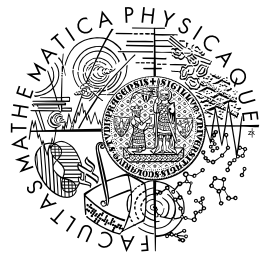


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Matúš König

Detection of Influential Individuals, Communities, and Link Prediction in Social Networks

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis:	doc. RNDr. Iveta Mrázová, CSc.
Study programme:	Computer Science
Study branch:	Artificial Intelligence

Prague 2023

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I want to express my most profound appreciation to the supervisor of this thesis doc. RNDr. Iveta Mrázová, CSc., whose invaluable advice and support assisted me in achieving success.

Title: Detection of Influential Individuals, Communities, and Link Prediction in Social Networks

Author: Matúš König

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: doc. RNDr. Iveta Mrázová, CSc., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Social network analysis provides several means to better understand the structure of the underlying social networks. This thesis is focused on the area of community detection in social networks. We discuss six of the main community detection algorithms and their hybrid variants involving a combination of rough and fine partitioning techniques. The text explains the measures used to quantify the detected communities' properties. For different problem sizes, the Zachary's karate club and Enron email datasets were used. Further, the work concentrates on experiments that provide performance assessment for the investigated methods. Based on the obtained results, we draw conclusions towards recommendations for a reliable usage of the findings in practice. At the same time, we aim to identify the appropriate number of communities in the data at hand since this is a parameter of many community detection algorithms. For the same reason, we also investigate whether non-hierarchical clustering algorithms could be used to form a sub-community hierarchy. All of the mentioned experiments were run by means of a community detection system CGAT - Config-based Graph Analysis Tool we developed and implemented as a part of the thesis.

Keywords: data mining, social networks, detection of influential individuals, community detection, link prediction, knowledge representation

Contents

Introduction

With the enormous rise of recently used internet and computational resources, it became possible to collect and process structured graph data on a large scale. Social network analysis and community detection have become strategic topics, mostly due to the creation of online social networks such as Facebook. This highlights the advantages of understanding the structure of social networks and is widely used in psychology, economy and politics.

Nowadays, multiple approaches and paradigms exist to detect communities in social networks. Naturally, with a vast offer of algorithms, choosing the most appropriate method is complicated due to all the differences. In practice, this results in the need to perform grid search on as many algorithms and parameters as possible, since only this technique can ensure the optimality of the result. The main disadvantage of this approach is time and memory requirements, creating pressure on all parts of the processing chain.

Our work aims to solve this issue and reduce the variety of feasible algorithms to only a few reasonably efficient ones, which will provide universally good results. The beforementioned condition should be fulfilled with all data types, including graphs with thousands of nodes and edges and different graph clustering conditions and objectives. This also includes an estimation of algorithm parameters that would yield maximum accuracy and performance.

In more detail, we specify social networks and their key properties in Chapter 1. Here, we illustrate major types and applications involving social networks and graph data. This includes the properties of homophily and triadic closure, which enable us to define and detect communities. In this chapter, we also introduce the terms centrality and link prediction, which are trending topics in social network analysis.

Chapter 2 is dedicated to formally defining the community detection problem. We describe what is perceived as a community and state multiple community detection measures. Later, we establish the main algorithms applicable to community detection. In this chapter, we also formulate hybrid variants of community detection algorithms, where we combine rough and fine graph partitioning. It is important to note the algorithms are, in all cases, optimizing objective functions

and their performance metrics are later assessed in the thesis.

This thesis uses two datasets to compare the algorithms defined in Chapter 3. Those datasets cover a range of small and medium-sized problems and provide a baseline for universal community detection. The Zachary Club dataset covering small data is labeled with regard to the actual node community. The assigned labels allow us to compare the achieved accuracies of the algorithm. Within this study, bigger networks are omitted to allow us to evaluate fairly all algorithms, even with high time complexity, which would not be possible with larger datasets.

The experiments outline can be found in Chapter 4, and the results of the respective experiments can be found in Chapter 5. First, we estimate the number of communities that can be found in the Enron dataset since the number of communities is a parameter for many community detection algorithms. Next, we study the behavior of nonhierarchical clustering, where new communities are not necessarily created on the basis of previous clustering. Finally, we evaluate the performance of the algorithms on two datasets of different sizes to form a conclusive recommendation.

All experiments are run using CGAT - Config-based Graph Analysis Tool. While the system was developed primarily for the thesis, it works independently, creates persistent replicable results and is open to extensions in the field of community detection. Chapter 6 illustrates the design of the framework in more detail.

Chapter 1

Social networks

Nowadays, most readers have associations such as Facebook or Twitter associated with social networks. This term, however, has a much broader application, unlocking much more resources to study. In this section, we will introduce social networks, their most important properties and many important measures in various fields according to Aggarwal et al. [1].

1.1 Definition

It is generally assumed that social network $G = (V, E)$ is a graph, where $V(G)$ is the set of vertices and $E(G)$ is a set of edges. The graph can be either undirected or directed and is directly derived from observed data. It will be assumed the social network is undirected unless stated otherwise. Usually, the naming convention calls the node an actor, and the edge is a link between actors. All parts of the social networks can contain additional data, including classification information. The definition assumes graph G is static for some timestep t. This definition can be extended to a dynamic graph by considering a series of snapshots $G_0, ..., G_t$.

The most common examples of real-life social networks include the following:

- Traditional social network as we know from sociology. Observation of humans and their social interactions over time. The main disadvantage is measurements of fields of interest, such as information flow or community detection, can not be done with much precision.

- Online social network and communication platforms. User interaction in online platforms forms a social network, whether we are considering online social networks such as Facebook or other means of online communication such as emails and instant messaging.

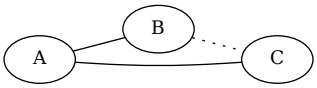


Figure 1.1 Triadic closure example. Since node A is connected to B and C, there is a high chance nodes B and C are also connected.

- World wide web. Document hyperlinks naturally form directed graphs describing connections between different web pages.
- General graph data. Many common things, such as scientific citations or money transactions, form social networks and their analysis can uncover hidden structures of the network.

1.2 Key properties of social network

As Aggarwal et al. [1] defines in his book, social network structure can be described using properties such as homophily and triadic closure. They describe how the network is formed and explain more information about found connections in the network.

1.2.1 Homophily

According to Aggarwal et al. [1],*homophily* is a fundamental property defining social network structure. Homophily is node similarity when connected nodes have a high chance of having similar properties or common background.

This can be explained by how the node connections are formed in the human social network. A lot of a person's connections are acquired through school, work experience or hobbies. This implies that connected nodes might have common properties and interests. As a direct consequence, links in social networks are formed using common ground rather than random acquaintances.

1.2.2 Triadic closure and clustering coefficient

The main undirect consequence of homophily is triadic closure. *Triadic closure* can be defined as a structural homophily, where it is assumed if two nodes have a friend in common, those nodes are either connected or a connection will form in the future. We can see an example of triadic closure in Figure 1.1.

Triadic closure can be explained easily using homophily. If two nodes have a common friend, it is assumed they might have common background or attributes and thus increasing the chance of the current or future connection.

Clustering coefficient is a triadic closure measure and according to Aggarwal [1], quantifying the tendency of the network to create clusters. Formally, let S_i mark a set of adjacent nodes to node $i \in N(G)$, where $G = (N, V)$ is an undirected graph. Let $n_i = |S_i|$. For the set S_i, there is a total of $\binom{n_i}{2}$ possible edges. If there is a connection $(j, k) \in V, j \in S_i, k \in S_i$, there is a mutual connection with node i. Node clustering coefficient $\eta(i)$ for node i is then a fraction of such connections to all possible connections.

$$\eta(i) = \frac{|\{(j,k) \in V \ : \ j \in S_i, k \in S_i, \}|}{\binom{n_i}{2}} \tag{1.1}$$

The concept of node clustering coefficient can be extended to the whole network by calculation of *average clustering coefficient* using all nodes of the graph.

1.3 Detection of influential individuals

Network structure is usually nonhomogeneous and therefore nodes which are considered central have a significant impact on properties of the social network. Naturally, many of hub nodes tend to have high centrality measure as a result of dynamic network creation process and have higher influence on the other nodes. Methods of determining centrality will be discussed with undirected networks, while with directed network we will consider *prestige*. Prestige can be summarized by how much is an actor being followed by other actors of the network. Opposite case is *gregariousness*, which is focused more on outgoing connections. This section defines all centrality measures as defined by Aggarwal [1].

1.3.1 Degree centrality

Simplest centrality metrics to define is *degree centrality* $C_D(i)$. Degree centrality is defined as

$$C_D(i) = \frac{Degree(i)}{n-1} \tag{1.2}$$

and represents a degree of a node divided by a maximal possible node degree and thus can be considered as a relative node degree. Analogously, we define degree prestige as

$$P_D(i) = \frac{Indegree(i)}{n-1} \tag{1.3}$$

and degree gregariousness as

$$G_D(i) = \frac{Inegree(i)}{n-1} \tag{1.4}$$

1.3.2 Closeness centrality

Main drawback of degree centrality is it is considering only local connections and discards any indirect ones. *Closeness centrality* is a more efficient metric for measuring indirect links.

Let $Dist(i, j)$ be a length of shortest path from node i to node j. Average node distance is then defined as pairwise node distance average as follows:

$$AvDist(i) = \frac{\sum_{i=1}^{n} Dist(i, j)}{n-1} \tag{1.5}$$

Closeness centrality $C_C(i)$ is then simply defined as inverse of average distance.

$$C_C(i) = 1/AvDist(i) \tag{1.6}$$

It is easy to see closeness centrality scales from 0 to 1 and that nodes close to the rest of the networm reach high closeness centrality.

1.3.3 Betweenness centrality

While closeness centrality provides great results using shortest path distances, many details about information flow in terms of number of shortest paths passing through the node is ommited. It is assumed nodes critical to the network have a great control on information flow, which is represended by high amount of passing by shortest paths.

Let q_{jk} denote number of shortest paths between nodes j and k and let $q_{jk}(i)$ mark number of shortest paths between nodes j and k passing through node i. We can observe the fraction $f_{jk}(i) = q_{jk}(i)/q_{jk}$ describes influence of the node i on information flow between the nodes j and k. We then define betweenness centrality $C_B(i)$ as an average through every possible pair of nodes $\binom{n}{2}$.

$$C_B(i) = \frac{\sum_{j<k} f_{jk}(i)}{\binom{n}{2}} \tag{1.7}$$

We can similarly extend betweenness centrality to edges instead of nodes by adjusting the formula to accept an edge rather than a node.

1.3.4 PageRank

PageRank was developed by Brin and Page [2] to model the importance of WWW pages using the natural graph structure of the internet. It is based on the idea that influential web pages are linked to other important documents. PageRank can be used in social network analysis to detect influential actors in the network.

PageRank is described as a random-walk model. To explain it further, define a random surfer who visits web pages by following random hyperlinks. We can observe the number of visits is directly connected with the number of incoming edges. In addition, long-term visit frequency will be higher if incoming links are from other frequently visited nodes. We note long-term frequency as a steady-state probability.

It is easy to see not all network topologies will work correctly with this simple random-walk model. The main issue is dead-ends, either as nodes or graph components, without any outgoing edges. In principle, any random walk will get stuck in such dead-ends, negatively adjusting steady-state probability. To solve matter in question, we adjust our model. Firstly, dead-end nodes can be solved easily by adding edges to every other node in the graph including itself. To solve isolated components, we introduce *restart step*. With probability of α we restart the random walk by teleporting random surfer to random node and with probability $1 - \alpha$ we continue by selecting random adjacent node. Parameter α is usually set to 0.1.

Let $G = (N, A)$ to be directed graph and let n mark total number of nodes. Assume A also contains added edges for dead-end nodes to all other edges. Denote $In(i)$ and $Out(i)$ as sets of nodes pointing to and from node i. Also denote steady-state probability as $\pi(i)$. The PageRank of the node i is then equal to steady-state probability $\pi(i)$. Mark $p_{ij} = 1/|Out(i)|$ the probability of transition from node i to j in case where restart step is not applied.

We will discuss possible transitions to node i below. Steady-state probability $\pi(i)$ is a sum of probability of teleporting to the node with probability α and a transition from adjacent nodes with probability $\alpha - 1$. We can see easily that the probability of random teleportation to node i is α/n. Probability of transition from adjacent nodes to node i can be then written as $(1 - \alpha) \sum_{j \in In(i)} \pi(j) p_{ji}$. We then have a system of equations, which holds for each node i:

$$\pi(i) = \alpha/n + (1 - \alpha) \sum_{j \in In(i)} \pi(j) p_{ji} \tag{1.8}$$

In practise, this is solved using power iteration by adjusting values of π iteratively.

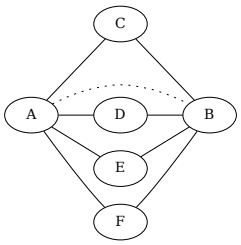


Figure 1.2 Link prediction example.

1.4 Link prediction

Predicting future links between pairs of nodes in a social network is often a desirable task. Online social networks such as Facebook or Twitter often use this method to recommend potential friends and followers to their users.

According to Aggarwal et al. [1], both structural and content similarity can be utilized to predict future links between nodes. *Structural measures* are based on the property of triadic closure. Main idea is if two nodes have common neighbors, it is high probability they will be connected in the future. *Content-based measures* is formed using homophily principle. In this case nodes with similar content have bigger chance of connection. For example, if two actors attended same high school and they are of similar age, there is a great chance they know each other and have interacted in the past. This section introduces all important measures related to link prediction as stated by Aggarwal [1].

1.4.1 Neighborhood-based measures

Simplest method for estimating probability of link is using neighborhood-based measures. In principle, number of common neighbors between pair of nodes i and j defines probability of node connection. We state different methods below.

Define S_i as a neighborhood set of node i, marking a set of adjacent nodes of node i. We can then define first metric:

Definition 1 (Common neighbor measure)**.** *Commom neighbor measure between nodes i and j is equal to number of common neighbors of nodes i and j and is defined as follows:*

$$CNM(i, j) = |S_i \cap S_j| \tag{1.9}$$

It is easy to see the Common Neighbors Measure takes into account only absolute values of neighbors, dismissing information about node degrees. This

is a disturbing property, since we can see high-degree nodes representing either public figures might cause connections being recommended only by chance. The *Jaccard measure* is designed to use relative value instead of absolute node count, taking into account different degree distributions.

Definition 2 (Jaccard Measure)**.** *Jaccard measure for link prediction is equal to Jaccard coefficient between respective neighborhood node sets S_i, S_j.*

$$Jaccard(i, j) = \frac{|S_i \cap S_j|}{|S_i \cup S_j|} \tag{1.10}$$

We can see in Figure 1.2 that Jaccard measure between nodes A and B is 1. If we increase either node A or B degree without adding common neighbor, it would result in lower coefficient.

Jaccard measure works great for degree adjustments of nodes between which the link prediction is measured. However, for degrees of intermediate neighbors it is not adjusted well. In Figure 1.2, common neighbors are only connected to nodes A and B. Nevertheless, those common neighbors could be popular actors represented with high degree nodes. As a result, those nodes would have much higher statistical occurance as they would occur as common neighbors of many neighboring nodes, marking them as less important in link prediction. The Adamic-Adar measure is introduced to consider for different importance of common neighbor. It can be seen it is a modification of commom neighbor measure with applied weights, where weight is defined as a function of node degree.

Definition 3 (Adamic-Adar Measure)**.** *Adamic-Adar common neighbor measure for nodes i and j is equal to weighted count of common neighbors. The weight of node k is then defined as* $1/\log(|S_k|)$.

$$AdamicAdar(i, j) = \sum_{k \in S_i \cap S_j} \frac{1}{\log(|S_k|)} \tag{1.11}$$

1.4.2 Katz measure

While neighborhood-based methods presents great estimation of probability of forming a connection between a pair of nodes, they lack capability to predict properly for pair of nodes with low amount of common neighbors. For example, we can see in Figure 1.3 that nodes A and B have no direct common neighbors. However, it is easy to see nodes have significant indirect connection after consideration of longer paths. For such cases, walk-based measures such as Katz measure are more suitable.

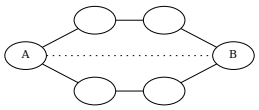


Figure 1.3 Link prediction example with indirect connectivity.

Definition 4 (Katz measure)**.** *Let n_{ij}^t be the number of paths of length t between nodes i and j. Then for parameter $\beta < 1$ the Katz measure is defined as follows:*

$$Katz(i,j) = \sum_{t=1}^{\infty} \beta^t n_{ij}^t \tag{1.12}$$

Parameter β is a discount factor penalizing longer paths. Convergence of Equation (1.12) is guaranteed for small values of β. Let A be an adjacency matrix of an undirected network. It is easy to see the matrix is symmetric. Pairwise $n \times n$ Katz coefficient matrics can be then computed as:

$$K = \sum_{i=1}^{\infty} (\beta A)^i = (I - \beta A)^{-1} - I \tag{1.13}$$

As per Aggarwal et al. [1], the value of β should always be smaller than the inverse of the largest eigenvalue of A to ensure convergence. As an extension, we can consider a weighted version of A considering of edge weights of the graph.

1.4.3 Random walk-based measures

Random walk-based measures are another way to quantify connectivity between pairs of nodes. Representative of this approach is PageRank defined in Section 1.3.4. To measure the similarity between nodes i and j, we perform personalized PageRank of node j, where the restart step is applied at node i. The consequence of the random-walk model being started at node i is that close nodes will be visited more often, representing a higher chance of indirect connection. That means if node j is close to node i, then the value of personalized PageRank will be high.

1.4.4 Feature-based link prediction

All measures mentioned above are unsupervised methods for link prediction. Usually, beforehand mentioned measures do not have equal accuracy in different networks. Aggarwal [1] we can improve the results by considering link prediction as a binary classification problem by treating the absence or presence of the link as

a binary classification result. This can be achieved by extracting multidimensional data record from each pair of nodes. In the feature construction, we can consider all link prediction measures as well as descriptors such as node degrees. In addition, we can add information special to the network, such as node and edge content.

We construct multidimensional feature values for each edge in the input graph to create data for positive training examples. To create negative examples, we sample pairs of nodes without a connection. Only a sample of negative connections is sufficient since real-world networks are sparse and thus, the ratio of present to absent links is unbalanced. Such data suits any binary classifiers such as binary trees, SVN, or neural networks. During model inference, we construct data records consistently with the training phase.

Chapter 2

Community detection

In this chapter, we focus on community detection as an important area of social network analysis. At first, we define what a community is, and we introduce multiple functions measuring the quality of the partition. Next, we state essential algorithms and methods for community detection. Finally, we summarize the theoretical results and conclude a set of candidates providing the best results.

2.1 Definition

Community detection is an integral part of social network science. It allows us to inspect the network structure in the bigger context, in the context of nodes sharing similar properties. It is an important field of study with further implications in psychology, public health, economics, and politics.

Intuitively, a community in a social network is defined as a densely connected part of the graph sparsely connected to other communities. In other words, a member of the community is much more likely to interact with other members of the same community than the rest of the network. Distinct algorithms usually work using different objective measures. However, all of those metrics are consistent with the intuition.

As a side note, there are exponentially many possible graph partitions. This is the equivalent of a partition of a set and that is exponentially large in the size of the set. The number of all set partitions is called *Bell number*.

2.2 Metrics

In this section, we will define quality functions for community detection. Quality functions are important, as they measure the goodness of clustering and determine the performance of different algorithms. Some of the measures are directly used

as algorithm objection functions and optimized. The literature suggests multiple metrics, all of them directly based on the interpretation of the definition of community in social networks.

2.2.1 Edge cut

Edge cut is the easiest metric to define, and it is based on a simple principle. Define intra-cluster edge weight as the sum of edge weights whose nodes lie within the same community and define inter-cluster edge weight as the sum of edge weights of edges connecting different communities.

We define edge cut as the difference between intra-cluster and inter-cluster edge weights. We want to maximalize this difference, as it reflects the need for strong connections between community members and weak links otherwise. Alternatively, we can define edge cut as only inter-cluster edge weight, measuring the sparsity of connections between different clusters.

2.2.2 Normalized cut and conductance

According to Parthasarathy, Ruan, and Satuluri [3], important metrics are normalized cut and conductance. Consider graph $G(N, E)$ with weighted edges w_{ij}. Mark $degree(i)$ as a weighted degree for node i of the graph G. For directed graphs, out degree is used.

Normalized cut for a group $S \subset N(G)$ is then defined as

$$Ncut(S) = \frac{\sum_{i \in S, j \in \bar{S}} w_{ij}}{\sum_{i \in S} degree(i)} + \frac{\sum_{i \in S, j \in \bar{S}} w_{ij}}{\sum_{j \in \bar{S}} degree(j)} \tag{2.1}$$

The normalized cut represents a sum of edges connecting S with the rest of the graph normalized by the sum of edge weights in S, resp. $G(V) \setminus S$. The lower the normalized cut, the sparser the connection of S and the rest of the graph, and the denser the links inside S itself. The low normalized cut then marks a good division of the graph.

Similarly, *conductance* is defined as

$$Conductance(S) = \frac{\sum_{i \in S, j \in \bar{S}} w_{ij}}{\min(\sum_{i \in S} degree(i), \sum_{j \in \bar{S}} degree(j))} \tag{2.2}$$

In the case of k clusters, we calculate the normalized cut or conductance of the graph as the sum of normalized cuts / conductances for each individual community.

2.2.3 Modularity

The breakthrough approach to community detection is comparing the graph to randomly wired networks. According to Barabási and Pósfai [4], random wired networks lack any structure resembling communities, all independent of network degree distribution.

The null hypothesis states random wired networks lack any structure related to communities. Consider a network with N nodes and L links. Consider a community C with N_C nodes and L_C edges. Using degree preserving model, we will decide whether C is the result of random wiring or whether C is close to what we define as a community.

Define *modularity for a community* C as

$$M_C = \frac{L_C}{L} - \left(\frac{k_C}{2L}\right) \tag{2.3}$$

where k_C is the total weighted degree of the nodes in the community C.

The value of the modularity M_C corresponds to how well the subgraph resembles random wiring. If M_C is 0, then the graph is random. Higher M_C correlates with better community structure, marking a better graph division. Negative M_C defines part of the graph, which is definitely not a community.

Modularity of a network is then defined as

$$M = \sum_{C=1} M_C \tag{2.4}$$

Graph clustering with high modularity implies good division and makes it reliable to compare different results. The main advantage of modularity is the naturally greater context it captures. The modularity is still fast to compute, which is greatly exploited in different algorithms optimizing the modularity value.

2.3 Algorithms

2.3.1 Kernighan-Lin Algorithm

One of the first attempts to formalize graph clustering comes from Kernighan and Lin [5]. The main motivation of his research was to evaluate different approaches to electronic circuit design, where the number of connections between different circuit boards has to be minimal. We can see the analogy to community detection as we are trying to find such partitioning of the weighted graph that results in dense subgraphs with sparse connections between different communities.

As the result of the research, a simple 2-way graph partitioning algorithm is suggested. We start with random graph bipartition. In each epoch, we swap

nodes in a defined manner between those partitions to improve the clustering metrics. We return the result when an exit condition, such as the number of epochs is met. Kernighan-Lin algorithm is a good heuristic even for larger inputs. However, there is no guarantee the result is global optimum.

Define *internal cost* I_i as the sum of the weight of incident edges of i, which are in the same partitioning as i. Then define *external cost* E_i as the sum of the weight of incident edges of i, which are in different partitioning as i. We can observe that if we switch the node to another partition, values of E_i and I_i are swapped. Thus define node gain D_i as the result of swapping partition for a node i.

$$D_i = E_i - I_i \tag{2.5}$$

As we are interested in exchanges of pairs of nodes between two partitions, we are interested in edge gain J_{ij}.

$$J_{ij} = D_i + D_j - 2 * w_{ij} \tag{2.6}$$

This equals to the exchange of pair of nodes between the partitions with the adjustment of (i, j) edge. This edge stays as part of the external gain of both nodes, thus the need for adjustment.

The algorithm then proceeds as follows. At each epoch, $k/2$ node exchanges are performed heuristically. At each exchange, a pair of nodes with maximum gain J_{ij} is selected. It is important to note the exchanges are done virtually, remembering the gains, adjusting the partitions, and recalculating the J_{ij} as if the exchange was performed. However, the actual input for the current epoch is left unchanged. In the next step, such $1 \leq k \leq n/2$ is found the overall gain is maximalized. Overall gain is the sum of partial gains from each individual exchange. If for any k holds the overall gain is negative, the algorithm terminates as no exchange can be performed to improve the partitioning.

According to Aggarwal [1], the algorithm converges rapidly to the local optimum and may terminate in as little as five epochs. However, as with any other NP-hard problems, the local optimum may differ from the global one. A single epoch can be amortized as $\mathcal{O}(N\log(L))$.

The algorithm can be sped up significantly by considering only nodes instead of edges at each exchange and using smart data structures to store the nodes and their gains in each partition. In addition, a balancing ratio can be defined to allow for unbalanced partitioning. Those improvements were suggested by Fiduccia and Mattheyses and reduced the epoch time to $\mathcal{O}(N)$.

2.3.2 Recursive Kernighan-Lin

A simple extension of any bipartition algorithm is by doing bipartitions recursively. In this subsection, we will discuss the suggested extension of the Kernighan-Lin

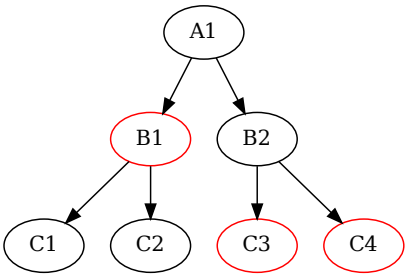


Figure 2.1 Example of recursive tree expansion. Red nodes are selected community partitions splitting the input graph to 3 communities.

algorithm. Instead of bipartition, we will create a whole partition tree and split the graph into k communities.

In the first step of the algorithm, we create a partition tree by recursively applying bipartition until we have a full tree consisting of T leaves, where T is nearest bigger power of 2 such that $K \leq T$. In the next step, we decide which leaves we want to keep to reduce the tree's last layer to contain exactly K leaves. We can use clustering performance metrics of our choice, however in the thesis, we use Kernighan-Lin objective. Enumeration of all possible leaf configurations might be potentially exponential, and it is totally dependent on the choice of K. When K is chosen exactly between two powers of 2, the number of configurations is maximal. We can see a simple example in Figure 2.1.

In the assumption the bipartition is producing partitions of similar size on each level of the tree, we split the previous layer in half. This means most of the processing time of the algorithm is spent on calculating the first bipartition. Possible performance gains are in parallelization, where we can parallelize computations on the same level as the tree and configuration evaluation.

We derive the time complexity of recursive bipartition below. We can observe the number of possible configurations is rising exponentially in regard to k. We can show that easily. Consider we set k exactly between two powers of 2. Then we would need to expand exactly half of the nodes to form the leaves of the final division. Asymptotically, this yields $\binom{k}{k/2}$ combinations, and it is the upper bound for the binomial coefficient. After applying Stirling's approximation of factorial, we receive a total of $\frac{2^k}{\sqrt{k\pi/2}}$ configurations. If E marks the complexity of the evaluation of a single configuration, the total complexity of configuration evaluation is $\mathcal{O}(\frac{2^k}{\sqrt{k\pi/2}}E)$. If we consider edge cut as optimization metrics, which

can be calculated in $\mathcal{O}(L)$ time, we get $\mathcal{O}(\frac{2^k}{\sqrt{k\pi/2}}L)$.

We will continue with estimating the cost of the partitioning itself. For simplicity, we start with the time complexity of optimized Kernighan-Lin. Optimized Kernighan-Lin runs in $\mathcal{O}(N)$ time for single bipartition. Considering we have $\log_2 k$ levels of bipartitions, we get

$$\overbrace{\mathcal{O}(N) + 2 * \mathcal{O}(N/2) + 4 * \mathcal{O}(N/4) + ...}^{\log_2 k \text{ times}} \tag{2.7}$$

After simplification, we get

$$\overbrace{\mathcal{O}(N) + \mathcal{O}(N) + \mathcal{O}(N) + ...}^{\log_2 k \text{ times}} \tag{2.8}$$

We can see the total complexity of the division is then $\mathcal{O}(N\log k)$ and the total complexity of the optimized recursive Kernighan-Lin algorithm is then $\mathcal{O}(N\log k + \frac{2^k}{\sqrt{k\pi/2}}E)$.

As mentioned above in Section 2.3.1, complexity of naive Kernighan-Lin implementation is $N\log(L)$. Under the assumption we create balanced bipartition in regards of nodes and edges, we get

$$\overbrace{N\log(L) + 2 * N/2\log(L/2) + 4 * N/4\log(L/4) + ...}^{\log_2 k \text{ times}} \tag{2.9}$$

After simplification, we get

$$\overbrace{N\log(L) + N\log(L/2) + N\log(L/4) + ...}^{\log_2 k \text{ times}} \tag{2.10}$$

By splitting logarithms, we receive

$$\overbrace{N(\log(L) - 0) + N(\log(L) - \log(2)) + N(\log(L) - \log 4) + ...}^{\log_2 k \text{ times}} \tag{2.11}$$

We can write it asymptotically as

$$\mathcal{O}(N\log(k)\log(L)) \tag{2.12}$$

and the total complexity of recursive naive Kernighan-Lin is then

$$\mathcal{O}(N\log(k)\log(L) + \frac{2^k}{\sqrt{k\pi/2}}E) \tag{2.13}$$

2.3.3 Girvan-Newman

In 2004, Newman and Girvan [6] suggested a novel approach to community detection using information flow. Instead of traditional weights, this algorithm uses edge costs c_{ij}. The main intuition is that the bigger the weight, the smaller the edge cost. Inversion or negation are good heuristics to estimate the edge costs.

The main metric in the Girman-Newman algorithm is betweenness centrality as defined in Section 1.3.3. Girvan and Newman proposed that edges with high centrality tend to connect different communities. This is explained by information flow, as nodes and edges with high centrality lie on more shortest paths. Tight edges can then be marked as communication bridges, connecting different communities. This is also explained by the anatomy of the community, where nodes inside a community are densely connected, whereas there are only a few edges connecting different communities.

This gives us a simple divisive algorithm. In the first step, we calculate the edge betweenness of the edges in the network. Until convergence, we then remove edges with the highest betweenness and recalculate the values. Since in this algorithm, every connected component is considered an independent community, the algorithm finishes when we have found the desired number of connected components.

The main disadvantage of this algorithm is the time complexity, as finding all shortest paths is slow by nature. This comes with a time cost of $\mathcal{O}(V^3)$, which might make it unfeasible for bigger networks.

2.3.4 Greedy modularity algorithm

One of the first successful attempts to exploit modularity as defined in Equation (2.4) is captured in the simple greedy algorithm. This algorithm as proposed by Newman [7] does greedy modularity maximization ensuring the community clustering is optimal

Define ΔM_{AB} as the difference in modularity when we merge communities A and B in comparison to modularity of the graph when they are separate. According to Barabási and Pósfai [4] for undirected graph holds

$$\Delta M_{AB} = \frac{l_{AB}}{L} - \frac{deg_A \times deg_B}{2L^2} \tag{2.14}$$

where l_AB is sum of edge weights connecting communities A and B and deg_A and deg_B are weighted degrees in community A, resp. B. L is then the weighted sum of the edges of the whole graph.

For directed graphs, we then define

$$\Delta M_{AB} = \frac{l_{AB}}{L} - \frac{\sqrt{deg_A^- deg_A^+}\sqrt{deg_B^- deg_B^+}}{L^2} \tag{2.15}$$

where $deg^- A$ is weighted indegree and $deg^+ A$ is weighted outdegree of the community A, resp. B.

The algorithm is then based on an agglomerative approach and works as follows:

1. Each node forms a community on its own at the start of the algorithm
2. Merge 2 interconnected communities with biggest ΔM_{AB}
3. Repeat until we have more communities to merge
4. Return partition with maximal modularity

Calculation of ΔM_{AB} is calculated using constant time, each merge then takes at most $\mathcal{O}(L)$ computations. In summary, the algorithm runs in $\mathcal{O}(NL)$ time and in $\mathcal{O}(N^2)$ when the graph is sparse.

2.3.5 Louvain algorithm

Modularity-based approaches have been the breakthrough in community detection. However, there are some improvements possible, both in times of the speed of the algorithm and in the quality of the results. In this subsection, we will introduce the Louvain method, an efficient algorithm for community detection. Louvain algorithm was introduced by Blondel in 2008 [8].

It is a simple extension of the greedy modularity approach defined in Section 2.3.4 and is reusing already defined terms and definitions. The algorithm works in 2 phases, which are repeated iteratively until convergence.

The algorithm starts by making each node an individual community, setting the number of communities in the beginning to N In the first step, for each node i and all its neighbors j, we evaluate modularity gain by removing node i from its community and placing it in the community of j. Node i is then placed to such a neighboring community where the modularity gain is maximum, under the condition the gain is positive. Those steps are repeated sequentially for every node i in the graph until there is no possible positive change in modularity. When no further positive adjustment to modularity is possible, the first phase finds the local optimum, and the algorithm proceeds to the second phase.

Define ΔQ as a difference in the modularity after moving individual isolated node i into community C. Then ΔQ can be defined as

$$\Delta Q = \left[\frac{\sum_{in} + k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m} \right)^2 \right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m} \right)^2 - \left(\frac{k_i}{2m} \right)^2 \right] \tag{2.16}$$

where $\sum_{in}$ is the weighted sum of the edges inside community C, $\sum_{tot}$ is the weight sum of edges incident to nodes in community C, k_i is the weighted sum of edges incident to node i, $k_{i,in}$ is the weighted sum of links from i to nodes in the community C, and m is the weighted sum of all graph edges. Similar formula can be derived for removing node i from its community. Total modularity gain is then composed of both components, where i is removed from its original community and placed into a community of node j.

In the second phase of the algorithm, a new network based on the clustering from the first phase is constructed. Nodes of a single community are contracted into a single node. Edges are contracted as well, where the weight of the edge connecting two contracted communities is the sum of the weights of the original edges connecting nodes of those communities. Edges inside the community will form a self-loop naturally. When the second phase is finished, this new network can be processed by the first phase of the algorithm, and the process repeats.

The algorithm finishes when no more changes can be made, namely when the first phase of the algorithm can not make any positive modularity gain. Louvain algorithm can terminate after a fixed number of epochs as well, however, this is not used in practice.

It is worth to note the algorithm spends most of the time in the first iteration of the first epoch. This is caused by decreasing number of communities as the algorithm proceeds. In typical social networks with low density, the algorithm finishes in linear time and belongs to one of the most performant algorithms.

2.3.6 Spectral clustering

A spectral clustering method is an efficient approach to community detection, according to Aggarwal [1]. The main idea is to create such embedding of the graph into k-dimensional space where standard machine learning methods could be used.

The main expectation from such embedding is that the stronger the connection between nodes of the graph, the closer are the nodes in the graph embeddings. Formally, assume $W \in \mathbb{R}^{n \times n}$ is a weighted adjacency matrix, where w_{ij} is the weight of the edge (i, j). Assume the nodes themselves are not weighted. For simplification assume we are looking for one-dimensional embedding $y = (y_1, ..., y_n)^T$

of the nodes $N(G) \in \{1, ..., n\}$. Define the objective function to minimize as

$$O = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}(y_i - y_j)^2 \tag{2.17}$$

We can see for the objective to be minimal, the node distance should be inverse proportional to the edge weight. Minimizing the objective function ensures the densely connected nodes are close together.

Laplacian matrix L is defined as $L = \Lambda - W$, where Λ is diagonal matrix satisfying $\Lambda_{ii} = \sum_{j=1}^{n} w_{ij}$. The objective function can then be rewritten as

$$O = 2y^T L y \tag{2.18}$$

Laplacian matrix L is positive semidefinite since the objective function is always non-negative. This results in nonnegative eigenvalues. Since we need to exclude trivial solution $y = (0, ..., 0)^T$ from optimization, we define a scaling constrain

$$y^T \Lambda y = 1 \tag{2.19}$$

After solving Lagrangian optimization, optimal solution is held by

$$\Lambda^{-1} L y = \lambda y \tag{2.20}$$

It can be seen that λ is eigenvalue and y is eigenvector of matrix $\Lambda^{-1}L$. Also, it can be proven the objective function $O = 2\lambda$ for eigenvector y.

The optimal solution is the smallest nontrivial eigenvalue λ and respective eigenvector y. This can be seen from trivial solution $y = (1, ..., 1)^T$, where by definition $O = 0$ and $\lambda = 0$. This solution contains no added information and must be omitted. That way, y holds one-dimensional graph embeddings with the smallest possible optimization function.

We can easily extend the embedding to k dimensions. Consider k smallest nontrivial eigenvalues $0 < \lambda_1 \leq ... \leq \lambda_k$ and corresponding eigenvectors $e_1, ...e_k \in \mathbb{R}^n$ of matrix $\Lambda^{-1}L$. Define matrix $D = [e_1, ..., e_k] \in \mathbb{R}^{n \times k}$. Then each row of such matrix corresponds to the k-dimensional embedding of the graph node, and for $k = 1$, it is identical with the one-dimensional case. It is good to mention that columns are not necessarily orthogonal and are not in the L2 norm, thus later normalization is required.

To reveal communities in a graph, the K-Means algorithm can be used to graph embeddings. That said, we are trying to find densely populated and isolated subspaces of k-dimensional space.

Finding eigenvectors and eigenvalues from sparse matrices according to the ARPACK library [9] takes $\mathcal{O}(nD^2)$ time, where D is the number of eigenvectors

and the dimension of the embedding. ARPACK is a state-of-the-art library for linear algebra implementing the Implicitly Restarted Arnoldi Method to efficiently find a defined number of eigenvectors according to sorting criterium. The time complexity of the K-Means algorithm for further clustering is $\mathcal{O}(nkD)$ for a fixed number of iterations where k is the number of communities to find. The total complexity of the algorithm is then $\mathcal{O}(ND^2 + NkD)$.

The main advantage of this approach is the use of standard linear algebra methods and algorithms. In practice, the underlying algorithms are usually fast and optimized and will benefit from future research, even outside community detection fields. In addition, social networks are sparse graphs in practice, which allow for even more optimization in terms of sparse matrix representation and parallelization, enabling great scalability of the method.

2.3.7 METIS - Multilevel graph partitioning

The main problem with many graph clustering algorithms is they do not scale well. That being said, the complexity of the algorithms becomes an issue as the network grows larger, with the algorithm becoming unusable for bigger networks.

Parthasarathy, Ruan, and Satuluri [3] in his article suggests using Multilevel graph partitioning. Multilevel graph partitioning (METIS) is a method for graph compression, allowing to use standard algorithms on compressed data. METIS works in multiple interconnected phases:

1. *Coarsing.* This phase creates a condensed graph of a similar structure as the original graph with collapsed nodes and edges, enabling faster calculation. The resulting graph is just a fraction of the original, and it is acceptible to go as low as 100 nodes.

2. *Partitioning.* This phase applies the black box graph splitting algorithm and creates graph clustering.

3. *Uncoarsing.* In this phase coarsened partition is mapped to the original graph.

Coarsening phase

In this phase, sequence of graphs $G = G_0, G_1, ..., G_m$ is created, where $|G_j| > |G_{j+1}|$ holds. At each step k, a small number of nodes and edges is contracted to form a single node in G_{k+1}. Aggarwal [1] defines multiple methods of graph compression, especially node and edge selection.

Random edge matching Random node is selected. If there is at least one adjacent unselected node, then such node is selected at random and such pair

will be contacted. Contracting in a single epoch is made until there are any unselected nodes.

Heavy edge matching This approach is similar to random edge matching, with the difference in edge weight taken into consideration. For every unselected random node i, unselected adjacent node j is selected such that w_{ij} is maximal. The main intuition is that heavy edges tend to connect nodes within the community and are not part of the partition cut.

Heavy clique matching It is beneficial to merge densely connected sets of nodes, as this approach will maximalize the number of contracted edges. Define v_i as the number of contracted nodes the node i represents (can be weighted sum in case of weighted nodes), and s_i denotes the weighted sum of collapsed edges at node i from previous contracted phases. We can observe, that if contracted node i is a clique in original graph, s_i will approach $\frac{v_i(v_i-1)}{2}$. Define edge density $\mu_{ij} \in (0, 1)$ of (i, j) edge as

$$\mu_{ij} = \frac{2(s_i + s_j + w_{ij})}{(v_i + v_j)(v_i + v_j - 1)}$$

Contracting high-density edges thus correlates with contracting cliques in graph G_0. Usually, the first node of the edge is selected at random.

Partitioning phase

In this phase, any partitioning algorithm is used to get a partition of the condensed graph. Algorithms such as spectral clustering, Girvan-Newman or recursive Kernigan-Lin can be used to obtain a high-quality approximation of original graph clustering. Even low-quality partitioning can produce reasonable output, mostly because densely connected parts of the graph are contracted, avoiding any partition cuts.

Uncoarsening phase

Mapping between graphs $G = G_0, ..., G_k$ is applied in reverse order to obtain partitioning of the original graph from the condensed variant. It is possible to extend the approach with fine-tuning between every transition. If this extension is used, then G_l is the initial seed for clustering algorithms such as Kernigan-Lin to obtain partitioning for G_{l-1}. This approach can uncover hidden structures in collapsed parts of the graph.

2.3.8 Nested variants of community detection algorithms

As different algorithms have different optimization objectives, the single algorithm might not be the optimal choice for the data selected. This shortcoming might be overcome by combining different algorithms into a hybrid approach. The combined nested algorithm works in multiple steps: We choose k (preferably different) algorithms. At each level of nesting, we take the input graph and run a single algorithm from the pipeline on the data. Each cluster returned will serve as an input for the next layer when we consider a subgraph defined by the community.

We can imagine it as performing community detection with different sensitivity settings. We start with a rough clustering, making it finer at each step. When we proceed with all selected algorithms, we have a final clustering of the original graph. A division tree is built in the process, with leaves representing the final results.

From now on, we will consider only a combination of two algorithms. There are multiple properties the individual algorithms must follow. Ideally, the first algorithm must have short runtime and create good rough clustering. There is also a requirement to have an adjustable number of communities detected so that we can regulate the roughness of the first clustering. For the second algorithm, there are no hard requirements. However, it is useful to set different algorithm as in the first step. Also, it might be beneficial to select the algorithm that works best when used on a small number of nodes to exploit the division of the original data.

Those algorithms were selected for further evaluation:

- Spectral clustering - Louvain
- Recursive Kernighan-Lin - Louvain
- Recursive Kernighan-Lin - Girvan-Newman

This selection of algorithms adheres to the requirements mentioned before and tries to maximalize overall utility. It is important to note that the number of communities in the first step is fixed in future experiments, however, this could be improved upon by automatically determining how rough the initial clustering should be.

2.4 Summary

In this chapter, we have defined multiple important metrics for community detection as well as the most important algorithms relevant to the community

Name	Complexity
Spectral clustering	$\mathcal{O}(ND^2 + NkD)$
Louvain	$\mathcal{O}(N + L)$
Kernighan-Lin	$\mathcal{O}(N)$
Kernighan-Lin Recursive	$\mathcal{O}(N \log k + \frac{2^k}{\sqrt{k\pi/2}} E)$
Greedy modularity	$\mathcal{O}(NL)$
Girvan-Newman	$\mathcal{O}(N^3)$
Kernighan-Lin Naive	$\mathcal{O}(N \log(L))$
Kernighan-Lin Naive Recursive	$\mathcal{O}(N \log(k) \log(L) + \frac{2^k}{\sqrt{k\pi/2}} E)$

Table 2.1 Algorithm complexity summary

detection field in social network analysis. By summarizing the algorithms and their respective complexities in Table 2.1 we observe some of the algorithms are not well scalable for bigger inputs.

From theoretical complexity, the Girvan-Newman algorithm and greedy modularity are both at least quadratic and thus they might be unusable for networks of bigger sizes. The recursive Kernighan-Lin algorithm might be unfeasible in case k is big enough. In that case, the algorithm might face a combinatorial explosion. Those algorithms should be run on a compressed version of the input graphs using the METIS compression algorithm and in that case, we have some guarantee the runtime of the algorithm would be reasonable.

Other algorithms should not have significant problems with performance and, in theory, should be able to process even large datasets in a reasonable time. We will test this hypothesis in the experiment section, where we compare the hypothesis of the runtime with actual implementation.

Chapter 3

Datasets

The choice of the data for research is as essential as the choice of the algorithms itself. In this chapter, we will describe in detail the datasets selected as the benchmark for community detection problem. The main factor driving the selection is dataset size, since as the instance of the problem grows, the priorities in the clustering shift.

As a representative of small social networks, we choose Zachary's karate club. In small graphs, we should achieve a similar length of the algorithm runtime. Although, only a slight change in the clustering can significantly alter the algorithm's accuracy.

For mid-sized networks, we selected the Enron email dataset. For more extensive networks, we are interested more in the speed of the algorithm, as algorithm time complexity might make some algorithms unfeasible. Opposing small datasets, minor deviations of the clustering are much more acceptable, as they have only a tiny impact on the resulting metrics.

3.1 Zachary's karate club

Zachary's karate club is a social network of the university karate club first introduced by Zachary [10] in 1977. The social network was observed from 1970 to 1972. The graph consists of 34 nodes and 78 edges. Edge is weighted by the number of interactions of the club members outside the club.

Due to rising tensions between instructor "Mr. Hi" and the club administrator over the member fees, the club split into two entities. As a result of the split, each former club member was assigned to a new club leader by their preference, effectively labeling each node as either a student of "MR. Hi" or "Officer". Labeling of the club members predetermines the social network to become a benchmark dataset for the study of community structure in social networks. Labels enable us

to view any clustering algorithm as a prediction function, allowing us to compare the results to the original labels.

We have chosen this dataset to represent small networks. With this choice, we are aiming more at the evaluation of algorithm precision and behavior, since in many cases difference in the runtime is insignificant.

3.2 Enron email dataset

Enron was an American company engaged in the energy sector, services, and commodities. The company had a significant share in the energy sector in the USA at that time. It belonged to Fortune 500 companies and was named among the most innovative American companies.

On January 25th, 2002 company filed for bankruptcy when the shares plummeted from $90 to 50 cents. The investigation revealed that the company's financial results were organized accounting fraud, hiding its previous debt and tremendous losses from failed investments. On top of that, the company was extensively involved in the California energy crisis in 2000-2001. Deregulations in the energy industry and market liberalization indirectly caused the crisis. As a result of market manipulations, energy prices skyrocketed, and due to artificially reduced power supply, complete blackouts started to happen. The investigation found leadership guilty of accounting fraud and insider trading.

As part of the investigation, company email communication was released to the public for research purposes, and the dataset was introduced in 2004 by Bryan Klimt and Yiming Yang [11]. For the public release, dataset maintainers restored all integrity issues and due to the request of touched persons, a small amount of email users were removed for privacy reasons.

The dataset contains roughly 500 000 emails. Each dataset file is an email in original data exchange format parsable by standard email parsers. The social network is then constructed by considering each email address as a node, and edges are weighted by the number of communications between two nodes. That way, we obtain the network containing 20 926 nodes and 181 303 edges. It is worth to note we can construct both directed and undirected variants of the Enron dataset. Nonetheless, only the undirected version is used later in the thesis.

We choose this dataset for benchmarking the mid-sized networks, where the run time of the algorithm is taken into consideration as well as its capabilities.

Below is an example of the dataset email. It is easy to see the file follows standard email format and thus is parsable by any standard email parser.

Message-ID: <29020904.1075840230207.JavaMail.evans@thyme>
Date: Mon, 2 Aug 1999 08:33:00 -0700 (PDT)
From: ggalata@enron.co.uk
To: kenneth.lay@enron.com
Subject: Franco Bernabe's tel number
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-From: "Giuseppe Galata" <ggalata@enron.co.uk>
X-To: Kenneth Lay
X-cc:
X-bcc:
X-Folder: \Kenneth_Lay_Dec2000\Notes Folders\Business
X-Origin: LAY-K
X-FileName: klay.nsf

Ken,

I finally found Franco Bernabe's work number: +39-06-3600-4741
+39-06-3600-4742

He actually returned my call (I had left a message with his secretary saying that you were looking to talk to him). He mentioned that IMI (the Italian investment bank) would like to do a joint-venture with us to develop a power trading structure. He will put me in contact with someone at IMI.

You can reach him at any of the above numbers.

Best wishes

Giuseppe Galata
ECT London (Italian Team)

Chapter 4

Experiment setup

In previous chapters, we performed a deep explanation of the social network analysis, introduced key algorithms and approaches to community detection and set up benchmarking datasets. This chapter is then dedicated to defining key experiments of this thesis, consuming previously gained knowledge.

4.1 Estimation of number of communities

In Chapter 2, we defined multiple algorithms performing community detection in social networks. We can observe many of the algorithms works similarly to K-Means, where the number of clusters is defined as a parameter k. This algorithm's common property puts us in a chicken and egg problem, where we need to know k to obtain optimal clustering. However, we do not know k in advance. For Zachary's karate club, the choice of k is simple: We will be consistent with the labels of the original dataset.

For the Enron email dataset, things are more complicated. Since we have no labels or external information about the expected graph structure, we must estimate k by ourselves. This estimate of community count will be processed later in Section 4.3. We will use this estimate for every algorithm where k is a hyperparameter to enable fairness in algorithm evaluation. An alternative to this approach is to fine-tune each individual algorithm for the optimal partition of the graph, however, this method is even more expensive in terms of computational resources.

In this experiment, we estimate the count of communities in the Enron dataset by performing grid search using spectral clustering. For each possible clustering with a different cluster count, we calculate 3 different metrics:

- inertia

- silhouette
- edge cut as defined in Section 2.2.1

Inertia is defined as

$$\sum_{i=1}^{n} \min_{\mu_j \in C}(\|x_i - \mu_j\|^2) \tag{4.1}$$

where μ_j is the respective cluster center. This metric represents objective function of K-Means and the function represents the sum of squared distances of data points from the nearest cluster centers. Minimal inertia means perfect clustering, as that would mean data points are close to cluster centroids. This approach evaluates the performance of K-Means clustering on spectral embeddings and does not use the original input graph.

Silhouette value for data point i is defined as

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{4.2}$$

where $a(i)$ is the mean distance of the data point to the other points in the same cluster and $b(i)$ is the mean distance of the data point to the nearest points from another cluster. Silhouette score $s(i)$ is scaled to [-1, 1] where 1 means perfect clustering with dense, well-distinguished cluster, 0 means insignificant clustering regarding distances, and -1 interprets as wrong cluster assignment. The silhouette score for the whole dataset is then calculated as the average silhouette score through all data points. The objective is to find k where the silhouette score is maximal.

Results of individual metrics are collected and analyzed. We then try to find either the global optimum of the data or use the elbow method to find a point where any future gains are insignificant. The elbow point is found using a second-order derivative of the curve. To obtain a second-order derivative of the original data, we will use 2 approaches. First, the polynomial curve is fit using observed data points. Secondly, the second-order central difference is calculated directly to approximate the second-order derivative.

A simple derivative approximation for a single dimension can be defined as

$$\partial_h[f](x) = \frac{f(x+h) - f(x)}{h} \tag{4.3}$$

For more precise calculation, *findiff* Python library [12] is used, which is based on Taylor series expansion.

As a result of this experiment, such k is chosen, which is expected to work well with all algorithms on the Enron dataset.

4.2 Hierarchy of non-hierarchical method

Algorithms such as greedy modularity or Kernighan-Lin work in either an agglomerative or divisive approach. That means it is possible to track down how individual community was formed, and it eventually means we have access to the whole partitioning tree. That stability and predictability is a major flaw in the spectral clustering method.

As defined in Section 2.3.6, spectral clustering is a two-step process. Firstly, multidimensional graph embedding is created, and then K-Means is used to determine the clustering. From the very definition of K-Means, results are greatly dependent on initialization, which is based on randomness and therefore are unstable. This holds particularly true when we consider clusters with partially unclear boundaries, where some boundary points are assigned differently.

K-Means clustering on the same graph embedding becomes much more interesting as we change the number of clusters we want to identify. This stems from the fact that it is new clustering, not using any information from clusterings with different cluster counts k. This opens up questions about behavior on $k \to k+1$ change and how adding a new community will reshuffle community label assignment.

For this experiment, assume we have source graph G_s and target graph G_t, both containing the same nodes and with community labels assigned. Assume we are observing $k \to k+1$ case and the difference in respective cluster counts is one. Then for the community label s in the source graph and label t in the target graph we calculate alignment $P(t|s)$ as the probability of transition from label s to t. The alignment can be estimated using maximum likelihood estimation as

$$P(t|s) = \frac{c(s,t)}{\sum_{t'} c(s,t')} \tag{4.4}$$

where $c(s_l, t_l)$ is a count function, marking the number of nodes that have s_l community label in the source graph and t_l community label in the target graph. It is easy to see the alignment calculates the percentage of how the source graph is split into target graph communities.

Another observation is that clustering itself can be seen as the division of nodes into equivalence classes, as reflexivity, symmetry and transitivity property hold trivially and we are finding alignment between such equivalence classes.

Behavior such as instability and randomness is expected and will be subject to examination for the Enron dataset. To obtain a bigger picture of how this splitting behaves, we will examine the sequence of clusterings $k \to k+1 \to k+2$. In this experiment, we will calculate alignments for respective pairs of labeled graphs and create a visualization of observed data.

4.3 Algorithm comparison

The main and most interesting part of this thesis is the evaluation of community detection algorithms on different data types. This section is devoted to the definition of a methodology for execution and result evaluation for differently sized datasets and algorithms.

4.3.1 Datasets

As mentioned in Chapter 3, we have chosen two differently-sized benchmark datasets. To represent small data, Zachary's karate club is the perfect option. In addition, nodes are labeled, allowing us to evaluate the accuracy of individual algorithms.

As the representative of mid-range data size, we have chosen the Enron email dataset. This network is large enough to test and evaluate the performance of community detection algorithms and find algorithms suitable for bigger networks. Opposing to Zachary's karate club dataset, Enron is unlabeled, adding additional burden to the estimation of the number of clusters.

4.3.2 Statistics and measures

To measure the performance of community detection algorithms, we select metrics describing the core properties of any partition of the input graph. To mitigate randomness, each algorithm is evaluated 5 times and for each algorithm run we calculate the statistics for that particular run. Aggregated statistics are then calculated from all runs of the same instance of the algorithm, calculating mean, sample standard deviation, and 95% confidence interval. Selected aggregated properties include:

- runtime of the algorithm
- community count and community size
- modularity and community average modularity
- conductance and community average conductance
- normalized edge cut and community average normalized edge cut
- edge cut ratio
- edge cuts
- clustering coefficient and average community clustering coefficient

- graph density and average community density

We select those properties because they are either direct objectives of the community detection problem or are tightly connected to how the ideal community is perceived. Quantitative analysis will strictly consider only these properties.

4.3.3 Algorithm configuration

Zachary's karate club

Since Zachary's karate club is a dataset of small size, the comparison consists only of basic algorithms as there are only a few nodes to test on. That means there is no graph compression using the METIS algorithm, nor is there the use of nested algorithms or recursive Kernighan-Lin. Algorithms and their parameters are listed in Table 4.1.

Name	Notes
Kernighan-Lin	20 iterations
Kernighan-Lin Naive	20 iterations
Louvain	-
Greedy modularity	-
Girvan-Newman	K=2
2D spectral clustering	K=2, 3, 4
3D spectral clustering	K=2

Table 4.1 Algorithms evaluated for Zachary's karate club.

Since all of the actors in the dataset are labeled by one of the two labels, we will also measure algorithm accuracy in addition to the previously mentioned metrics. In the modularity-based approaches, the number of communities is detected automatically and is not subject to the parameter.

Enron enail dataset

As this dataset is considerably larger than Zachary's karate club, we have a greater choice of algorithms available. On the other hand, due to the algorithm's theoretical complexity, we now have time constraints as mentioned in Chapter 2. This means the graph compression algorithm METIS must be used to enable the evaluation of bigger datasets. Algorithms where METIS compression is used are listed in Table 4.2, and their runtime is calculated using compressed graphs. Algorithms, where theoretical time complexity suggests they can process large input graphs, are listed in Table 4.3. Those algorithms are not compressed, and their run time matches the processing time of the original data.

As spectral clustering is directly optimized for parallel processing, we allow it to use all CPU cores to measure end-to-end performance. This results in the serial execution of all instances of spectral clustering. All other algorithms are executed simultaneously, allocating only a single CPU core for each algorithm run. This setup greatly improves the run time of the experiment as a whole and still let the algorithms run uninterrupted.

Name	METIS	Notes
Kernighan-Lin Naive	4000 nodes	25 iterations
Kernighan-Lin Naive Recursive	4000 nodes	25 iterations
Girvan-Newman	4000 nodes	-
Greedy modularity	1000 nodes	-
Kernighan-Lin recursive → greedy modularity	4000 nodes	see notes

Table 4.2 Algorithms evaluated for Enron dataset using METIS compression. Since we estimate the optimal number of communities in Section 4.1, we use the community count parameter from this experiment where applicable. In nested algorithm Kernighan-Lin recursive → greedy modularity, we will use the nearest lower power of 2.

Name	Notes
Kernighan-Lin	25 iterations
Kernighan-Lin Recursive	25 iterations
Louvain	-
10D spectral clustering	-
30D spectral clustering	-
Kernighan-Lin recursive → Louvain	see notes
Spectral clustering → Louvain	see notes

Table 4.3 Algorithms evaluated for Enron dataset. Since we will estimate the optimal number of communities in Section 4.1, we use the community count parameter from this experiment where applicable. In nested algorithm Kernighan-Lin recursive → Louvain, we will use 2 lowest powers of 2 and in Spectral clustering → Louvain we will use lowest power of 2.

Chapter 5

Results

5.1 Estimation of number of communities

In this chapter, we estimate the optimal number of communities of the Enron dataset in an experiment defined in Section 4.1. For every $k \in \{2, ..., 3500\}$, we calculate inertia, silhouette score and edge cost of clustering with k clusters using a 30D spectral clustering algorithm. Results are then stored in a CSV file for further processing. All calculations are done in parallel, and calculations of all possible clusters took 21 hours.

For all mentioned measures, we plot their respective values for each k. In addition, we fit the polynomial curve on observed data and calculate the second derivative of that curve. To cross-check the results, we calculate the second-order central difference directly from observed data. To illustrate how the curve behaves, we also calculate the difference of values for all neighbor pairs.

In the next subsections, we describe each of the metrics independently and we will conclude by setting a K for the next experiments.

5.1.1 Inertia

Inertia is an optimization objective of the K-Means algorithm. It is a squared distance of data points from the nearest clusters center, thus we want to minimize the function value. As we use K-Means directly on the graph embeddings, we obtain the inertia value directly from the clustering object. Estimating the number of clusters by observing the inertia curve is a standard procedure in machine learning, used mostly with K-Means.

As we can see in Figure 5.1, the graph is convex and decreasing, thus, estimation of elbow point is possible. When we analyze second-order derivative, we come to the conclusion that for $k = 9$ the second-order derivative is maximal. The elbow point has been marked into graphs. Visual inspection of the inertia

and difference curve confirms this discovery, as bigger K contribute less to inertia changes.

5.1.2 Silhouette score

Similarly as in inertia, the evaluation of silhouette score is a standard procedure in machine learning. Since for each data point i higher silhouette score means better clustering, we want to obtain maximum results.

From Figure 5.2 we can see the silhouette score is more noisy and less predictable than inertia. We can see the curve is rising from the lowest possible values to the highest ones, and as a result, there is no maximal value in the calculated interval. The rising tendency is caused by adding more clusters to the graph, which makes clusters naturally denser, as even a small group of close nodes can form a cluster. With lower k, naturally, such nodes are a part of bigger clusters.

Using the curve fitting technique brings no conclusive results, as the second-order derivative is in all cases on a small range of nondecreasing value levels. The second-order central difference suggests there is a local maximum on previously found $k = 9$, however it is not a global optimum, which suggests for $k = 2$. The same applies to examining the difference between two consecutive values, where we alternate between negative and positive values.

5.1.3 Edge cut

The main drawback of the before-mentioned measures is they operate on graph embedding instead of input graph. As edge cut represents one of the main community detection measures, naturally, we want to have it maximal. This points to measuring edge cut metrics for all of the performed clusterings. We can observe all measured values in Figure 5.3.

Similarly, as using silhouette score, edge cut value is noisy, making it impossible to estimate elbow point from second order derivative or central difference. There is as well no global maximum, as the curve has decreasing tendency. This is caused by adding more communities, which naturally add edges connecting different communities. As a result, we might observe denser clusters, however, with much more inter-cluster connections. Inversion of the values happens at $k = 54$ where there are more edges connecting different communities than edges within those communities.

As a result, there is a rapid decrease in edge cut value, providing no significant interpretation of the data.

5.1.4 Conclusion

As we can see from the observed values, the only significant measure is inertia. Other metrics are noisy. Silhouette score provides support for inertia analysis, however, it is incapable of the verdict of its own. Edge cut analysis provided no added value as no elbow point nor optimal value can be determined. To conclude, for further work we will use $K = 9$ as a baseline community count for the Enron dataset.

5.2 Hierarchy of non-hierarchical method

As mentioned in Section 4.2, spectral clustering is not a divisive or agglomerative algorithm, meaning we can not reconstruct the exact process of how communities are formed, and we can not create community dendrogram. In this experiment, we observe community splitting behavior for consecutive community counts. As a baseline, we choose $K = 9$ as per the result of the experiment in Section 5.1. Alignments $K = (9, 10)$ and $K = (10, 11)$ are calculated and visualized.

We can see calculated alignments for $K = (9, 10)$ in Table 5.1 and for $K = (10, 11)$ in Table 5.2. Visualized alignments are in Figure 5.4. It is easy to see the alignment value of 1 is equal to the isomorphism of community labels.

Most of the alignments are either isomorphisms or only small diversions of the original communities. In the $K = (9, 10)$ alignment we can see the only significant diversions of ratio $75 : 25$ are $1 \to 1, 2$ and $5 \to 1, 5$. From the visualization it is easy to see those diversions create a new community labeled 1 for target graph $K = 10$. Similarly by observing $K = (10, 11)$ we come to similar conclusion with $0 \to 9, 2$ and $5 \to 4, 8$ with ratio up to $60 : 40$. These diversions create community label 8 in graph $K = 11$. We can also see the migration of nodes caused by $0 \to 2$. This can be explained by the slight change in node cluster centers by adding a new cluster.

The joint diagram shows that the newly formed community 1 in $K = 10$ is untouched in the next iteration. We can also see many nodes untouched by the clustering through multiple iterations, suggesting parts of the embedding clearly distinguished from the rest of the nodes and parts with ambiguity. From the definition of K-Means, clustering labels might change in ambiguous spaces as different clustering might be found more optimal.

5.2.1 Conclusion

In theory, when we gradually increase the detected community count, we should observe divisions, merges and isomorphisms. Using the Enron dataset, we verified this assumption and observed the behavior of the spectral clustering algorithm.

	0	1	2	3	4	5	6	7	8
0	**0.96**	0	0	0	0	0	0	**0.02**	0
1	0	**0.21**	0	0	0	**0.26**	0	0	0
2	0	**0.79**	0	0	0	0	0	0	0
3	0	0	**1.00**	0	0	0	0	0	0
4	0	0	0	**1.00**	0	0	0	0	0
5	0	0	0	0	**0.07**	**0.74**	0	0	0
6	0	0	0	0	0	0	0	**0.98**	0
7	0	0	0	0	0	0	0	0	**1.00**
8	0	0	0	0	0	0	**1.00**	0	0
9	**0.04**	0	0	0	**0.93**	0	0	0	0

Table 5.1 Results for graph alignment for $K = 9$ and $K = 10$

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	**0.89**	0	0	0
1	0	0	**0.95**	0	0	0	0	0	0	0
2	**0.34**	0	0	0	0	0	0	0	0	**0.61**
3	0	0	0	**1.00**	0	0	0	0	0	0
4	0	**0.07**	0	0	0	**0.77**	0	0	0	0
5	0	0	0	0	**1.00**	0	0	0	0	0
6	0	0	0	0	0	0	0	0	**1.00**	0
7	0	0	0	0	0	0	0	**1.00**	0	0
8	0	0	0	0	0	**0.23**	0	0	0	**0.39**
9	**0.66**	0	0	0	0	0	**0.11**	0	0	0
10	0	**0.93**	**0.05**	0	0	0	0	0	0	0

Table 5.2 Results for graph alignment for $K = 10$ and $K = 11$

New communities were formed as a combination of small parts of two former communities. We can also observe small migration of other nodes, possibly caused by a light shift of cluster centers. This points out that spectral clustering works in a more predictable approach than previously thought, creating a hierarchy of communities in the process. In all examined alignments, we observed most of the communities are mapped (nearly) isomorphically, suggesting clusters are well distinguished at a given scale.

5.3 Algorithm comparison

5.3.1 Zachary's karate club

As theory suggests, community detection is a straightforward task in small datasets. As Table 5.3 shows, the run-time of individual algorithms is extraor-

Name	Mean	STD
Kernighan-Lin	4 ±3 ms	2 ms
Louvain	5 ±2 ms	2 ms
Greedy modularity	16 ±4 ms	3 ms
Kernighan-Lin Naive	24 ±2 ms	2 ms
Spectral clustering 3D (K=2)	42 ±5 ms	4 ms
Spectral clustering 2D (K=2)	48 ±7 ms	5 ms
Spectral clustering 2D (K=4)	49 ±7 ms	6 ms
Spectral clustering 2D (K=3)	51 ±10 ms	8 ms
Girvan-Newman	73 ±7 ms	6 ms

Table 5.3 Zachary karate club algorithm runtime

dinarily small. Algorithm duration is kept under 100ms in all cases, where the fastest algorithm takes 4 ms and the slowest algorithm 70ms to complete the calculation. While the range itself is small, we can see the 20-fold difference between the fastest and slowest variants. This proposes a much bigger difference on a larger dataset.

Analyzing Figure 5.5, we see algorithms discovered either 2, 3, or 4 communities. Algorithms, where the number of communities is determined from the data, found more communities as the dataset's author suggests, namely greedy modularity detected 3 communities, while the Louvain algorithm detected even 4 clusters.

Figure 5.9 shows accuracy for algorithms which found exactly two communities. Kernighan-Lin and Girvan-Newman both achieved label accuracy of 94 %. When we visualize 2D spectral encoding in Figure 5.10, we will discover that the resulting structure resembles at least 3 clusters. Spectral clustering thus might reveal hidden structures and potential communities of the graph. When we observe spectral embedding applied to Louvain algorithm labels in Figure 5.11, we can see the algorithm correctly distinguished most of the communities, and we can also note discrepancies with the spectral clustering in a single case.

Consider key objective metrics modularity and edge cut in Figure 5.6. Edge cut is dominated in Kernighan-Lin, followed by Girvan-Newman. Modularity is dominated by modularity-based methods followed by spectral clustering and Kernighan-Lin. Interestingly, the Louvain algorithm performed best in modularity metrics, however, it was among the worst in edge cut metrics. This does not apply in opposite direction, as Kernighan-Lin performs reasonably well in the modularity objective.

We can also see that spectral clustering detecting the same amount of clusters as the Louvain algorithm performed much worse. This can be compared to overfitting in machine learning, suggesting the algorithms are sensitive to community

count parameter.

If we consider normalized edge cut and conductance in Figure 5.7, we can clearly see that the resulting graphs are similar. As the number of communities grows, both metrics reach higher values. Those metrics are highly similar to regular edge cut objective. To add, observation of Figure 5.8 suggest the community subgraph density in small datasets more relies on community count than on partition structure. For reference, the graph density for Zachary's karate club is 0.139 and graph clustering coefficient is 0.241.

To conclude, we performed tests on Zachary's karate club dataset. As results show, all algorithms perform similarly well on small datasets in both run time and results. That suggests that the choice is dataset dependant and no silver bullet is possible. For edge cut maximization, Kernighan-Lin works best for bipartition problems and Girvan-Newman scores great for any general case.

Main disadvantage of edge cut measure is the requirement to know community count in advance or determine it manually. For cases where k is unknown modularity-based approaches are recommended. Results provided by modularity-based approaches are substantial, however, their representation of the community might differ from suggested clustering in the case of provided labels.

5.3.2 Enron email dataset

Name	Mean	STD	METIS
Spectral clustering (K=10)	5.9 s ±0.3 s	0.2 s	-
Spectral clustering	6.3 s ±0.9 s	0.7 s	-
Louvain	6.7 s ±0.8 s	0.6 s	-
Nested Spectral clusterin - Louvain	8.8 s ±0.2 s	0.2 s	-
Kernighan-Lin	9.5 s ±1.0 s	0.8 s	-
Nested KL Recursive - Louvain (init split K=4)	25.8 s ±2.8 s	2.3 s	-
Kernighan-Lin Recursive	26.6 s ±2.7 s	2.2 s	-
Nested KL Recursive - Louvain (init split K=8)	28.7 s ±3.1 s	2.5 s	-
Greedy modularity	2.2 h ±13.2 m	10.6 m	1000 nodes
Girvan-Newman	3.0 h ±1.7 h	1.4 h	4000 nodes
Nested KL Recursive - Greedy modularity	4.0 h ±59.5 m	47.9 m	4000 nodes
Kernighan-Lin Naive	15.2 h ±12.2 m	9.8 m	4000 nodes
Kernighan-Lin Naive Recursive	19.4 h ±9.7 m	7.8 m	4000 nodes
Nested KL Recursive - Girvan-Newman	23.9 h ±10.5 h	8.5 h	-

Table 5.4 Enron algorithm runtime

As the size of the data grows, the complexity of the community detection problem rises. We will observe a much wider spectrum of observed properties

when we consider mid-sized datasets such as the Enron email dataset. As we can see in Table 5.4, run time of algorithms is as small as 6 seconds ranging to 24 hours. Fast algorithms include spectral clustering, the Louvain algorithm, Kernighan-Lin, Kernighan-Lin recursive and their nested variants. We can also see METIS compression can make complex algorithms feasible by effectively reducing input graph.

As we can see in Figure 5.12, the difference in the number of detected communities is in order of magnitudes. In the most extreme case, 6315 communities were formed with an average size of just 3 nodes. This might be caused by algorithm nesting, where we treat communities as standalone subgraphs, removing information about the rest of the graph in the process. Surprisingly, modularity-based approaches found more communities than expected, where greedy modularity was detected in an average of 30.4 communities and Louvain even 38.4. Even though community count was set as a parameter, the Girvan-Newman algorithm found 25 communities. This stems from how the algorithm works, where we are removing edges with maximal centrality and observing connected components.

When we visualize Girvan-Newman clustering in Figure 5.16 we observe the main difficulty with the algorithm. The community in this algorithm is defined as a graph component. Since the Enron email dataset is not a single connected component, this creates pressure on the algorithm and moves to focus more on the small components of the original graph. As a result, the division is uneven, creating a dense super component in the process.

When we analyze key metrics in Figure 5.13, we see diverse results. Modularity metrics confirm Louvain as state-of-the-art modularity optimization. Greedy modularity did underperform since it can be considered a median of all results. Most of the algorithms are in 2/3 of the maximal modularity, including the bipartition algorithm Kernighan-Lin. Nested algorithms tend to perform better, which might be probably caused by the smallness of detected communities.

Edge cut metrics is heavily correlated with the number and size of detected communities. The best algorithms are either bipartition algorithms or algorithms which create the dense super component in the process since this minimizes the number of edges connecting different communities. This can be verified using edge cut ratio, which confirms only a small portion of edges actually form a bridge connecting different communities. The second group forms algorithms such as spectral clustering, Louvain and recursive Kernighan-Lin. Those algorithms show the great trade-off between actual performance and expected result properties.

Conductance and normalized edge cut can be found in Figure 5.14. In contrary to Zachary's karate club, conductance and normalized cut ordering are more different from edge cut. However, similarly, as with edge cut, the chart is dominated by bipartition algorithms and Girvan-Newman. Nested algorithms perform orders of magnitude worse than their nonnested counterparts. Similar applies to

normalized edge cut, where we can create 3 groups of algorithms divided by their performance.

It is easy to see in Figure 5.15 that greedy modularity, Louvain algorithm and spectral clustering tends to create denser communities compared to nested algorithms and all of the variants of the Kernighan-Lin algorithm. Spectral clustering, recursive Kernighan-Lin, and Louvain also have higher clustering coefficient, which points to a higher number of triadic closures. For comparison, the graph density of the Enron dataset is 0.0008 and the clustering coefficient is 0.00048.

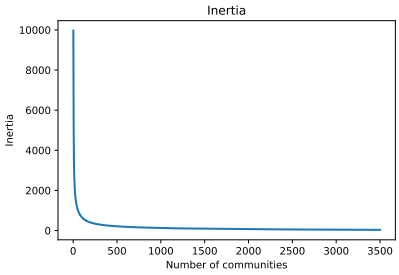
In conclusion, we tested different algorithms on the mid-sized Enron email dataset. Similarly, as with smaller datasets, we identify two use cases of algorithms, depending on whether the number of communities is known beforehand or not.

In the first case, when the number of communities is defined and known, the best approach for community detection is using recursive Kernighan-Lin and spectral clustering. For recursive Kernighan-Lin, choosing K near the power of two is important to ensure the size balance of partitions. In cases where the input graph is a single connected component, Girvan-Newman combined with METIS might be a viable choice. In our case, however, the input graph has 24 independent components, which leads to the creation of a single dense supercomponent.

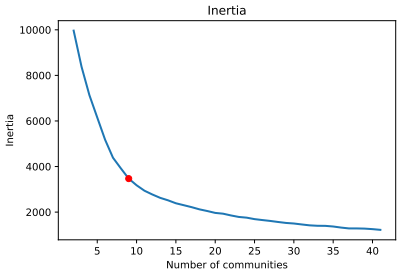
When the number of clusters is unknown, the Louvain algorithm is the only feasible choice. Louvain algorithm proved itself to be a state-of-the-art modularity optimization algorithm. Greedy modularity can also be used in combination with METIS compression, which should provide denser communities, however, on the cost of much higher run time and as a result, the greedy modularity algorithm can not be recommended as a fast universal community detection algorithm.

We can also observe METIS compression greatly reduces the input graph and enables us to evaluate fairly all algorithms. By observing algorithms with METIS applied, we can see there is a difference in performance, however, it can be considered as expected and acceptable. Usually, one of the metrics is affected more, leaving the second metric nearly untouched.

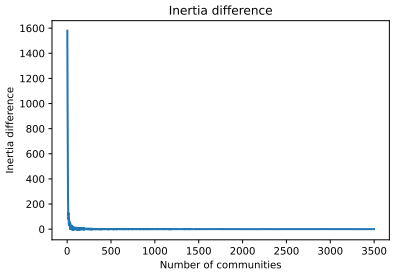
Those results also suggest that for even bigger datasets, spectral clustering, recursive Kernighan-Lin and Louvain algorithm will provide reasonable performance. Those algorithms then form a set of universally good community detection methods.



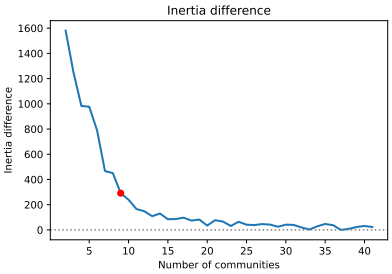
(a) Inertia



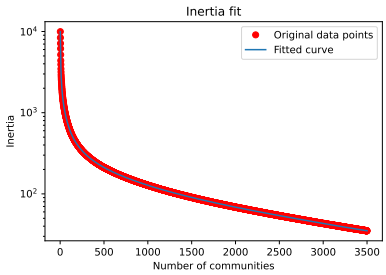
(b) Inertia zoomed



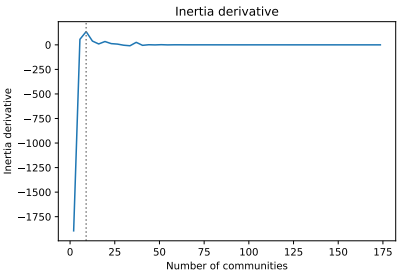
(c) Neighbor difference



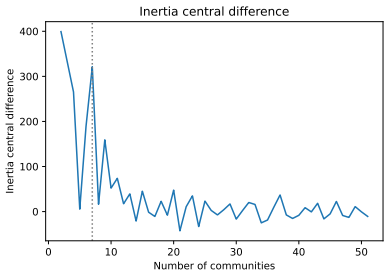
(d) Neighbor difference zoomed



(e) Curve fit

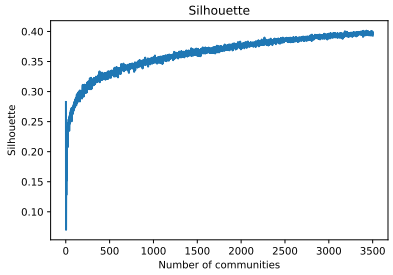


(f) Curve derivative

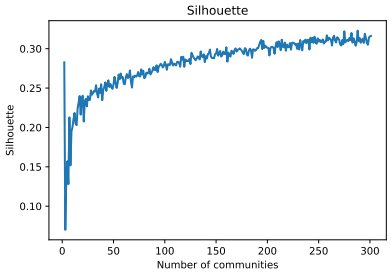


(g) Second order central difference

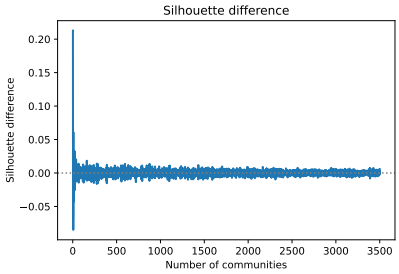
Figure 5.1 Inertia measured data.



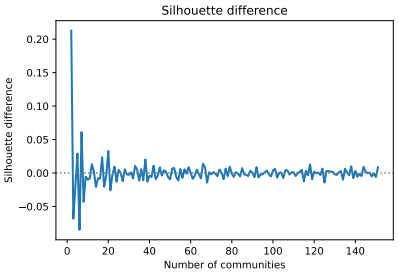
(a) Silhouette score



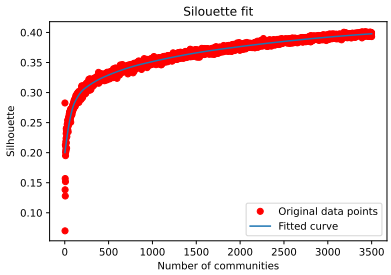
(b) Silhouette score zoomed



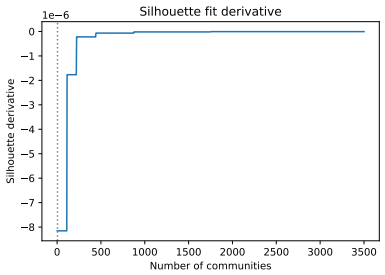
(c) Neighbor difference



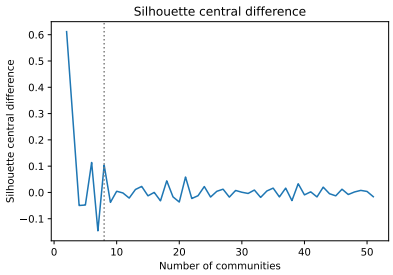
(d) Neighbor difference zoomed



(e) Curve fit

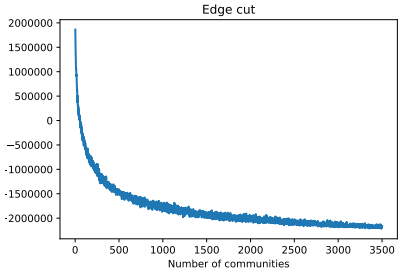


(f) Curve derivative

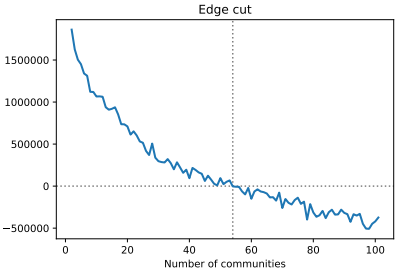


(g) Second order central difference

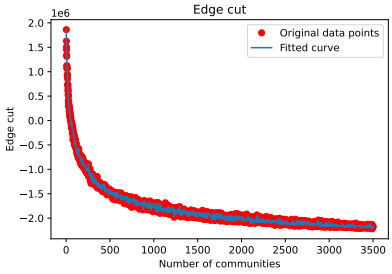
Figure 5.2 Silhouette score measured data.



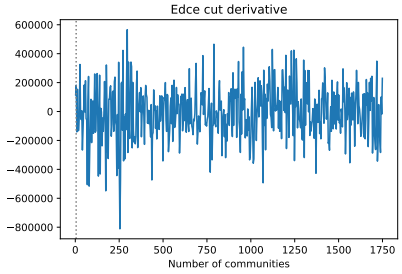
(a) Edge cut



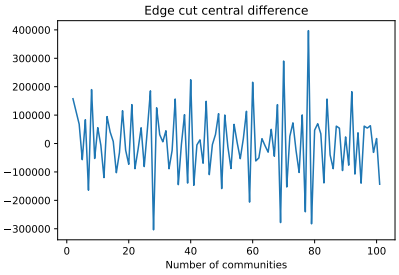
(b) Edge cut zoomed



(c) Curve fit

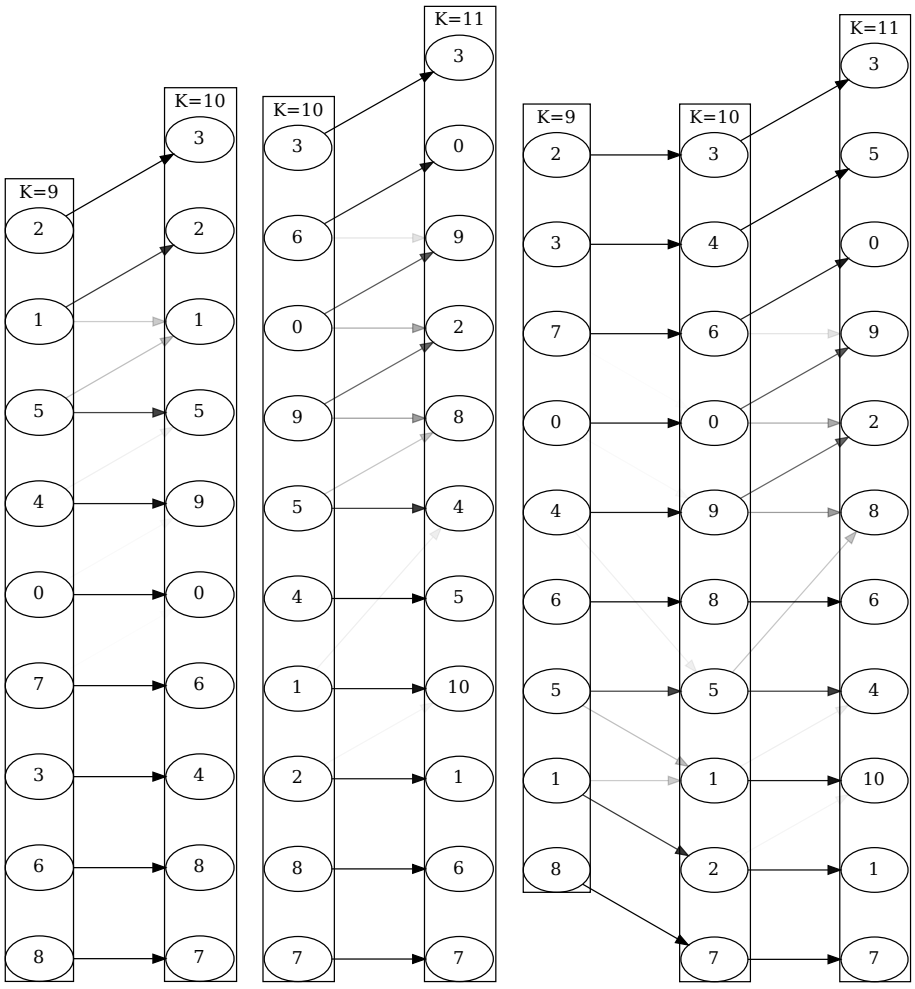


(d) Curve derivative



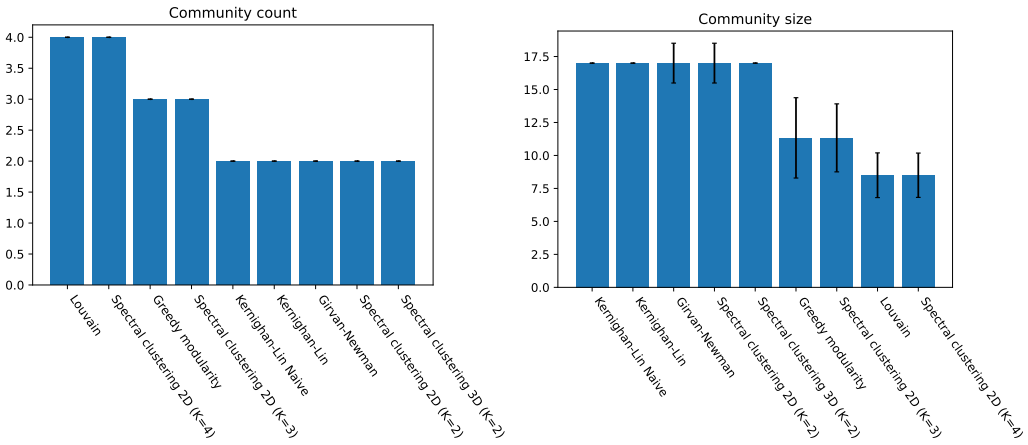
(e) Second order central difference

Figure 5.3 Edge cut measured data.



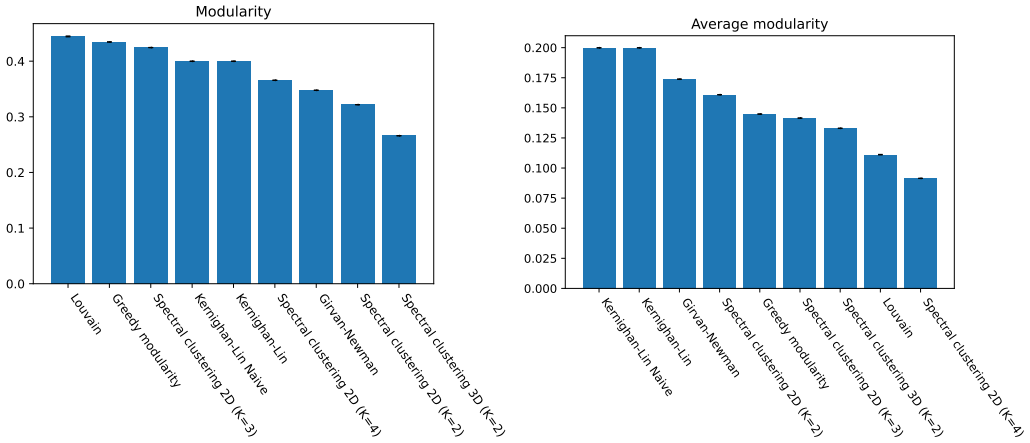
(a) $K = (9, 10)$ **(b)** $K = (10, 11)$ **(c)** Joint diagram

Figure 5.4 Visualization of community splitting.



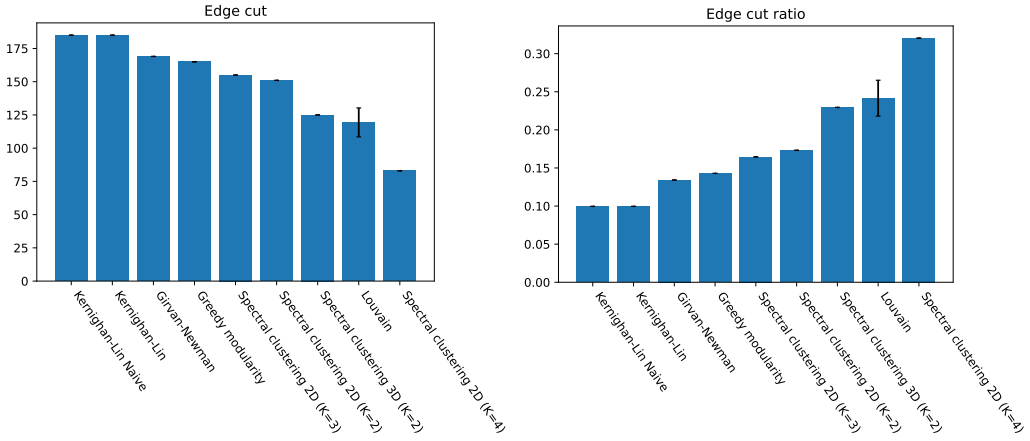
(a) Community count **(b)** Average community size

Figure 5.5 Zachary's karate club: Basic community info.



(a) Modularity

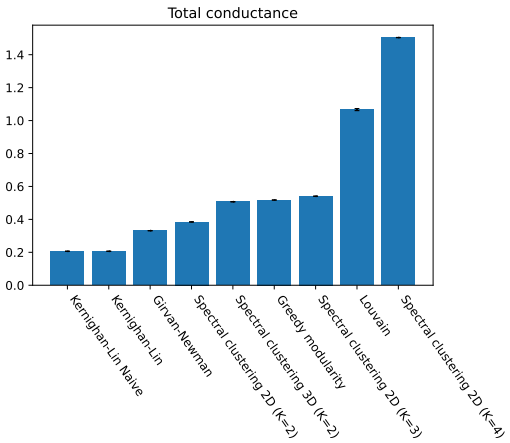
(b) Average community modularity



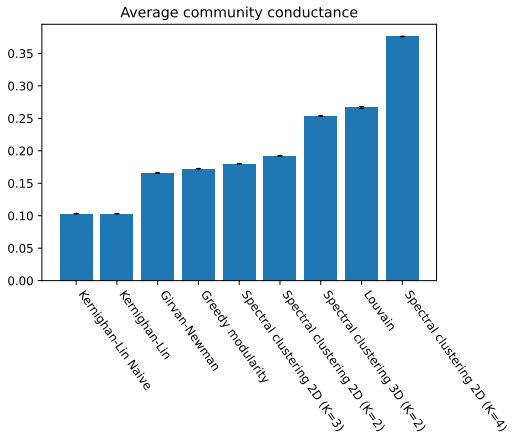
(c) Edge cut

(d) Edge cut ratio

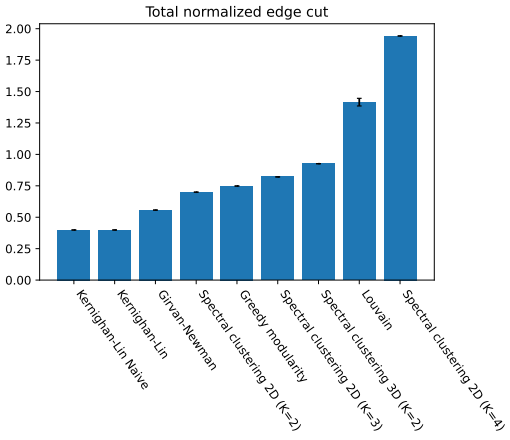
Figure 5.6 Zachary's karate club: Modularity and edge cut.



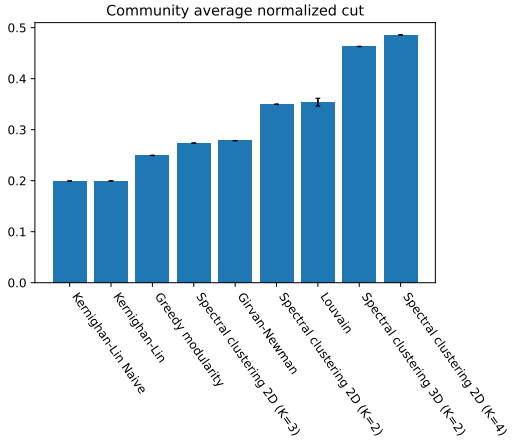
(a) Conductance



(b) Average community conductance

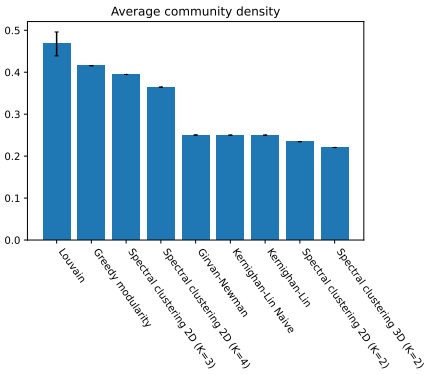


(c) Normalized edge cut

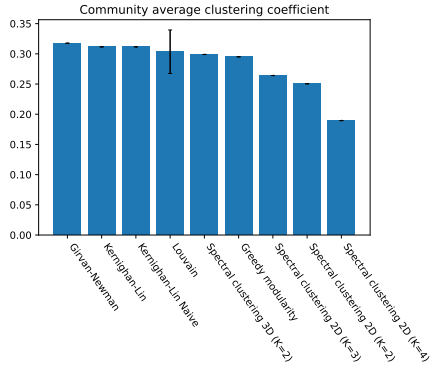


(d) Average normalized edge cut

Figure 5.7 Zachary's karate club: Conductance and normalized edge cut



(a) Average community density



(b) Average community clustering coefficient

Figure 5.8 Zachary's karate club: Density and clustering coefficient

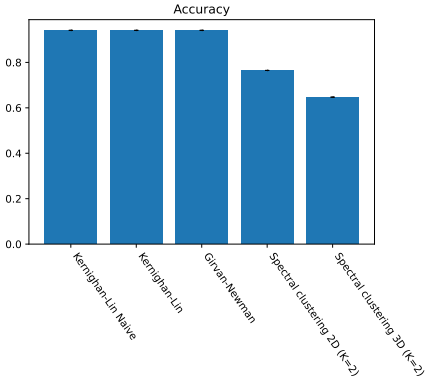


Figure 5.9 Zachary's karate club: Accuracy

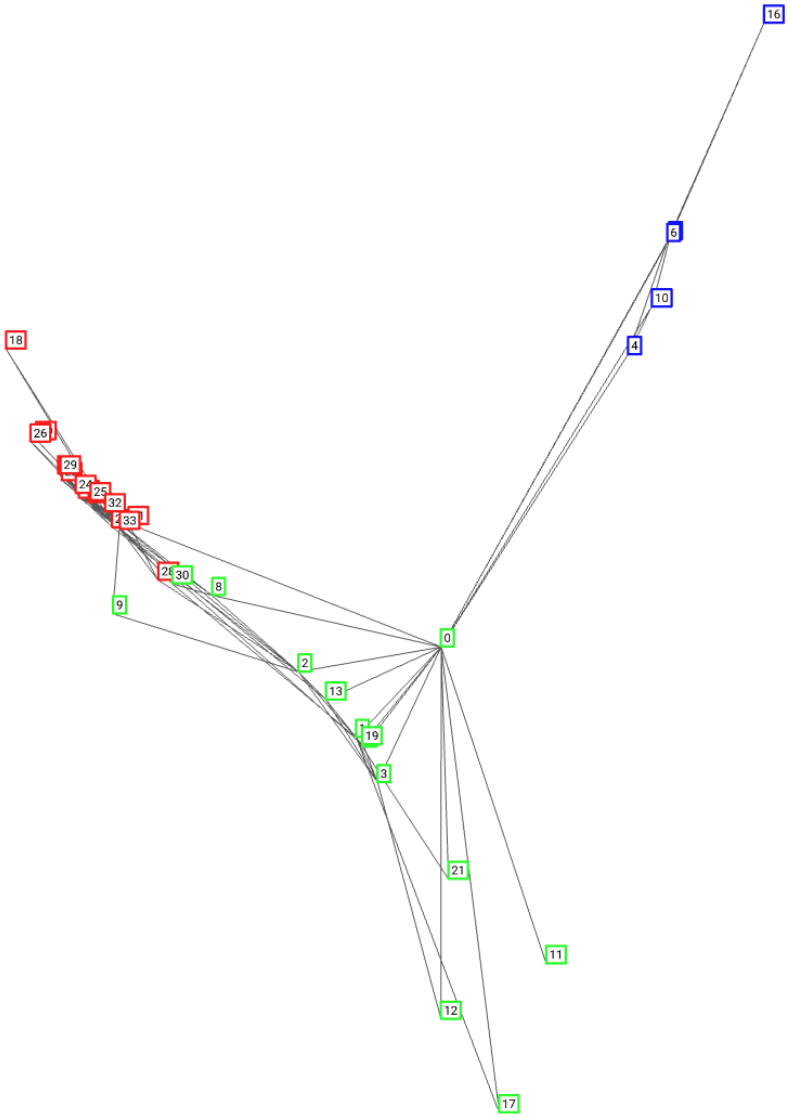


Figure 5.10 Zachary's karate club: Spectral encoding using GraphVisualizer application

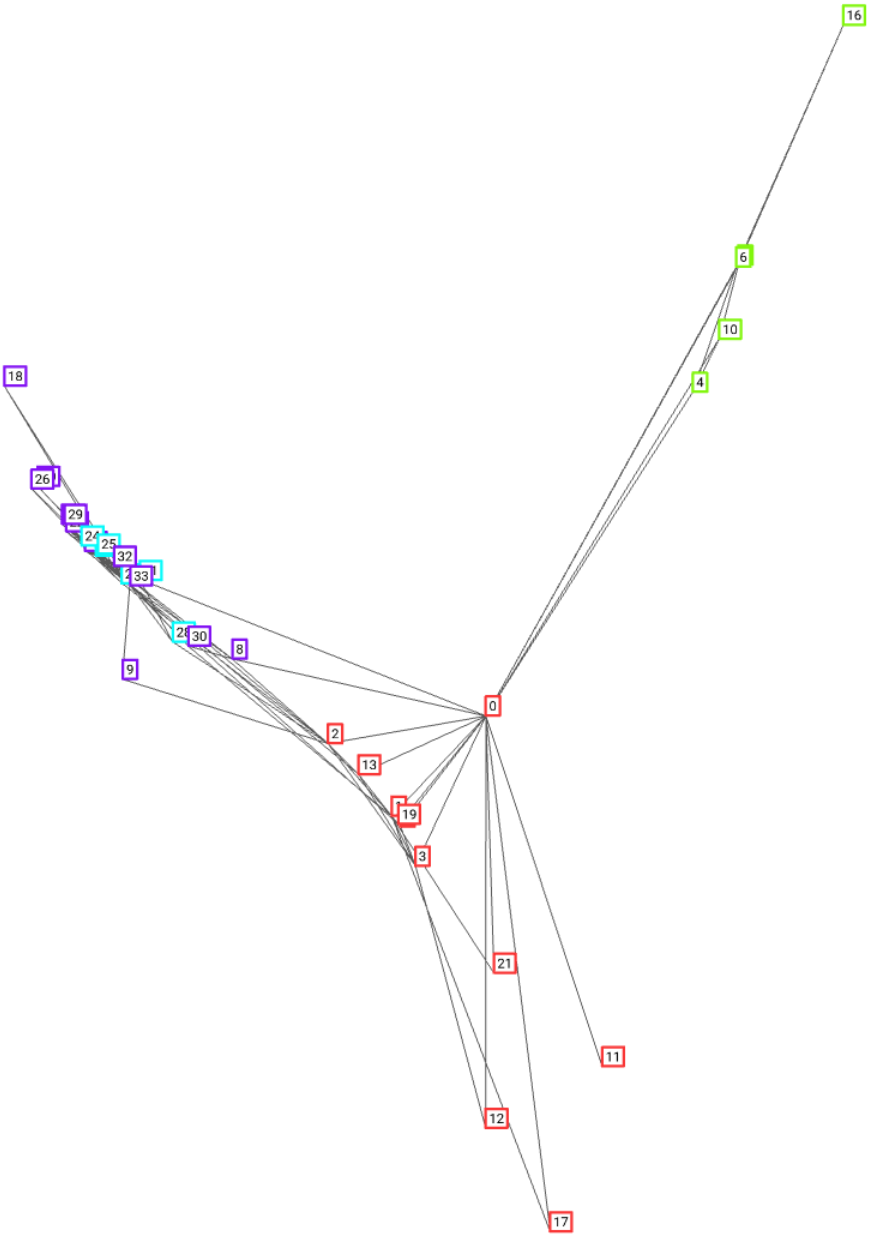
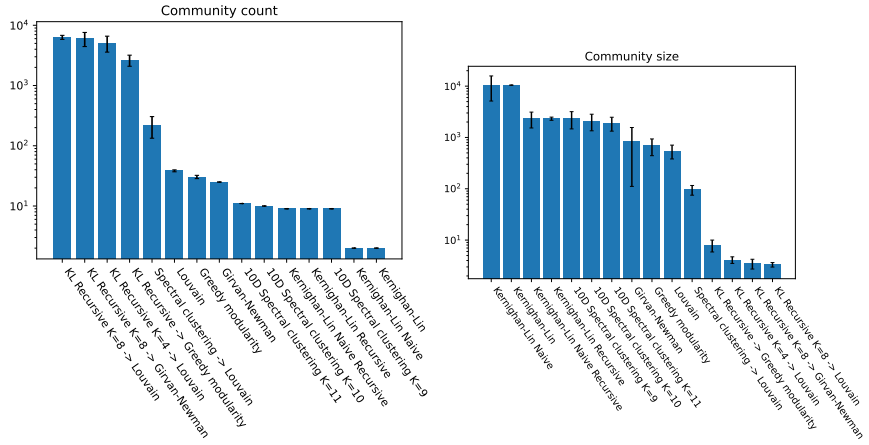
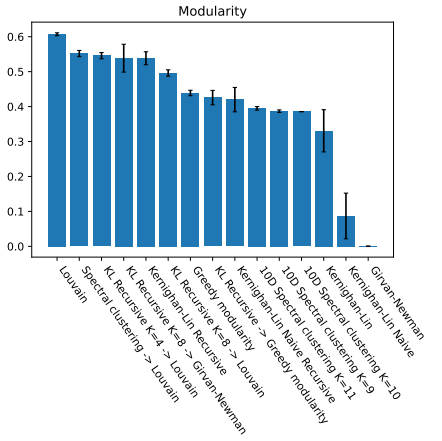


Figure 5.11 Zachary's karate club: Spectral encoding of Louvain algorithm using GraphVisualizer application

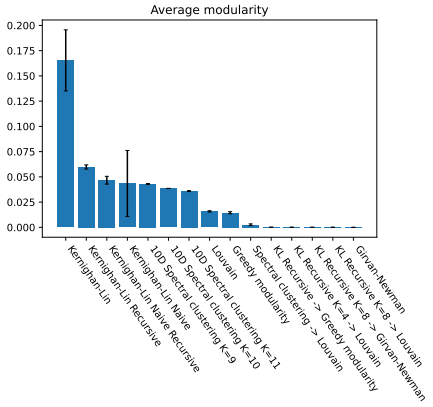


(a) Community count **(b)** Average community size

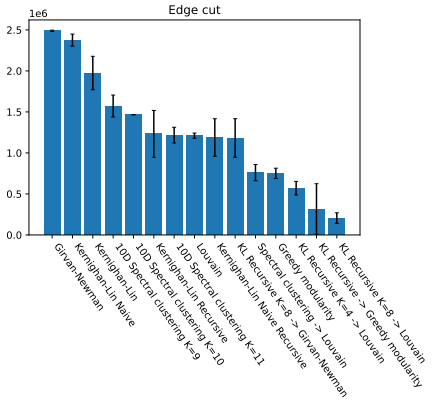
Figure 5.12 Enron: Basic community info.



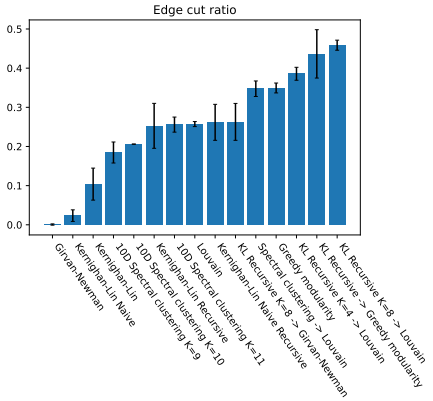
(a) Modularity



(b) Average community modularity

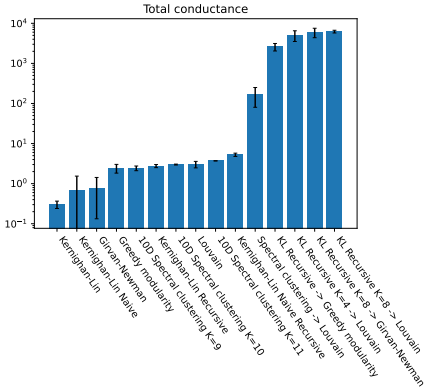


(c) Edge cut

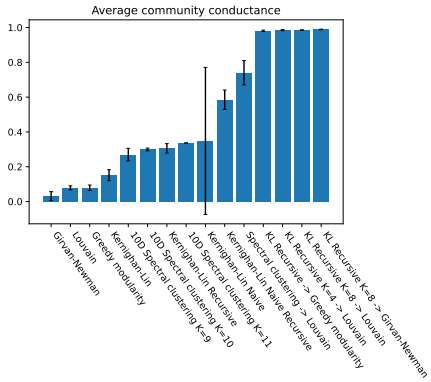


(d) Edge cut ratio

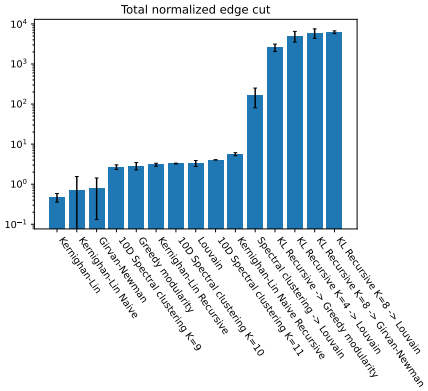
Figure 5.13 Enron: Modularity and edge cut.



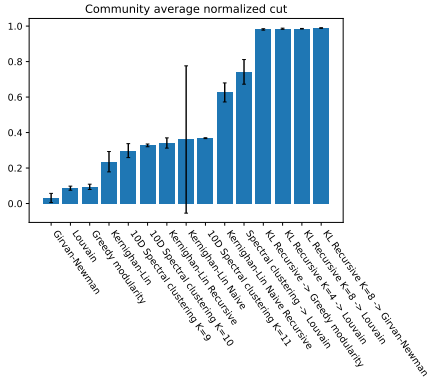
(a) Conductance



(b) Average community conductance

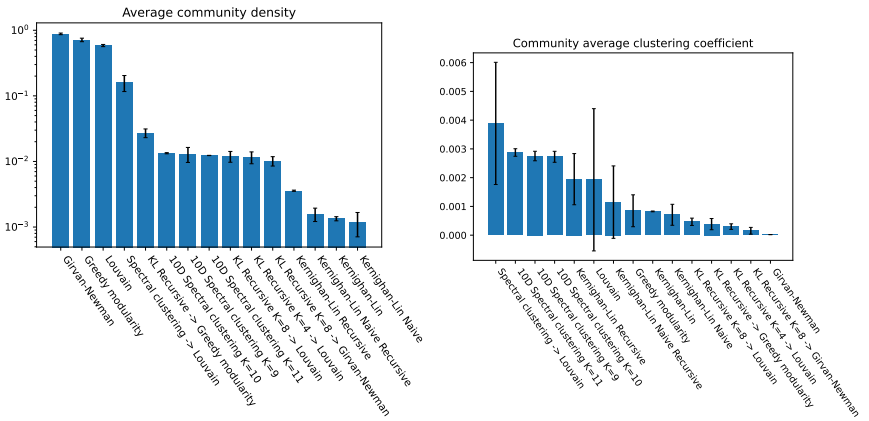


(c) Normalized edge cut



(d) Average normalized edge cut

Figure 5.14 Enron: Conductance and normalized edge cut



(a) Average community density

(b) Average community clustering coefficient

Figure 5.15 Enron: Density and clustering coefficient

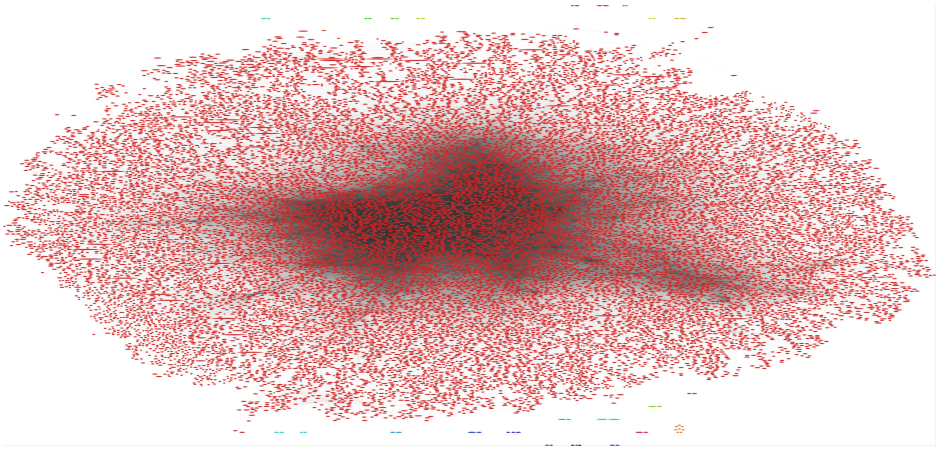


Figure 5.16 Enron: Girvan-Newman clustering

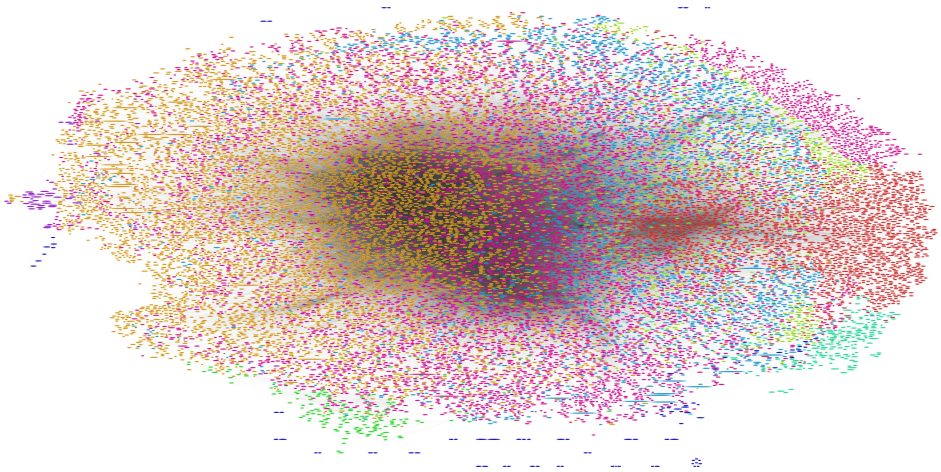


Figure 5.17 Enron: Spectral clustering visualization

Chapter 6

Implementation

As an integral part of the thesis, we have developed a community detection library and CLI application for automated community detection. The main requirements for the system are:

1. Running of community clustering algorithms. It should be open to extensions, stable in results and replicable even for algorithms containing randomness.

2. Export of graph visualization oriented to community detection.

3. Statistics calculation with results stored in the parsable format for future processing.

4. No programming is expected from the end user. The user should be able to specify input and computation instructions without writing a single line of code. A persistent solution is preferable, limiting command line arguments to a minimum.

5. Suitable for use in the server environment. Should be able to use the maximum of allocated resources and enable maximum parallelization for selected data.

In addition, we have built a graph visualization tool, enabling us to observe clustering behavior in small data.

The implementation repository is reachable by the public, and it is located at `https://gitlab.mff.cuni.cz/konigma/social-network-tools/`.

All scripts use `Python 3.10.4` and `networkx 2.8`. In addition, standard data analysis libraries such as `numpy`, `scipy`, `pandas` and `scikit-learn` are used.

6.1 Repository structure

The implementation consists of multiple folders representing different aspects of the repository. In the root of the repository, we find software documentation and `requirements.txt`, which is a file describing all Python dependencies. Dependencies can be installed by using `pip install -r requirements.txt` command. It is recommended to install dependencies using Python virtual environment.

Main application scripts are in the `src` folder. The main algorithm library is located at `src/algos` and exports all community detection algorithms. Folder `src/GraphVisualizer` contains interactive visualization application runnable with `python src/GraphVisualizer/app.py`. Respective CLI endpoints for community detection, persistent visualization, and metrics calculation as well as Jupyter notebooks performing result analysis are located in the root of `src` folder.

All datasets are located in `datasource` folder. This includes Zachary's karate club dataset and Enron dataset as well. There are both directed and undirected variants as well as their reduced variants with only 200 nodes. Configuration files are located in `configs` folder and are set up for evaluation of community detection algorithms defined in the thesis.

Graphs labeled with detected communities are located in the `results` folder. In addition, together with community detection results, we ship console logs processed later to perform algorithm run time evaluation. Visualizations are placed in `visualizations` folder, and calculated statistics are in `stats` folder. All folders contain `enron_undirected` and `karate_club` subfolders, dividing computed data along their source datasets.

6.2 Input and output format

One of the main decisions of the system itself was how the graph should be represented on the input. There are many different ways how to represent and store graphs starting from the simple textual list of edges.

We will compare two different approaches to storing graphs. Since we are using `networkx` library to represent graphs in the code, naturally, the first approach is to store graphs using Python binary serializer `pickle`. Using this method, we can store Python objects in persistent storage. The second approach consists of using an XML-based format named GraphML. Both approaches can store any graphs and attributes. Furthermore, GraphML is also a multiplatform human-readable format independent of the Python version.

There are multiple advantages to using binary serialized over text-based formats. The first one is the size of uncompressed data, the second is load speed.

In practice, however, we can mitigate the size of XML by using compression, where we can reduce graph size by about 90%.

We have chosen GraphML to be the default format of input and output of all graph data, since it is a standard format. In conclusion, using a native binary serializer provides no added benefits, as the speed difference is insignificant and binary format is not universally transferable.

Page `http://graphml.graphdrawing.org/` provides more information about GraphML format, including the formal definition of the standard.

Listing 1 Example of the simple graph with 2 nodes and 1 edge in valid GraphML format.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml>
    <key id="d0" for="edge" attr.name="weight"
        attr.type="long" />
    <graph edgedefault="undirected">
        <node id="0" />
        <node id="1" />
        <edge source="0" target="1">
            <data key="d0">1</data>
        </edge>
    </graph>
</graphml>
```

6.3 CGAT: Config-based Graph Analysis Tool

6.3.1 Configuration files

Since the main requirements for the system are ease of use and reproducibility, a custom computation description format was developed. Persistent configuration allows the storage of definitions for all experiments and all related calculations. Configuration file stored in JSON format can be described by abstract type `GraphRuntimeConfig` and subtype `SingleRunConfiguration` listed in Table 6.1 and Table 6.2. We can see the example of the JSON configuration file in Listing 2.

An algorithm run is a single instance of algorithm and respective algorithm parameters executed the number of times defined in the configuration file. This ensures the algorithm remains stable, and a rerun of the script does not change

Parameter	Type	Notes
in_file	string	path to GraphML file
out_folder	string	folder where results are stored
weight_key	string	node and edge attribute for weight
community_key	string	node attribute for community id
seed	int	seed for randomness
repetitions	int	number of repetitions for single run
runs	list[SingleRunConfiguration]	

Table 6.1 GraphRuntimeConfig type

Parameter	Type	Notes
algorithm	see below	path to GraphML file
metis	bool	folder where results are stored
optimal_node_count	int | None	METIS, compressed graph size
parallel	bool	enable parallel execution with other algorithms
dimension	int | None	parameter for Spectral clustering
max_iter	int | None	parameter for Kernighan-Lin
community_count	int	parameter for Girvan-Newman, Spectral clustering

Table 6.2 SingleRunConfiguration type. The parameter algorithm can be girvan_newman, greedy_modularity, louvain, kernighan_lin_naive, kernighan_lin_naive_recursive, kernighan_lin, kernighan_lin_recursive, spectral_clustering and nested X Y, where X and Y are algorithms from the list above.

clustering. It is important to note the user can decide whether to use parallel or serial execution of selected runs in the config. This is an important factor in parallel algorithms such as spectral clustering, where we can measure end-to-end performance using multiple CPU cores.

6.3.2 Instructions to run

Computation of the results

When input data and configuration file are both provided, the calculation can be then easily run with

```
python src/run_communities.py configs/config.json
```

from the repository root folder using provided valid JSON configuration file. The config file is read, and all respective calculations are performed. For this thesis, to run clustering for Zachary's karate club and Enron dataset, we use commands

```
python src/run_communities.py configs/karate_club_config.json
python src/run_communities.py configs/enron_config.json
```

Listing 2 Example of simple JSON configuration

```
{
   "in_file":"datasource/karate_club.graphml",
   "out_folder":"results/karate_club",
   "weight_key":"weight",
   "community_key":"community",
   "seed":42,
   "runs":[
      {
         "name":"karate_club_louvain",
         "algorithm":"louvain",
         "metis":false,
         "parallel":true
      },
      {
         "name":"karate_club_kernigan_lin",
         "algorithm":"kernighan_lin",
         "metis":false,
         "parallel":true,
         "max_iter":20
      }
   ]
}
```

When algorithm execution is finished, the graph is copied, and a community label is assigned to each node according to detected clustering. Resulting graphs from predefined configs relevant to this thesis are placed in `results/enron_undirected`, respectively `results/karate_club`. The standard output of the script is the runtime of the algorithm measured in seconds, which will be evaluated in algorithm analysis.

Running the visualization

When graphs have their respective community assignments computed, the system can be used to create persistent visualizations. Graph visualizations are saved in both SVG and JPG, as both formats have their valid use cases. Graph visualization can be run with simple command

```
python src/create_visualization.py \
  configs/config.json visualizations/dataset --dpi 100
```

Listing 3 Standard output from Zachary's karate club dataset.

```
python src/run_communities.py configs/enron_config.json
karate_club_kernigan_lin 0.0011 0.0020 0.0009
karate_club_louvain 0.0020 0.0023 0.0021
karate_club_louvain: 0.002267599105834961s
karate_club_louvain: 0.001999378204345703s
karate_club_kernigan_lin: 0.0020437240600585938s
karate_club_kernigan_lin: 0.0011334419250488281s
karate_club_kernigan_lin: 0.0009000301361083984s
karate_club_louvain: 0.0020751953125s
```

from the repository root folder. `--dpi` is a parameter used in JPG generation and marks pixel density in the resulting image.

Below we find standard commands for predefined config files. This will run the visualization for Zachary's karate club and Enron dataset, saving the visualizations into `visualizations` folder.

```
python src/create_visualization.py \
  configs/karate_club_config.json visualizations/karate_club \
  --dpi 100
python src/create_visualization.py \
  configs/enron_config.json visualizations/enron_undirected \
  --dpi 20
```

Similarly, the runtime is printed to standard output as with result calculation. Found communities are distinguished visually and are clustered together if applicable. Visualization, however, starts to struggle when either the graph becomes large, or there is a great amount of detected communities.

Running the statistics computation

An important part of the evaluation of community detection is the result evaluation. To evaluate results and calculate all important metrics, a standalone script has been added to the system. Command

```
python src/calculate_stats.py \
  configs/config.json stats/config
```

called from the repository root folder calculates the statistics. The config file is read, and all statistics are saved to the defined output folder. Statistics are saved as a JSON file with a predefined structure to enable further processing.

Below we can find commands for predefined config files. Commands will calculate all significant metrics and statistics for Zachary's karate club and Enron dataset, saving results as JSON files in `stats/karate_club`, respectively `stats/enron_undirected`.

```
python src/calculate_stats.py \
  configs/karate_club_config.json stats/karate_club
python src/calculate_stats.py \
  configs/enron_config.json stats/enron_undirected
```

We calculate results for each iteration of a single algorithm run as well as aggregated statistics for all iterations. For each aggregated metrics value, we calculate the mean, sample standard deviation, and 95% confidence interval.

6.3.3 Makefile

For easier usage on the server and Linux machines in general, we have created Makefile to run all connected commands. We can see all implemented commands in Table 6.3. Commands are supposed to be run in the order listed in the table. However, because the repository itself contains all the data and the results, only commands related to processing the data are relevant.

Command	Notes
make download_enron	downloads original Enron tar file
make create_karate_club	create karate club GraphMl file
make create_enron	process downloaded Enron into GraphML file
make create_reduced_enron	create Enron dataset with top 200 nodes
make create_datasets	create karate club and all Enron datasets
make calculate_enron_community_count	measure metrics for all community counts
make run_clustering	run clustering for karate club and Enron
make visualize	create visualizations for the datasets
make calc_stats	calculate statistics for karate club and Enron

Table 6.3 Makefile commands.

We can see that use of CLI commands `make run_clustering`, `make visualize` and `make calc_stats` is sufficient. Those commands will run all respective scripts for community detection, visualization, and statistics calculation for both karate club and Enron datasets, consuming provided configs and datasets. It also might be useful to pipe the standard output to a standalone text file with > to save the runtimes of individual scripts.

6.4 Community detection library

The main implementation consists of the community detection library and its use in standalone CLI applications. The library contains multiple Python functions performing specified clustering algorithms. In this section, we will provide programmer documentation for the community detection library since the application script was documented in detail in the previous section. The library is located in `src/algos` in the repository. For a start, it is important to mention all algorithm implementations return iterable structure over sets of nodes, which marks the final clustering for provided parameters.

6.4.1 Girvan-Newman

Function *girvan_newman_clustering(graph: Graph, community_count: int)* will perform Girvan-Newman clustering according to Section 2.3.3. The algorithm is based on iteratively removing the current highest centrality edge. During the algorithm computation, the connected component marks a single community, and graph disintegration is the basis for the creation of a community dendrogram. In this implementation, however, we do not return the whole dendrogram and instead, we return first clustering with at least *community_count* communities.

Algorithm accepts Graph or DiGraph instances and desired minimal community count and returns an iterable collection of node sets, each representing a community.

6.4.2 Greedy modularity

Function *greedy_modularity(graph: Graph | DiGraph, weight_key: str = 'weight')* performs community detection using simple modularity maximization as defined in Section 2.3.4. In simplicity, the greedy modularity algorithm works as follows:

1. In initialization, each node forms a community.

2. Merge 2 communities with the highest modularity gain until we have communities to merge.

3. Return graph partition with maximal modularity.

Implementation is plug & play as the only required parameters are a graph to perform clustering and weight attribute name. The algorithm returns the collection of node sets.

6.4.3 Heavy clique METIS

As introduced in Section 2.3.7, a heavy clique matching METIS compression algorithm is implemented in the function *heavy_clique_matching_metis(graph: Graph, optimal_node_count: int, clustering_algo: Callable[[Graph], Iterable[frozenset]], weight_key: str = 'weight')*. The algorithm works in 3 phases. In the first phase, the graph is compressed using heavy clique edge matching. In the second step, provided community detection algorithm is used on the compressed graph. In the last step graph is decompressed into the original input graph together with assigned communities.

Function besides input graph receives *optimal_node_count* as a parameter, marking the desired size of the compressed graph. The community detection algorithm is supplied as *clustering_algo* marking a function receiving graph and returning clustering and can be viewed as a curried version of another community detection algorithm. We need to supply also graph weight attribute marked as *weight_key* to calculate edge matching properly.

6.4.4 Kernighan-Lin Naive

kernighan_lin_naive_bipartition(graph: Graph, weight_key='weight', generator: numpy.random._generator.Generator, max_iter: int = 20) is a function applying naive Kernighan-Lin bipartition into the input graph. As defined in Section 2.3.1, Kernighan-Lin algorithm works by using selective switch of two adjacent nodes maximizing edge cut in the process.

The algorithm in the input receives graph, weight_key marking weight attribute, generator initialized for random number generation and max_iter marking the maximum number of epochs of the algorithm. As a result, a tuple of two sets of nodes is returned.

6.4.5 Kernighan-Lin

Function *kernighan_lin_bipartition(graph: Graph, weight_key='weight', generator: numpy.random._generator.Generator, max_iter: int = 20)* is the optimization of the naive Kernighan-Lin algorithm. The main improvement is we no longer consider adjacent nodes connected by an edge, but instead, we select 2 nodes with maximal modularity gain from opposing bipartitions. This results in huge performance improvement. This function is based on the standard implementation of Kernighan-Lin in `networkx` package.

This function has the same inputs and outputs as the naive variant.

6.4.6 Recursive bipartition

As defined in Section 2.3.2, we can extend the Kernighan-Lin algorithm to perform a general k-partition of the input graph. Function *recursive_bipartition(graph: Graph, community_count: int, bipartition_algo: Callable[[Graph], tuple[frozenset, frozenset]], weight_key: str = 'weight')* is designed to apply partitioning algorithm recursively.

Function in addition to standard Kernighan-Lin parameters accepts *bipartition_algo* parameter. It is a function accepting graph and returning bipartition of the such graph and in practice it is a curried version of Kernighan-Lin. This ensures the great extensibility of the recursive method.

6.4.7 Spectral clustering

Python function *spectral_clustering(graph: Graph | DiGraph, dimension: int = 2, clusters: int = 2, weight_key: str = 'weight')* performs spectral clustering on the input graph. As defined in Section 2.3.6 spectral clustering consists of two steps. In the first step, we calculate spectral embeddings, a mapping into k-dimensional space where densely connected nodes have minimal connecting distance. In the second step, we run K-Means to calculate real-valued clusters.

Besides the input graph, the algorithm on the input also receives a dimension of the embedding and number of clusters to detect (which maps directly to the number of communities on the output). List of communities represented as a set of nodes is returned when the algorithm finishes.

6.4.8 Louvain algorithm

State-of-the-art modularity optimization is reached using the Louvain algorithm as described in Section 2.3.5. The function *louvain(graph: Graph | DiGraph, weight_key: str)* implements the Louvain community detection algorithm. Louvain algorithm can be summarized in two steps. Initially, each node represents a community on its own. In the first step, for each node i and its neighbors j, we calculate modularity gain by moving the node from the community of node i to the community of node j. We perform the move with maximal gain under the condition the gain is positive. In the second step, we concatenate each community into a single node and move to the first step. The algorithm terminates when there are no positive modularity gains possible.

Similarly, as with greedy modularity, the function accepts the input graph and *weight_key* parameter specifying the name of the weight attribute. It is easy to see the algorithm is considered plug & play since no other hyperparameters are required to detect communities in the input graph.

6.5 GraphVisualizer UI application

For research purposes, we also built a social network visualization application, GraphVisualizer. This application is needed to gain more insight into Zachary's karate club clustering and to visualize connections between spectral embedding and other clustering methods. Application is located in `src/GraphVisualizer` and can be launched using
`python src/GraphVisualizer/app.py`. Visualization is performed on provided graphs with `community` label provided on each node. If no label is provided on some graph nodes, the graph is considered to be unclustered.

Visualization application is built using Python and `kivy` framework. It is an UI application with an infinite scrolling canvas on the right-hand side of the screen dedicated to showing information about the currently selected node. If the graph contains valid clustering, each node has a color according to the assigned community. Supported layouts include spring layout, community layout, and spectral layout. The latter is important to understand the reasoning behind spectral clustering. The application also supports performing simple community detection using the Louvain community detection algorithm.

Visualizations are supported for small graphs only. Even though the UI framework is heavily optimized and operates on OpenGL graphic layer directly, many rendered elements significantly impact performance.

6.5.1 Color generation

The interesting part about the application was creating a method for generating a set of distinguishable colors for different communities. This is solved by using **H**ue **L**ightness **S**aturation model. Hue is defined as a radial basis, ranging from 0 to 360, where lightness and saturation range from 0 to 1.

To generate n colors, we split the circle radius into n uniformly spaced angles, which form the hue value. Lightness and saturation are picked randomly. Lightness is picked uniformly from interval $[0,5; 0.6]$ and saturation from interval $[0,9; 1.0]$. That way, we can generate n different colors covering the whole color spectrum.

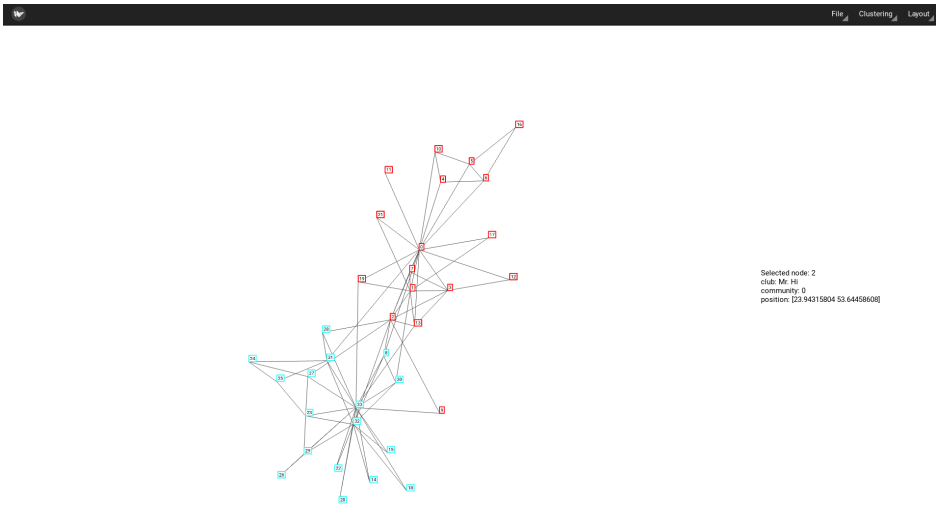


Figure 6.1 Zachary's karate club: Kernighan-Lin communities

Conclusion

In this thesis, we have provided a deep dive into social network science and its part devoted to community detection. The thesis covers the definition of social networks and their fundamental properties (Chapter 1) and specifies the notion of communities and the methods for their detection (Chapter 2). Datasets that are used for evaluation, namely Zachary's karate club and Enron dataset, are discussed in Chapter 3. The work outlines all related experiments (Chapter 4) and analyzes their results (Chapter 5).

Within the framework of the research, we implemented a community detection system, which is able to detect communities automatically, visualize social networks and evaluate the performance of the selected community detection algorithms. In addition, this system can be configured with persistent JSON files and permanently save the obtained results. The system is extendable and is perfectly suitable for any future research.

As a part of the experiments, we proposed and evaluated a novel approach to community detection. The recursive extension of the Kernighan-Lin bipartition algorithm has proven worthy, where a combination of a reasonable tree expansion with a well-defined objective function significantly improved the results. Secondly, we introduced and evaluated nested method to community detection that creates gradually finer partitions using two different clustering algorithms. However, this technique provided mixed-quality results at best.

In further experiments, we estimated the actual number of communities in the Enron dataset. A large amount of time and computing power is necessary to solve this task correctly. We discovered that two out of three criteria delivered no promising value. However, inertia analysis provided all the required information to proceed to further experiments.

Secondly, we investigated the behavior of a non-hierarchical algorithm named spectral clustering and its ability to form a sub-community hierarchy. This was observed by progressively increasing the community count parameter. In theory, even a slight change in the number of clusters can significantly change the clustering itself. According to this experiment, the spectral clustering method resembles other hierarchical approaches that create new communities by merging

base subparts, building more predictability. This suggests we can analyze a community hierarchy even with the non-hierarchical algorithm.

Name	Speed	Modularity	Edge cut	Plug & play
Spectral clustering	++	+	++	-
Louvain	++	++	+	+
Kernighan-Lin	++	+	++	+
Kernighan-Lin Recursive	+	+	+	-
Greedy modularity	-	+	-	+
Girvan-Newman	-	–	++	-
Kernighan-Lin Naive	–	–	++	+
Kernighan-Lin Naive Recursive	–	+	+	-
Nested algorithms	+	+	-	–

Table 6.4 Algorithm recommendation summary. The table summarizes key properties of the community detection algorithms and highlights the most usable algorithms. Plug & play marks algorithms where no other parameters except input data are required.

Finally, we performed community detection on two differently-sized datasets. We tested and evaluated algorithms listed in Table 6.4. The table contains four main criteria for good community detection, and each algorithm is assigned a different score. Plug & play means the algorithm is ready to be used without any other parameters but the input graph.

As expected, the differences in the algorithm performance began to emerge as we scaled up the data. The best algorithms for the known number of communities are spectral clustering and recursive Kernighan-Lin. These algorithms provided superior results at a rapid speed.

In cases where the number of communities is unknown beforehand, the Louvain algorithm represents the best choice due to automatically deciding on the appropriate clustering. For simple bipartitions, Kernighan-Lin performed well and provided reasonable results. For any other algorithm, METIS compression is needed to reduce the runtime to a runnable limit. As we discovered, the METIS compression works perfectly, providing only a small loss to the processed data. We show the Girvan-Newman algorithm behaves oddly when the input graph is not a single connected component, and in such case, other algorithms shall be preferred.

Selected algorithms provide superior results for small and mid-sized datasets. As mentioned earlier, datasets of great size were not evaluated to promote fairness. However, results suggest the top-performing algorithms should also stand out in big data.

Further research

While we observed remarkable results in community detection, there are more areas for improvement and further research. Firstly, as in machine learning, estimating the number of clusters is a time-demanding task. This problem is currently solved via grid search, where we test a set of predefined parameters to estimate the number of clusters. This can be improved further upon by defining a method for automated computation of the number of communities using fewer computational resources.

Secondly, while our suggestion of nested algorithms did not provide sufficient results, we believe there are more possible improvements to the model. We observed the method is sensitive to the choice of the roughness of the first algorithm, and if chosen wrong, the algorithm returns only a great number of extremely small communities. We believe this can be improved by developing an online method for roughness estimation, which connects the first and second step. This might be based on adjusting the first algorithm parameters based on the overall results. In that particular case, there is only a relatively small overhead since there are not that many parameters to evaluate for the first algorithm. However, other approaches would be greatly admittable, discarding grid search altogether.

Bibliography

[1] Charu C Aggarwal et al. *Data mining: the textbook*. Vol. 1. Springer, 2015. Chap. 19.

[2] Sergey Brin and Lawrence Page. "The anatomy of a large-scale hypertextual web search engine". In: *Computer networks and ISDN systems* 30.1-7 (1998), pp. 107–117.

[3] Srinivasan Parthasarathy, Yiye Ruan, and Venu Satuluri. "Community discovery in social networks: Applications, methods and emerging trends". In: *Social network data analytics* (2011), pp. 79–113.

[4] Albert-László Barabási and Márton Pósfai. *Network science*. Cambridge: Cambridge University Press, 2016. ISBN: 9781107076266 1107076269. URL: http://barabasi.com/networksciencebook/.

[5] B. W. Kernighan and S. Lin. "An efficient heuristic procedure for partitioning graphs". In: *The Bell System Technical Journal* 49.2 (1970), pp. 291–307. DOI: 10.1002/j.1538-7305.1970.tb01770.x.

[6] Mark EJ Newman and Michelle Girvan. "Finding and evaluating community structure in networks". In: *Physical review E* 69.2 (2004), p. 026113.

[7] Mark EJ Newman. "Fast algorithm for detecting community structure in networks". In: *Physical review E* 69.6 (2004), p. 066133.

[8] Vincent D Blondel et al. "Fast unfolding of communities in large networks". In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10 (2008), P10008. DOI: 10.1088/1742-5468/2008/10/p10008. URL: https://doi.org/10.1088%2F1742-5468%2F2008%2F10%2Fp10008.

[9] Richard B Lehoucq, Danny C Sorensen, and Chao Yang. *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*. SIAM, 1998.

[10] Wayne W Zachary. "An information flow model for conflict and fission in small groups". In: *Journal of anthropological research* 33.4 (1977), pp. 452–473.

[11] Bryan Klimt and Yiming Yang. “Introducing the Enron Corpus”. In: *CEAS 2004 - First Conference on Email and Anti-Spam, July 30-31, 2004, Mountain View, California, USA*. 2004. URL: http://www.ceas.cc/papers-2004/168.pdf.

[12] M. Baer. *findiff Software Package*. https://github.com/maroba/findiff. 2018. URL: https://github.com/maroba/findiff.

List of Figures

List of Tables