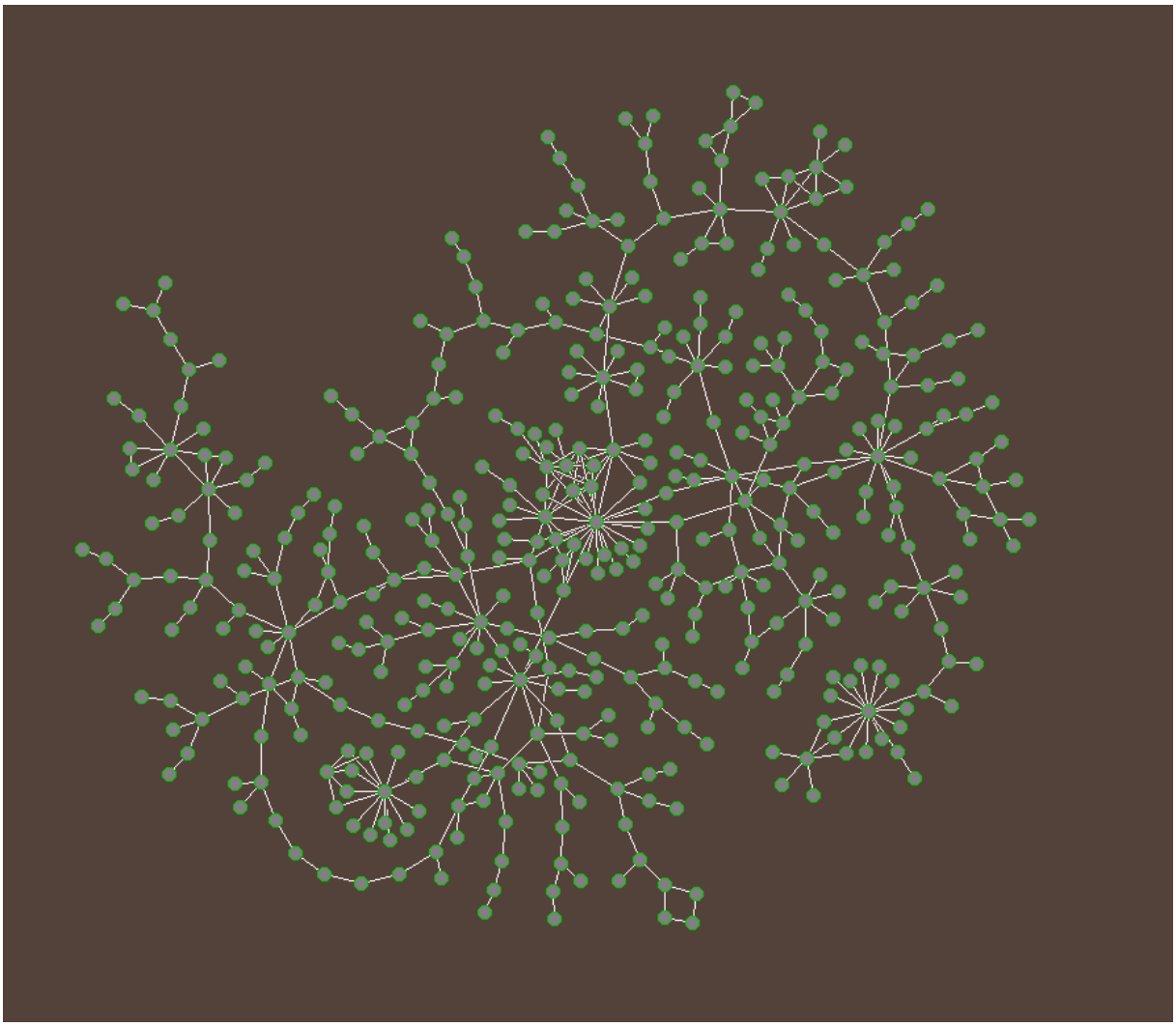


Figure 7.12 WA graph

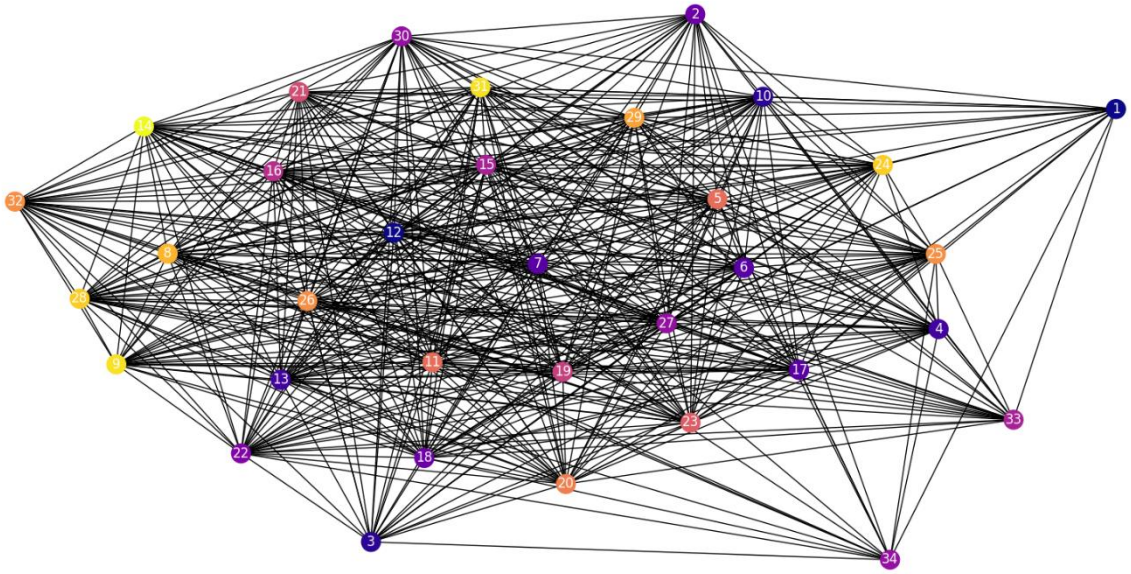


Figure 7.13 Complement Coloring

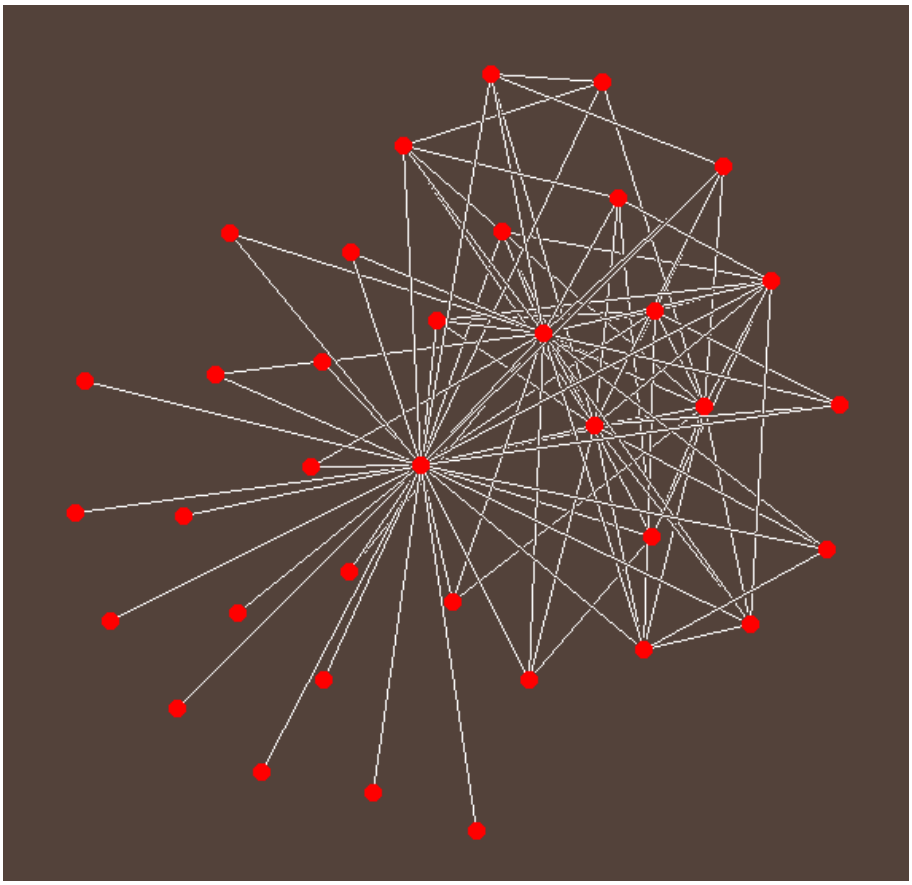


Figure 7.14 transactional and community pattern in the same cluster in OTP graph MCL

7.2.2 Embedded

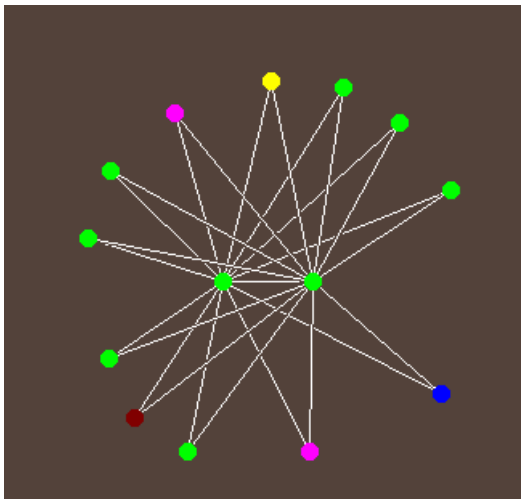


Figure 7.15 pattern in WA graph

Vertex set details: 3059

Vertex	Community	Name	Cluster
1508	[1332, 1909, 3059]	CRAMP	4135
1871	[2416, 3059]	DISCOMFORT	431
2942	[1788, 2172, 2223, 2624, 3059]	HARM	4053
2943	[2937, 3059]	HARMFUL	431
2971	[419, 912, 1772, 2182, 2434, 3059]	HEADACHE	5784
3160	[583, 817, 1059, 1059, 1332, 1382, 1772, 1772, 1788, 1788, 1791, 2172, 2223, 2416, 2423,...	HURT	431
3567	[2403, 2870, 3059]	LASH	431
372	[2092, 2958, 2974, 3059]	ATTACK	431
4473	[260, 260, 779, 817, 1080, 1173, 1173, 1332, 1413, 1413, 1772, 1788, 1894, 1909, 2141, 2...	PAIN	431
6103	[731, 2455, 3059]	STING	431
6463	[1413, 2098, 2098, 2105, 2455, 3039, 3059, 3166]	THORN	2527
6571	[1149, 1149, 3059]	TORTURE	431
6613	[2423, 2851, 3059]	TRAUMA	431
865	[164, 2486, 3059]	BURNT	2527

This pattern shows up, when there is a deeper connection among the meanings of the words. Usually, one is the subset of the other, creating the embedded structure mentioned above. In the transaction networks, this structure often suggests competitive areas between the nodes with high degree. The presence of an edge between these nodes is not necessarily forbidden, see below.

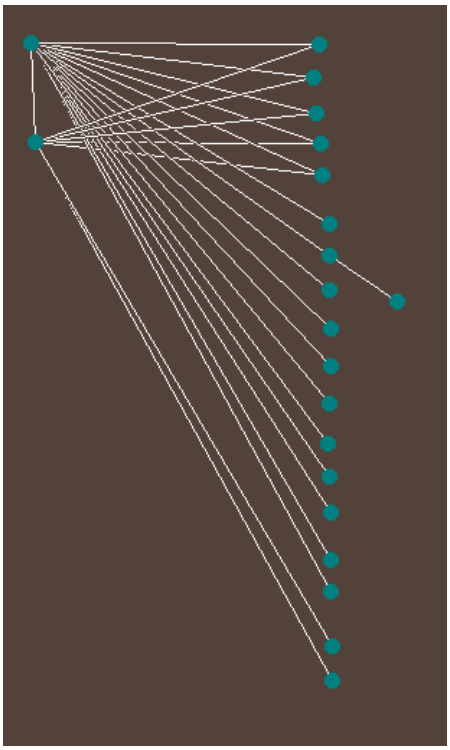


Figure 7.16 OTP

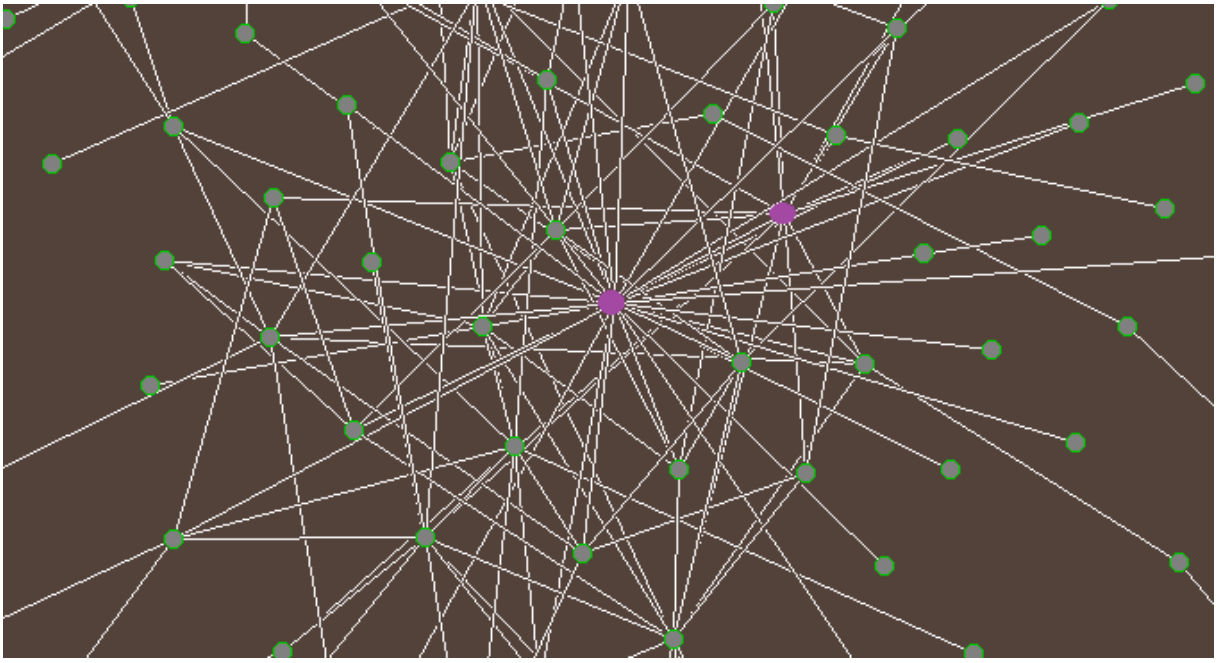


Figure 7.17: 2 supplier with a common customer set

Nodes: 15473275, 15473718

cluster: 15473275

community: 3468

7.3 Efficiency

- if the number of the vertices is larger than 600, the clustering algorithm makes too large clusters.

8 Further studies

- Implement a user interface to provide easy access to the algorithms for non-IT users.
- the application should provide opportunity to manually color graph nodes, modify the location of the nodes.
- The functions should be covered with unit tests
- Import the datasets into a graph db.
- Measuring tools for non-perfect coloring heuristics

This solution is scalable and modularized which makes further implementation more easier. The code and be found in the github repository liked below:

https://github.com/daniellanikov/Community-detector

9 References

[1] https://snap.stanford.edu/

[2] Usha Nandini Raghavan, Réka Albert and Soundar Kumara, 2007, Near linear time algorithm to detect community structures in large-scale networks

[3] NetworkX documentation: https://networkx.org/

[4] https://github.com/guyallard/markov_clustering

[5] Zachary, W. W., 1977, An Information Flow Model for Conflict and Fission in Small Groups

[6] Bron, C. and Kerbosch, J, 1973, Algorithm 457: finding all cliques of an undirected graph

[7] Etsuji Tomita, Akira Tanaka, Haruhisa Takahashi, 2006, The worst-case time complexity for generating all maximal cliques and computational experiments

[8] F. Cazals, C. Karande, 2008, A note on the problem of reporting maximal cliques

[9] M. E. J. Newman and M. Girvan, Finding and evaluating community structure in networks. Phys. Rev. E 69, 026113 (2004).

[10] M. E. J. Newman, 2006, Modularity and community structure in networks

[11] M. E. J. Newman "Networks: An Introduction", page 224. Oxford University Press, 2011.

[12] Clauset, Aaron, Mark EJ Newman, and Cristopher Moore. "Finding community structure in very large networks." Phys. Rev. E 70.6 (2004). <https://arxiv.org/abs/cond-mat/0408187>

[13] Reichardt and Bornholdt "Statistical Mechanics of Community Detection" Phys. Rev. E 74, 016110, 2006. https://doi.org/10.1103/PhysRevE.74.016110

[14] M. E. J. Newman, "Equivalence between modularity optimization and maximum likelihood methods for community detection" Phys. Rev. E 94, 052315, 2016. https://doi.org/10.1103/PhysRevE.94.052315

[15] A London, Ryan R. Martin, A. Pluhár, 2021, Graph clustering via generalized colorings

[16] https://www.python.org

[17] https://git-scm.com

[18] https://github.com/pyenv/pyenv-virtualenv

[19] https://wikipedia.org

- Cliques
- Small-world graphs

[20] https://matplotlib.org/stable/tutorials/colors/colormaps.html

10 Acknowledgements

Many thanks to all the scientists and colleagues, who contributed to this very old but rather fascinating topic of graph clustering in network theory and special thanks to my advisor, András Pluhár, who provided the help with patience and understanding through my learning path.

Tartalmi összefoglaló Magyar nyelven

A diplomamunka keretein belül egy prototípusnak megfelelő konzol alkalmazás került lefejlesztésre Python nyelven. Az applikáció tartalmazza a már felfedezett közösség detektáló és klaszterező algoritmusokat a NetworkX framework által behúzva. Ilyenek például a Newman-Girvan algoritmus, vagy pedig a Markov-lánc. Ezen felül számos saját fejlesztésű helper függvény segíti a felhasználókat az eredményes kutatás érdekében.

A diplomamunka témájának elődjéhez tartozik a kutatók és szoftverfejlesztők által közösen elkészített Sixtep szoftver. Az alkalmazás beépített funkciói közé tartozik a tanszék által fejlesztett közösség detektáló, a Markov-lánc és a maximum Modularitás algoritmus. Nem utolsó sorban az app rendelkezik grafikus kezelőfelülettel, ami nagyban megkönnyíti a dolgát a felhasználóknak. A felület meglehetősen felhasználóbarát, intuíciók alapján könnyen kitalálható a program működéséhez szükséges gombok használata. Az alkalmazás lehetővé teszi a gráfok betöltését fájlból, és még azt is, hogy a pontok helyzete szabadon testreszabható legyen a felhasználó által: a pontok mozdítása az egér bal gombát nyomva tartva történik. Ez az aprónak nem nevezhető funkció nagy jelentőséggel bír a gráfok szabad szemmel való vizsgálatánál.

Van lehetőség továbbá a klaszterek és közösségek kiexportálására. Ez azonban nem olyan zökkenőmentes, mint elsőre tűnik, ugyanis az adat nem a megfelelő formában van rendezve, így ez későbbi átalakításokat igényelt. Miután a konzol app megfelelő szerkezetbe rendezte a pontokat, már létre lehetett hozni a Python által használt Gráf objektumot.

Mivel a Sixtep szoftverben implementált közösségdetektáló algoritmus eltér a konzol app által behasznált Newman-Girvan algoritmustól, ezért nagy jelentősége van az exportálási lehetőségnek, további irányokat biztosítva a közösségek közti élek tanulmányozására. Nem csak a közösségek, de a klaszterek modularitása is eltér alkalmazásonként, habár az eltérés mértéke nem olyan drasztikus. Az eltérést az okozza, hogy a klaszterező algoritmusok a pythonos konzol appban jórészt a legnagyobb összefüggő komponensre vannak futtatva a sikeres futás érdekében. Ez a jelenség okozza a modularitás kis mértékű csökkenését a Sixtep szoftverben működő algoritmusokhoz képest. Még egy további oka is van a jelenségnek. Ugyanazt a gráfot vizsgálva a két programban az élek száma eltér, aminek a másik oka az, hogy az irányítottságot különböző mértékben kezelik. X -> Y és Y -> X él a Sixtep programban 2 élnek számít, míg a konzolos applikáció egy élt rendel hozzá.

A Python nyelv választásának okai

A script nyelvek általánosságban jó lehetőséget biztosítanak a prototípusok, Proof of Concept (POC) projektek kivitelezésére. Lehetővé teszik a programok gyors fejlesztését, cserébe a teljesítményből kell kis mértékben feladnia a programozónak. Ellentétben például a c++ nyelv adta lehetőségekkel, itt nincs mód a memória precíziós kezelésére, viszont így a hibák lehetősége is csökken, nagyobb kapacitást biztosítva így az algoritmusok tervezésére, kivitelezésére. Amennyiben egy heurisztika eredményesnek bizonyul, és a projekt váza kezd összeállni, úgy érdemes lehet áttérni a hatékonyabb programozási nyelvekre.

Ezen kívül a Python széles közben elterjedt, nagy támogatottságú könyvtárai is nagyban könnyítik a fejlesztő életét. Amennyiben mégis elakadna a fejlesztés valamilyen hiba folytán, rengeteg segítség érhető el online, tutorial cikkek és példa kódok formájában. A diplomamunkában felhasznált könyvtárak dokumentációjára sem lehet panasz. Rendkívül érthető és hatékonyan kereshető formában szolgáltatják az információt, az igényesen megfogalmazott API dokumentáció tehát megtérül, rengeteg mérnök órát spórolva.

A virtualizációs környezet lehetővé teszi, hogy különböző Python verziók ne ütközzenek egymással, illetve biztosítja, hogy minden függőség feltelepül a program sikeres futásához.

A PyCharm fejlesztői környezet rengeteg hasznos funkcióval segítséget nyújt, könnyebbé és gyorsabbá téve a fejlesztést, gördülékenyebbé a hibakeresést. Általában ezen környezetek ingyenesen hozzáférhetőek közösségi használatra, így a diplomamunka készítése során is lehetőség van a használatukra.

Az így készült konzol alkalmazás objektum orientált, modularizált ezáltal könnyen bővíthető. Karbantartása és továbbfejlesztése nem igényel sok munkaórát.

A diplomamunka során felhasznált könyvtárak egyike a NetworkX Python modul, amit gráfok tanulmányozására, közösség detektálók és klaszterezők alkalmazására fejlesztettek ki. Széles körben használt és rendkívül nagy támogatottsággal bír. A gráfok megjelenítésére a MatPlotLib függvénykönyvtár lett felhasználva, továbbá egyéb kisegítő célokra a SciPy függvényeket alkalmaztam. Ezen csomagok verziószáma a requirements.txt fájlban található, tételenként felsorolva. A megfelelő verzió letöltéséről és telepítéséről a fejlesztői környezet gondoskodik, amint megadtuk számára az előre definiált Python környezetet. Ennek szerkesztését a PyEnv modullal lehet elvégezni.

Rendelkezésre álló gráfok

Több forrásból is elérhetővé váltak nagyobb méretű, valós adatokból felépülő gráfok, nemcsak szociális struktúrával, de tranzakciós jelleggel is. Az IWIW közösségi oldalról legyűjtött adatokat, a szó-asszociációs gráfot, valamint az OTP negyedéves kimutatásainak anonimizált gráfját az Informatikai Intézet Számítógépes Optimalizálás Tanszéke biztosította a kutatáshoz. Ezen kívül még érdemes megemlíteni, hogy a Stanford Egyetem honlapján szép számban elérhetőek szociális jellegű gráfok Facebookról, Twitterről és egyéb neves helyről legyűjtve, természetesen szintén anonimizált formában. Ezeknek az adatoknak a linkje elérhető a felsorolt referenciák között. A Zachary-féle közismert szociális gráf is fel lett használva a dolgozatban példagráfként illusztrálva az algoritmusok helyes működését. A karate klubot ábrázoló kisebb gráf illusztrációként szolgált, az algoritmusok teljesítményét igyekeztem a többi nagyobb gráfon mérni.

Az applikációban elérhető funkciók

Newman-Girvan algoritmus

A közösségi szerkezet felderítésére és elemzésére szolgáló Girvan-Newman algoritmus azon élek iteratív kiküszöbölésére támaszkodik, amelyeknél a legtöbb a legrövidebb út a rajtuk áthaladó csomópontok között. A gráf éleinek egyenkénti eltávolításával a hálózat kisebb darabokra, úgynevezett közösségekre bomlik. Az algoritmust Michelle Girvan és Mark Newman vezette be. Az ötlet az volt, hogy megkeressük, hogy a hálózat mely élei fordulnak elő leggyakrabban más csomópontpárok között, azáltal, hogy megtaláljuk a központiság közötti éleket. A közösségeket összekötő élek ekkor várhatóan magas peremekkel rendelkeznek. A hálózat mögöttes közösségi struktúra sokkal finomabb lesz, ha megszűnnek a legnagyobb közötti élek, ami azt jelenti, hogy a közösségek sokkal könnyebben észrevehetők. [3]

Maximum Modularitás

Ez az algoritmus közösségeket talál gráfban a Clauset-Newman-Moore mohó modularitásmaximalizálás segítségével. Ez a módszer jelenleg nem veszi figyelembe az élsúlyokat. A mohó modularitás-maximalizálás minden egyes csomóponttal kezdődik a saját közösségében, és csatlakozik ahhoz a közösségpárhoz, amely leginkább növeli a modularitást, amíg ilyen pár nem létezik. [3]

Klikk kereső

Minden n csomópont esetében az n maximális klikkje egy legnagyobb teljes részgráf, amely n-t tartalmaz. A legnagyobb maximális klikket néha maximális klikknek is nevezik.

Ez a függvény egy iterátort ad vissza a klikkek felett, amelyek mindegyike csomópontok listája. Ez egy iteratív megvalósítás, ezért nem szenvedhet rekurziós mélységproblémákat.

Ez a függvény elfogadja a csomópontok listáját, és csak az összes ilyen csomópontot tartalmazó maximális kattintás jelenik meg. Jelentősen felgyorsíthatja a futási időt, ha néhány konkrét klikkre van szükség.

A find_cliques(G) függvény listakimenetét használták az összes maximális kattintás meghatározásához. A legrosszabb forgatókönyv esetén azonban a lista hossza exponenciális lehet a grafikon csomópontjainak számában. Ez a funkció elkerüli az összes kattintás tárolását a memóriában, mivel a keresés során csak az aktuális jelölt csomópont-listákat tartja a memóriában.

Ez a megvalósítás a Bron és Kerbosch (1973) [6] által közzétett algoritmuson alapul, amelyet Tomita, Tanaka és Takahashi (2006) [7] adaptált, és Cazals és Karande (2008) [8] tárgyalt.

Ez az algoritmus figyelmen kívül hagyja az önhurkokat és a párhuzamos éleket, mivel a klikkeket hagyományosan nem ilyen élekkel határozzák meg. [3]

Markov-lánc

Az MCL algoritmus a Markov Cluster Algorithm rövidítése, amely egy gyors és méretezhető, nem felügyelt klaszter-algoritmus gráfokhoz (más néven hálózatokhoz), amely a gráfok (sztochasztikus) áramlásának szimulációján alapul. Az algoritmust Stijn van Dongen találta fel/fedezte fel a Hollandiában található Matematikai és Számítástechnikai Központban (más néven CWI). [4]

H-elkerülő színezés

Az általános (vagyis nem feltétlenül kétoldalú) tranzakciós gráfok újfajta klaszterezését mutatták be a megfelelő színezések bizonyos osztályán keresztül. A klaszterek színosztályok, mivel a klaszteren belül nem kívánatos élek. Az élek szerkezete az osztálypárok között

korlátozott. A fenti példák arra utalnak, hogy bizonyos esetekben teljesen beágyazott vagy ezzel egyenértékű beágyazottsági relációnak kell lennie bármely kétszínű osztály között. Ezt a fogalmat általánosítjuk egy tetszőleges G gazdagráfra és egy tiltott kétrészes H részgráfra az alábbiak szerint.

Kondenzáció

Minden közösséget vagy éppen klasztert egy pontra tömörít az algoritmus. Az élek jelenléte az új pontok között bizonyos heurisztikákhoz kötött. Minél nagyobb a közösségben lévő pontok száma, annál biztosabb, hogy húzódni fog él a két közösség között. Ezért az új él behúzásánál figyelembe kell venni az eredeti osztály méretét, az élsűrűséget, az osztály szerkezetét. Az algoritmus akkor működik jól, ha az input gráf összefüggő.

Kis-világ tulajdonság

Ez a függvény a gráfokban való kis-világ tulajdonság jelenlétének becslésére szolgál. Egy kis-világ hálózatot kis átlagos legrövidebb úthossz és nagy klaszterezési együttható jellemez. A kis-világ tulajdonságot általában a szigma vagy omega együtthatóval mérik. Mindkét együttható összehasonlítja egy adott gráf átlagos klaszterezési együtthatóját és legrövidebb úthosszát ugyanazokkal a mennyiségekkel egy ekvivalens véletlenszerű vagy rácsos gráf esetében. Egy gráfot általában kisvilágúnak minősítenek, ha a szigma>1 vagy omega nullához közeli értéket vesz fel. Az algoritmus időigényes, nagy gráfok esetén használhatatlan.

Színezős stratégiák

Egyedi implementációt igényelt a detektáló algoritmusok által visszaadott osztályok pontjainak színskálákhoz való rendelése. A megvalósított logika meglehetősen naiv, cserébe működik. Erre a célra lett létrehozva a Cluster nevű helper class, ami egy uuid-t rendel a csoportokhoz. A függvény mohó logika alapján választ egy színt a skáláról és a csoportokhoz rendeli, lehetőleg úgy, hogy a színek távol essenek egymástól, így a csoportok a megjelenítés után szabad szemmel is könnyebben megkülönböztethetőek lesznek. Természetesen minél több az osztály, annál nehezebb ezt az igényt megvalósítani. A skála választása sem triviális feladat annak ellenére, hogy a MatPlotLib függvénykönyvtár remek lehetőségeket biztosít. Az

átmeneti skálák nem ideálisak a precíz megkülönböztetésre, a jól elhatárolt színek meg beleolvadnak a háttérbe, ezért a kettő között érdemes színskálát választani. Például az egyik sokat használt színskála a dolgozatban a „Plazma” nevet viseli, az ábrája megtekinthető a dolgozat Colormap című fejezetében.

Mérések, eredmények, minták

Az algoritmusok jóságának meghatározása nem egyszerű feladat, elvégre honnan tudjuk, hogy mi számít jó detektálásnak. A gráfok nagy része anonimizált, és még referencia adatok sem állnak rendelkezésre, ezért a probléma összetettsége megkívánja, hogy több oldalról legyen mérve a hatékonyság.

Az egyik módszer az algoritmusok hatékonyságának számszerűsítése a modularitás számítása. Az algoritmus nem csak hatékonyságot tud mérni, de egyben maga is megad egy lehetséges osztályozást.

„Egy hálózat közösségi struktúrája az élek statisztikailag meglepő elrendeződésének felel meg, számszerűsíthető a modularitásként ismert mértékkel [9]. A modularitás egy multiplikatív állandóig annyi, hogy a csoportokba eső élek száma mínusz a várt szám egy ekvivalens hálózatban, ahol az élek véletlenszerűen vannak elhelyezve. A modularitás lehet pozitív vagy negatív, a pozitív értékek a közösségi struktúra lehetséges jelenlétét jelzik.” [3]

A kísérleti eredmények a modularitás vizsgálatára több érdekességet is mutattak. Például minél nagyobb a gráf, a modularitás az algoritmusok jóságától függetlenül kúszik lefele. Persze az is igaz, hogy minél nagyobb egy gráf, annál nehezebb értelmes osztályokat adni. Megfigyelhető továbbá, hogy ha nem összefüggő gráfnak vesszük a legnagyobb összefüggő komponensét, akkor is esik a modularitás. Ez is könnyen belátható, értelemszerűen a leszakadt darabok jól definiált osztályokat alkotnak, így növelik a modularitást. Az eredmények összehasonlításánál érdemes tehát figyelembe venni, hogy a Sixtep szoftver a gráf egészén futtat algoritmusokat, míg a dolgozat keretein belül fejlesztett konzol alkalmazás a gráf legnagyobb összefüggő komponensén, ezáltal az értékek megtévesztőek lehetnek.

A létrehozott színosztályok száma

Egy egyszerű, ám meglehetősen hasznos paraméter, hogy az algoritmus hány színt használt el az osztályozás során. Különösen igaz ez a tranzakciós gráfokon végzett klaszterezések esetében, ahol a modularitás nem annyira számottevő. (Bár sokkal jelentősebb, mint azt az ember elsőre gondolná)

Az algoritmus futási ideje

Cseppet sem elhanyagolható tényező, hogy mennyit kell várnunk egy-egy klaszterezési eredményre.

Egy szoftverfejlesztő szemszögéből meglehetősen bináris az ügylet: Ha lefut az algoritmus, ameddig utántöltődik a kávé, akkor teljesen rendben van. Egyébként pedig vissza kell térni és csiszolni kell még a hatékonyságot. (Egy applikációban nem mutat jól, ha várnunk kell. A felhasználó könnyen elbizonytalanodik, hogy az alkalmazás hibás, de nem jelez, vagy pedig az algoritmus lassú, akkor is, ha egyébként az információ jelezve van.)

Ha elérkezik az a pont, hogy feltehetjük, hogy az algoritmus jól működik és az adatokban sincs semmi hiba, akkor érvényt szerez magának az ügy tudományos megközelítése, az algoritmus futási idejének tudományos számítása. Például a Stanford Egyetem által legyűjtött óriás gráfokon hasznos, ha meg tudjuk becsülni, hogy a pontok és az élek ismeretében mennyi lesz a futási idő.

E kritikus paraméter csökkentésére rengeteg megoldási irány keletkezett, többek között az adatok feldolgozásának a többszálúsításának a gondolata. Numerikus műveleteknél célravezető, ha nem egymásra épülő számításokat nem lineárisan, hanem párhuzamosan végeztetjük el. Erre a legalkalmasabb hardver a grafikus kártya (GPU).

Minták a gráfokban

Ez talán az egyik legszubjektívebb és legabsztraktabb része a dolgozat keretein belül elért eredményeknek. Az, hogy mit tekintünk gyakori mintának és hogy különböző gráfokban az adott struktúrát hogyan értelmezzük, meglehetősen kreatív gondolkozásmódot igényel és koránt sem biztos, hogy mindig ugyanazt jelenti.

Például egy tranzakciós gráfban a termelő – fogyasztó szerkezet egy központi egységhez sok kapcsolódó pontot mutat. Szociális viszonylatban pedig lehet egy összekötő ember, vagy főnök. A szó-asszociációs gráfban ez a minta egy általános szót takar, mint például az „eszköz”. A szó beleillik rengeteg témakörbe, így számos él fut be az őt reprezentáló pontba. Ha a mintából kiindulva további mélységig vizsgáljuk a gráfot, akkor egy fa-jellegű struktúra rajzolódik ki, jellemzően tranzakciós adatokon, vagy pedig a szó-asszociációs gráfon.

Jelen van továbbá egyfajta beágyazottság is bizonyos osztályok között. A szó-asszociációs környezetben főként akkor jelenik meg, amikor a szavak jelentése meglehetősen eltér, de a téma viszont szorosan kapcsolódik egymáshoz. Gyakori jelenség, hogy egyik téma egy részhalmaza a másiknak, létrehozva ezzel a beágyazott szerkezetet. A tranzakciós gráfoknál viszont a piaci viszonyok egyik jele lehet és versenyzést mutathat a beszállításra alkalmas területek között. Ennek ellenére nem ritka az sem, hogy a versenyző felek között is kapcsolat van.

A fent említett mintákról készült ábrákat a dolgozat Patterns fejezetében lehet megtekinteni. Az egyetlen nagy méretű gráf, ahol valamilyen formában elérhetőek voltak nem anonimizált adatok, az a szó-asszociációs gráf, így a minták vizsgálata, összehasonlítása különböző gráfokban meglehetősen fapados eredményeket produkált.

A téma tanulmányozásának további lehetséges irányai

Nem minden ember barátkozik meg a konzolos lehetőségekkel, illetve ma már széles körben vannak támogatva a felhasználókat segítő grafikus interfészek, így a jelen alkalmazáshoz is célszerű lenne egy grafikus felület, ami megkönnyíti a program használatát. Ha az alkalmazás rendelkezik grafikus megjelenítéssel az további lehetőségeket is hordoz magában. Például a gráf pontjainak egérrel való mozgatását, illetve manuális színezését lehetővé tevő funkciók hasznos részei lehetnek a programnak.

A szoftver minőségének egyik ismérve, hogy mennyi a unit tesztekkel való lefedettsége. Ez nem csak a regresszió ellen nagyon hasznos, de valamilyen mértékig garantálja a program megfelelő működését. Biztosítékot ad továbbá a robusztusságra is, amennyiben nem csak a happy path-t teszteltük le.

Nagyméretű gráfoknál kifejezetten sokat számíthat a teljesítményen a gráf adatbázisok használata. Egy kommunális célokra ingyenesen hozzáférhető, felhő alapú adatbázis mindig

rendelkezésre áll és nem komplikált az adatok importja sem. További érv a használata mellett, hogy az alkalmazás mostani állapotában alkalmas a gráf adatbázishoz való csatlakozásra.

Abban az esetben, ha nem egyértelműen eldönthető, hogy egy pont hova is tartozik igazán, érdemes megengedni bizonyos hibákat az algoritmus futása során és egy határt, hogy mennyire engedjük meg.

Az alkalmazás fejlesztése közben igyekeztem figyelmet szentelni az objektum-orientáltságra és a megfelelő mértékű modularizációra, hogy egy esetleges refaktor, vagy éppen továbbfejlesztés során a ráfordított idő gazdaságos legyen.

A forráskód a következő github linken érhető el:

https://github.com/daniellanikov/Community-detector

Nyilatkozat

Alulírott Nikov Daniella programtervező informatikus MSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Számítógépes Optimalizálás Tanszékén készítettem, programtervező informatikus MSc diploma megszerzése érdekében. Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel. Tudomásul veszem, hogy diplomamunkámat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.