Utrecht University

Master's Thesis

Community Detection in Historical Data Using Knowledge Graphs

Daily Supervisor:
J. (Jurian) Baas MSc
j.baas@uu.nl

Author:
Iman Hashemi (*5707617*)
iman_hashemi@live.nl
ICA-5707617

Primary Supervisor:
Dr. A.J. (Ad) Feelders
a.j.feelders@uu.nl

Second Supervisor:
Prof. dr. M.M. (Mehdi) Dastani
m.m.dastani@uu.nl

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science in Computing Science

Department of Information and Computing Science
May 30, 2022

Abstract

In the field of Digital Humanities, knowledge graphs are used to store and model archival data. For instance, the ECARTICO dataset describes actors involved in the cultural industries of the Low Countries and the STCN dataset describes published books and their authors and printers. This data opens up the possibility of performing community detection on parts of these (combined) datasets. For instance, a combination of parts of the STCN and ECARTICO knowledge graphs could reveal networks of people who worked together through shared acquaintances, such as printers and publishers. The goal of this research project is to performs static and dynamic community detection on these (combined) datasets, in order to find interesting clusters and follow their evolution through time. To do this, community detection algorithms designed for Heterogeneous Information Networks are analysed, the historical data is converted and the algorithms are applied to the data. Internal evaluation measures indicate that our method finds structure in the data and that, according to domain experts' evaluation, the results are valid.

Contents

Acknowledgements

I would like to express my gratitude to my daily supervisor, Jurian Baas, and my primary supervisor, Ad Feelders, who have helped and guided me throughout this research project. I would also like to thank the Golden Agents project and the Processes of Creativity group for the the insightful input and productive discussions we had, which helped form this research. Special thanks to Leon van Wissen, Harm Nijboer and Judith Brouwer, who helped with the evaluation and were available for any questions I had.

Chapter 1

Introduction

To understand state building, we have argued, one needs to penetrate beneath the veneer of formal institutions and apparently clear goals, down to the relational substratum of people's actual lives. Studying "social embeddedness," we claim, means not the denial of agency, or even groups, but rather an appreciation for the localized, ambiguous, and contradictory character of these lives. Heterogeneity of localized actions, networks, and identities explains both why aggregation is predictable only in hindsight and how political power is born.

John F. Padgett
Christopher K. Ansell
Robust Action and the Rise of the Medici, 1400-1434

As social networks have emerged and grown substantially in the past few decades, think of networks such as the *Facebook* network, the need for methods that extract meaningful information from these networks has been growing. One of these methods is community detection, which is the pursuit of finding clusters of nodes in networks. In the field of Digital Humanities, this emerging research field is used to analyse digitized historical data. In [Padgett and Ansell, 1993], the rise of the Medici family was studied by analysing clusters and applying blockmodeling techniques [Lee and Wilkinson, 2019]. In this research, the authors used these techniques to understand the ascent of the political dynasty under Cosimo de' Medici, with which they found that the network around Cosimo and the heterogeneity of it played an important role in this process.

The Golden Agents project studies the rise of the creative industries in the Dutch golden age, by "*developing a sustainable research infrastructure to study relations and interactions between producers and consumers of creative goods across the long Golden Age of the Dutch Republic (ca. 1580 – ca. 1750)*" [Gol, 2021]. Processes of Creativity, a case study of the project, focuses on measuring innovation in book culture in the Dutch golden ages, based on an observation of a 'rise' of the illustrated book in the second half of the 17th century and the beginning of the 18th century. The goal of this research is to assist the Processes of Creativity case study in this pursuit, by analysing the available data concerning this industry in this time period and applying community detection to the data.

The data that is to be analysed is archival data stored as Knowledge Graphs (KGs). More precisely, it is a conglomeration of multiple KGs, namely the following:

- **Short Title Catalogue of the Netherlands (STCN)**: The STCN contains bibliographic information on books produced in the Netherlands up to the year 1800.
- **Rijksmuseum Library (RM)**: The Rijksmuseum Library contains similar information as the STCN, but it contains more information about the entities that were involved in the creation of a book.
- **ECARTICO**: ECARTICO also contains bibliographic data on many kinds of actors involved in the cultural industries in the Netherlands, such as engravers and book sellers.

There are links between these KGs, which means that they can be combined to perform a more interesting analysis. For example, The Rijksmuseum Library contains extra information about the authors of a certain book, information that may not be present in STCN. As some books in the Rijksmuseum Library can be also be found in the STCN dataset, and these can be identified through their Universal Resource Identifiers (URIs), this extra information can be queried and added to a new KG. The KG is then a combination of the STCN and Rijksmuseum Library datasets, containing all the books in the Rijksmuseum Library that are also present in the STCN KG, as well as the extra information the Rijksmuseum Library has to offer on the actors involved in the creation of these books.

Finding communities in this data can produce relevant information for domain experts, such as historians. For example, it may be interesting to find communities of authors that worked together closely. One could also find communities of authors based on shared acquaintances, such as publishers. This could produce insights that are not immediately obvious from a cursory look at the data. Furthermore, it might also be beneficial to analyse these communities. For instance, one can question why a group of authors is part of a community. Is it based on the genre of the books they authored? Or is it mainly based on the time period they worked in? Finally, one could track the evolution of these communities through time, which may help the analysis of developments in the industry.

1.1 Research Questions

The main goal of this research project is to develop a method of applying community detection to the the historical data, which is represented by a collection of KGs. The communities that are found have to be useful to domain experts, mainly historians in this case, in analysing the data and finding new interesting patterns in it. Therefore, we formulate the following research question:

Research Question 1. Is it possible to find interesting clusters in historical data by performing community detection on KGs?

To address this issue, we must make choices on how to represent the data in order to apply community detection to it, and determine what algorithms perform best on the data.

Furthermore, we are interested in the evolution of communities through time, due to the important temporal aspect of the data. Thus, the second question this research attempts to answer is:

Research Question 2. Is it possible to track the evolution of communities in the network through time?

These questions quickly produce the following questions: what is an 'interesting' cluster? How can this be validated? Validating results is a well-known issue in the field of unsupervised learning. Of course, certain measures, like modularity (see 3.2.1), can be used to indicate that the data does contain different clusters. This, however, is not enough. Validation using domain experts' opinion is very much needed in this case. More specifics about the validation of the results are given in chapter 5, but in the pursuit of helping domain experts as much as possible with their evaluation, the following question arises:

Research Question 3. Is it possible to find descriptions of clusters based on node attributes to support the interpretation and validation of the clusters?

A description could say, for example, that most of the people in a community worked in Amsterdam, or that there is a certain common religion in a community.

Chapter 2

Preliminaries

2.1 Knowledge Graphs

A Knowledge Graph (KG) is, in its core, a graph based model used to represent information. Although the term *knowledge graph* was first coined in [Schneider, 1973], one of the first widely used KGs was WordNet [Miller, 1995], which is an open source topic-specific KG that represents the semantic relationships between words. Google's Knowledge Graph [Goo, 2012], which was announced in 2012, was one of the first widely used commercial KGs. The Google KG was and is still used to provide users with structured information about their queries.

There is no real consensus on a definition of KGs. In [Hogan et al., 2021], KGs are defined as follows: "*a graph of data intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent potentially different relations between these entities.*". In the Recource Description Framework [RDF, 2014], knowledge is represented using triples in the form (*subject*, *predicate*, *object*). These triples are also sometimes denoted as *facts*. A set of these triples, linked together, can be seen as a KG. KGs can more formally be defined as follows:

Definition 2.1.1 (KG)**.** A knowledge graph is a multi-relational graph $G = \{\mathcal{E}, \mathcal{R}\}$, where $\mathcal{E}$ is a set of entities, regarded as nodes, and $\mathcal{R}$ is a set of relations, regarded as directed edges.

2.2 Heterogeneous Information Networks

A Heterogeneous Information Network (HIN) is a form of network that is structurally very similar to a KG. HINs have a clear and concise definition and there is much research on data mining in HINs, including on the problem of community detection. Because of this, this research will use the HIN data structure as a basis, as it is possible to convert a KG such that it adheres to the definition of a HIN. HINs, first introduced in [Sun et al., 2009], are networks that consist of multiple types of nodes and multiple types of edges, more formally defined as follows:

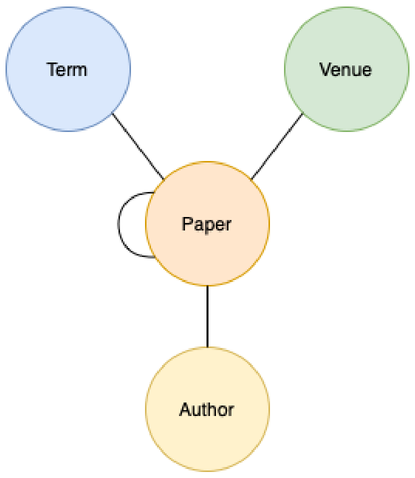


Figure 2.1: A simple network schema.

Definition 2.2.1 (Heterogeneous Information Network)**.** A Heterogeneous Information Network is a directed graph $G = (V, E)$ for which there exist two mapping functions $\varphi : V \to \mathcal{A}$ and $\psi : E \to \mathcal{R}$, which map nodes to object types and edges to link types respectively. $\mathcal{A}$ is the set of object types and $\mathcal{R}$ is the set of link types, where $|\mathcal{A}| > 1$ and $|\mathcal{R}| > 1$. In other words, there are multiple types of nodes and multiple types of edges. Each node $v \in V$ belongs to one object type and each edge $e \in E$ belongs to one link type. Furthermore, any link type $R \in \mathcal{R}$ has one starting object type and one ending object type.

In addition to that, for any relation R between starting object type S and ending object type T, the inverse of R, R^{-1}, holds for starting object type T and ending object type S. R is equal to R^{-1} if S and T are equal, or in other words, if R is symmetric. A HIN can now be described by a network schema, which is a meta template of the network.

Definition 2.2.2 (Network Schema)**.** A network schema, $T_G = (\mathcal{A},\ \mathcal{R}_u)$, is an undirected graph where the nodes are object types from $\mathcal{A}$ and the edges represent the combined relationship $\mathcal{R} \cup \mathcal{R}^{-1}$.

A network schema can quickly give an idea of the structure of the network. In figure 2.1, a simple network schema is shown for a network with object types *Term, Venue, Paper* and *Author*. The link types are *mentioned/mentionedBy* by between *Term* and *Paper*, *write/writtenBy* between *Paper* and *Author*, *publish/publishedBy* between *Venue* and *Paper* and *cite/citedBy* between *Paper* and *Paper*.

According to the network schema, you can draw paths between two object types. Such a path is called a meta-path.

Definition 2.2.3 (Meta-path)**.** A meta-path P is a sequence of relations $R \in \mathcal{R}$ between objects $A \in \mathcal{A}$.

In this case, a possible meta-path could be: $Term \xrightarrow{mentionedBy} Paper \xrightarrow{writtenBy} Author$. If there exists only one relation type between any pair of objects in the HIN, it suffices to only write the object types in a meta path. The same meta-path can then be written as: $Term \to Paper \to Author$.

2.3 Dynamic Graphs

As we are working with historical data, there is a temporal dimension in the data. Dynamic graphs are used to model graphs with temporal dimensions. In [Michail, 2016], dynamic graphs are informally described as "graphs that change over time". A dynamic graph can be modelled as a temporal graph, which is defined as follows:

Definition 2.3.1 (Temporal Graph)**.** A temporal graph is a graph $G = (V, E, T)$, where V is a set of triplets in the form (v, t_s, t_e), where v is a vertex and t_s and t_e are the birth and death timestamps of the corresponding vertex.

From this definition, it follows that any edge $e \in E$ also has a birth and death timestamp, namely the latest birth timestamp and the earliest death timestamp of the two nodes it connects. A dynamic graph can also be modeled as a snapshot graph (SG), which is defined as follows in [Rossetti and Cazabet, 2018]:

Definition 2.3.2 (Snapshot Graph)**.** A snapshot graph G_τ is an ordered set $<G_1, G_2...G_t>$, where each snapshot $G_i = (V_i, E_i)$ is identified by the sets of nodes V_i and edges E_i .

2.4 Communities

As our aim is to find communities in the networks, we must define the concept of a 'community'. In [Barabási, 2016], a community is defined as follows:

Definition 2.4.1 (Community)**.** A community is a locally dense connected subgraph in a network.

This is a very general definition. It states that a community is a component that is connected, i.e, every node in the community can be reached from every other node in the community. Furthermore, it states that a community is locally dense, meaning that nodes have a higher probability of connecting to a node within the same community, as opposed to connecting to a node from a different community. There are more strict definitions of communities, a very strict one being the definition that was posed in [Luce and Perry, 1949]. Here, a community was defined as a *clique*, meaning that there must be an edge between every pair of nodes of the community. As one can imagine, this definition may be too stringent.

When performing community detection in dynamic graphs, the goal is to find *dynamic* communities. Dynamic communities are communities that can change over time, they can grow, shrink, split and more. A *static* community can be represented simply by a list of nodes that are part of the community. A dynamic community can then be represented by a sequence of 'communities', where each element of the sequence simply records the state of the dynamic community at a certain time step.

Chapter 3

Community Detection Methods

Community detection can be performed in various ways, using various network types. There are methods that perform community detection directly on HINs. HINs can also be converted to homogeneous graphs (HGs), for which there are a plethora of community detection methods available. Furthermore, there are a multitude of algorithms for finding dynamic communities. In this section, an overview of these methods will be given.

3.1 Community Detection on HINs

Many solutions have been proposed to solve the problem of community detection in HINs, varying from methods using generative statistical models to methods using spectral clustering.

3.1.1 PathSelClus

In [Sun et al., 2013], clustering in HINs is done in a semi-supervised manner, using a generative probability model. A distinction is made between target types and feature types, where a target type is the type of the nodes that should be clustered and the feature type is the type of the node at the end of a meta-path beginning from the target type. The user provides the target type T, the number of clusters K and an object seed $\mathcal{L}_k$ for each cluster, which can contain zero or more objects per seed. This object seed will be used as user guidance. Finally, the user can also provide a set of M meta-paths (starting from the target type T). The algorithm can also select meta-paths by itself, in case the user does not provide any.

For each meta-path, an adjacency matrix is calculated by multiplying the adjacency matrices of all neighbors in the meta-path. This means that for meta-path $P_m = APA$, the adjacency matrix is calculated as follows: $W_m = W_{AP} \times W_{PA}$, where W_{AP} denotes the adjacency matrix for all nodes of type A and P. A set of weights for the meta-paths is initialized, which will also be part of the output of algorithm, along with the clustering assignment vectors. Now, a generative probabilistic model is created based on the observed relationships in the HIN, the user guidance (the object seeds for each cluster) and the meta-path weights. From this, an objective function is formed, which is then optimized in a two-step iterative process. In this process, first, the clustering results are optimized, while fixing the meta-path weights, using an EM-algorithm. In the second step, the meta-path weights are optimized, while fixing the clustering result, using gradient descent. The output of the algorithm is a matrix of clustering results and a set of meta-path weights. These meta-path weights can be interesting, as they can provide information on what the importance of certain meta-paths is for the results.

3.1.2 Multiple clusterings

In [Wei et al., 2021], a different approach is taken. Here, multiple distinct clusterings are generated based on multiple meta-paths and multiple values for the number of clusters. The method employs a skip-gram like method for generating node embeddings and clustering assignment vectors, simultaneously. The user has to provide a set of meta-paths, a set of values for the number of clusters, and some additional hyper-parameters. The algorithm optimizes an objective function that consists of three parts, namely a node embedding vector for all nodes that are present in the meta-path instances, a normalization function optimizing the diversity of different embeddings for the different meta-paths and a soft clustering assignment part that minimizes the cross-entropy of the assignment vectors for the nodes.

The node embedding is done using an approach similar to node2vec [Grover and Leskovec, 2016], called metapath2vec [Dong et al., 2017]. Here, a random walk approach is used to construct the neighbourhood of a node, and using negative sampling, the probability of that neighbourhood given the node and the parameters of the model is maximized. Negative sampling is used to improve performance. Instead of using all other nodes to maximize the probability of the neighbour nodes given the target node, negative sampling randomly samples a set of nodes for this computation.

3.1.3 Spectral clustering

In [Li et al., 2019], an approach based on spectral clustering is proposed and the SClump algorithm is introduced. A similarity matrix is constructed using the PathSim similarity measure introduced in [Sun et al., 2011], after which spectral clustering is applied to this similarity matrix.

3.1.4 Joint graph embedding and nonnegative matrix factorization

In [Zhang et al., 2021], Non-Negative Matrix Factorization (NMF) is used to perform clustering, using the GEjNMF algorithm. First, the HIN is divided into a set of bipartite networks, of which one type is the center type, which connects to other types in the network. It is assumed that there is always a center type present to do this. These bipartite networks are then embedded and NMF is performed on them, obtaining a result for each bipartite network. These results are then combined to obtain a clustering for the full HIN.

3.1.5 Attributed HINs

In [Li et al., 2017], a slightly adjusted format of a HIN is used, namely an Attributed Heterogeneous Information Network (AHIN). In AHINs, every object has an attribute vector. The SCHAIN algorithm is introduced, which uses these attribute vectors as a measure of similarity between two objects. More precisely, SCHAIN composes a similarity matrix S, which is derived from attribute vector similarity and meta-path based connections in the graph. As the algorithm is semi-supervised, a loss function is composed that takes into account the user input and optimizes clustering assignments, along with weights for object attributes and meta-paths.

3.1.6 Transforming to homogeneous graphs

In [Liu et al., 2020], the NodeVecClus algorithm is introduced. First, this method creates homogeneous graphs using meta-paths. Given a meta-path, where the feature node and the target node have the same type, a homogeneous graph can be created by creating an edge for all instances of the meta-path in the HIN. All nodes in this homogeneous graph will have the target type and every edge will represent a meta-path instance in the initial HIN between the nodes the edge connects. Given a set of meta-paths, NodeVecClus first creates a homogeneous graph for each meta-path. Then, for each graph, node2vec is used to create embedding vectors of the target nodes [Grover and Leskovec, 2016]. A weight vector for each meta-path is initialized and using a loss function based on the probability of observing structural neighbours given a meta-path, the node embedding matrix and the weight vector are obtained. These results are then used to perform spectral clustering.

In [Liu and Li, 2021], a similar approach is taken. The proposed algorithm, HINLP-Clus, first creates a homogeneous graph for each meta-path. More practically, it creates an adjacency matrix for each meta-path, in the same way that is done by the PathSel-Clus algorithm described in section 3.1.1. Then, a weighted adjacency matrix is created for the set of meta-paths by summing all adjacency matrices. In this manner, nodes that are connected to each other through multiple meta-paths will have a larger edge weight. Then, using this adjacency matrix for the weighted homogeneous graph, label propagation is performed. Label propagation, a method introduced in [Raghavan et al., 2007], is a community detection method for homogeneous graphs. First, it assigns every node its own label. Then, each node broadcasts its label to its neighbours. Then, each node chooses the label that is communicated to it most frequently. This process is then iterated until the cluster labels in the network do not change anymore. In this case, the graph is weighted, so the procedure is adjusted slightly to calculate the sum of the edge weights for each label communicated to a node. The edge weight that is used is the weight value of the edge through which a label is propagated. The label with the highest total value is chosen for the next iteration.

3.2 Community Detection on Homogeneous Graphs

The algorithms discussed above can be roughly divided into two categories: methods that perform community detection on the HIN data structure and methods that convert the HIN to one or multiple homogeneous graphs, after which they perform a (possibly slightly adjusted) community detection algorithm designed for homogeneous graphs. In [Liu et al., 2020] and [Liu and Li, 2021], experiments were performed comparing the NodeVecClus and HINLPClus algorithms to the SCHAIN and PathSelClus algorithms. NodeVecClus and HINLPClus both first transform the HIN to one or multiple homogeneous graphs before performing community detection. In these experiments, the Rand Index (RI) and the accuracy of the results were compared. In order to calculate these measures, a ground truth was inferred from the data. The RI is a measure that models the similarity between two clusterings. Both NodeVecClus and HINLPClus seem to outperform SCHAIN significantly. They also both performed comparatively or, in some cases, better than PathSelClus in the experiments. The runtime of the algorithms was also compared, from which it was apparent that both NodeVecClus and HINLPClus perform much better in terms of runtime.

Certainly, more experiments need to be done to be able to draw conclusions. However, the fact that these experiments did show promising results for these, comparatively simple, algorithms, is interesting. Converting a HIN to a homogeneous graph makes it possible to use existing community detection algorithms that work on homogeneous graphs. There are many existing and diverse possibilities here, including hierarchical clustering and overlapping community detection. Now, three different forms of community detection on homogeneous graphs will be explored: modularity-based community detection, overlapping community detection and hierarchical community detection

3.2.1 Modularity-based community detection

Modularity has been widely used as an optimisation measure for finding community structure [Girvan and Newman, 2002]. The modularity measure is based on the hypothesis that a randomly connected graph does not have an inherent community structure. This means that the distribution of the density of edges in a randomly connected graph is expected to be uniform, while this distribution in a graph with inherent community structure is expected to contain fluctuations where communities are present. Given a network G and a set of communities C, the modularity of a community C_c in G is defined as follows:

$$M_c = \frac{1}{2L} \sum_{(i,j) \in C_c} (A_{ij} - p_{ij}) \tag{3.1}$$

Here, L is the number of edges in G, A is the adjacency matrix for G and p_{ij} denotes the expected number edges between i and j, if they were randomly connected. It is important that the degree sequence, the degrees of all nodes in the network, is preserved when calculating p. The degree of a node is the number of edges the node has. The probability that node i connects to node j, given their respective degrees k_i and k_j, is determined as follows:

$$p_{ij} = \frac{k_i k_j}{2L} \tag{3.2}$$

Now, the modularity for the whole network and a given set of communities can be defined as follows, where L_c is the number of edges in community c, n_c is the total number of communities and k_c is the sum of the degrees of all nodes in community c:

$$M = \sum_{c=1}^{n_c} \left[\frac{L_c}{L} - \left(\frac{k_c}{2L} \right)^2 \right] \tag{3.3}$$

This value represents the difference of the fraction of intra-community edges and inter-community edges of a community partition with the expected number of such edges if they were distributed randomly. A higher M means that there is a stronger community structure present in the network. If $M = 0$, the network edges are randomly connected and, according to the previously discussed hypothesis on the community structure of a randomly connected graph, there is no community structure. M can also be negative. This happens in the case that every node is assigned to its own community. In this case, there are no intra-community edges in any community, so L_c is always 0, which means that the equation will produce a negative value.

Modularity-based community detection algorithms are based on the hypothesis that modularity is a measure of community structure in a network. Hence, they attempt to maximize this value to find the optimal community structure, if it exists. Many algorithms have been devised to use this measure for discovering communities in networks, the first being the Greedy Modularity Optimisation algorithm [Clauset et al., 2004]. This algorithm first assigns every node to its own community. Then, it calculates the difference in modularity, ΔM, for joining a pair of communities for each pair of communities that have an edge connecting them. The two communities that obtain the highest ΔM are joined. This process is repeated, registering the partitions and corresponding modularity values along the way, until there is only one community left. The partition with the highest modularity is returned.

In [Blondel et al., 2008], the Louvain algorithm is introduced. This is one of the most widely used algorithms for modularity optimisation. Its method is similar to the Greedy Modularity Optimisation algorithm, with a bigger focus on local changes. The algorithm starts off similarly, assigning every node to its own community. Then, every node is added to the the community of one of its neighbours, the one that yields the highest modularity increase. If there is none, the node stays in its own community. After all nodes have gone trough this process, all nodes belonging to the same community are aggregated into one 'node'. The previous steps are then repeated until there is no more modularity increase. The advantage of this method is that the number of nodes that need to be processed decreases after each iteration, because all nodes of a community are merged.

3.2.2 Overlapping community detection

So far, we have assumed that a node only belongs to a single community. However, this is not a straightforward assumption to make. It is conceivable that a node may belong to multiple communities. For example, a clustering can be based on the genre of books that authors wrote. In this case, an author may have written many books in two genres. This author can then be part of both communities. In the previous solution, only one of those communities is chosen. Overlapping community detection, however, finds communities that may or may not overlap.

In [Yang and Leskovec, 2013], the BigClam (Cluster Affiliation Model for Big Networks) algorithm is introduced. BigClam uses Maximum Likelihood Estimation (MLE) to derive soft clustering assignment vectors. A matrix F is initialized, where each row is a vector containing weight values for every possible community. More formally, for node u, F_u contains a weight value for each community in the set of communities C. To do this, the number of communities in the graph, k, has to be predetermined. This is done using the method described in [Airoldi et al., 2008]. For a given k, the BigClam model is fitted using 80% of the nodes in the network. Using the remaining 20% of the network, the likelihood of the model is calculated. The value of k that results in the highest likelihood is chosen. For very small networks, the Bayes Information Criterion (BIC) is used to select a value for k. Then, for a pair of nodes (u, v), the probability of an edge existing between them is modeled as follows:

$$p(u,v) = 1 - \exp(-F_u F_v^T) \tag{3.4}$$

Now, the probability of a given graph $G(V,\, E)$ can modelled as follows:

$$p(G) = \prod_{(u,v)\in E} p(u,v) \prod_{(u,v)\notin E} 1 - p(u,v) \tag{3.5}$$

From here, the following log-likelihood function can be constructed:

$$l(F) = \sum_{(u,v)\in E} \log(1 - \exp(-F_u F_v^T)) - \sum_{(u,v)\notin E} F_u F_v^T \tag{3.6}$$

This log-likelihood function is maximized using gradient descent. Then, given the resulting matrix F, a community membership threshold δ is determined. The value of δ is determined such that the edge probability of two nodes belonging to the same community is higher than the edge probability in a random graph. For weight vector F_u of node u, every community that has a weight value that is higher than δ is assigned to that node.

3.2.3 Hierarchical clustering

Clusters can also be present in the data on multiple 'levels', or, in hierarchies. Hierarchical clustering attempts to find hierarchies of clusters. This can be done by using a bottom-up approach, called agglomerative clustering, or in a top-down manner, which is called divisive clustering. The Louvain algorithm can be regarded as a form of agglomerative hierarchical clustering, as it starts with communities of single nodes and then merges these nodes, at every step returning a clustering result for that 'level'. Louvain proceeds this process until the modularity value stops increasing, but if you also return these intermediate steps, the Louvain algorithm can be used as a hierarchical clustering method.

3.3 Dynamic Community Detection Methods

Dynamic Community Detection (dynamic CD) is the pursuit of finding the set of all dynamic communities in a dynamic graph. Methods that perform dynamic CD can be categorized based on the way the dynamic graphs they work on are modeled. A first distinction can be made between dynamic CD algorithms using temporal graphs and dynamic CD algorithms using snapshot graphs, as is done in [Rossetti and Cazabet, 2018]. As our data is more suitable for the use of snapshot graphs, we will be focusing on algorithms using snapshot graphs. Furthermore, algorithms using temporal graphs can be considered *online* dynamic CD algorithms, which is not our use-case.

A further distinction can be made between dynamic CD methods, dividing them into three categories [Rossetti and Cazabet, 2018] [Greene et al., 2010]:

- *Instant-Optimal CD*: the class of Instant-Optimal CD algorithms encompasses algorithms that perform community detection on snapshots and attempt to match communities from these snapshots. The algorithms assume that a partition at time t is solely dependent on the current state of the network, at time t.
- *Temporal Trade-off CD*: Temporal Trade-off CD algorithms assume that communities at time t are not only dependent on the current state of the network, but also on the state of the network and/or partitions of the network at time steps earlier than t. A trade-off is then made between the optimal solution at time t and the influence of the past time steps.
- *Cross-Time CD*: the class of Cross-Time CD algorithms consider the whole timeline, i.e. all snapshots, and create a single partition of the network. They do this by first creating one graph out of all snapshots and them performing community detection on that graph.

3.3.1 Community instability

When using stochastic static community detection (static CD) algorithms, it is obvious that the solutions such an algorithm produces may vary significantly when applied multiple times to the same graph. Furthermore, even when using deterministic static CD algorithms, a small change in the graph can result in a significant change in the resulting partition. This phenomenon, called community instability, may also be a problem when trying to track the evolution of communities in a dynamic network.

Community instability can be explained in various ways. First, many algorithms, such as the Louvain algorithm, are only guaranteed to find a local maximum for a predetermined quality measure. The solutions that result from this may be close to the global maximum in terms of the quality measure, but a small change in the network can cause the algorithm to find a different local maximum, resulting in an entirely different partition. Additionally, non-overlapping community detection methods assign every node to only one community. As previously mentioned, it is reasonable to assume that a node may belong to more than one community. In this case, if an algorithm does not make this assumption, there can be many correct solutions in the search space, which can cause the aforementioned instability.

Bootstrapping is one way to solve this problem. For example, one can perform community detection multiple times on the same network. If the algorithm is not stochastic, one can run it on multiple slightly adjusted versions of the network. It is then possible to look for parts of the partition that are identical in most runs of the algorithm, these can be seen as the 'stable' parts.

Another possible way to smooth out the evolution of communities is to incorporate it into the community detection algorithms. This is often done implicitly in Temporal Trade-Off CD methods.

3.3.2 Instant-Optimal CD

In [Hopcroft et al., 2004], one of the first Instant-Optimal CD methods was introduced. The authors used a centroid-based agglomorative hierarchical clustering method. Similar to the Louvain algorithm, this method produces a tree of partitions. The proposed method first performs this clustering algorithm on a set of graphs that have been slightly modified (a small percentage of nodes have been randomly removed). It does this in order to find "natural communities", which are communities that are not susceptible to community instability. Networks from two different intervals are partitioned and the partition trees are compared, matching communities from the different intervals with each other.

In [Greene et al., 2010], another Instant-Optimal CD method is proposed. At every snapshot, a static CD algorithm is used to obtain a partition. Their method is not dependent on a specific algorithm, but in their experiments, they use the Louvain algorithm. A matching strategy is used that attempts to match communities in time step $t+1$ with communities in time step t, by calculating the Jaccard similarity between the communities. The Jaccard similarity measure represents the similarity between two sets by dividing the size of the intersection of the sets by the size of their union. If this value exceeds a certain threshold, the communities are considered to be part of the same dynamic community. Matches and splits are also recorded, by looking at the number of communities that a certain community is matched to.

In [Palla et al., 2007], a very similar approach is taken. They use the Clique Percolation Method (CPM) [Derényi et al., 2005] to perform overlapping community detection at each time step. Then, Jaccard similarity is used as a measure to match communities in different time steps. They produce some interesting insights from their analysis, the first one being that large communities change at a higher rate than small communities. Furthermore, they asses that, for small communities to 'survive' (have a long lifespan), it is optimal to remain static over time. In contrast, for large communities, it seems to be optimal to be more dynamic over time.

[İlhan and Öğüdücü, 2015] takes a different approach. They start similarly to the previously discussed methods, by creating a snapshot graph and performing a static CD method on the snapshots, in this case Louvain. Then, a similarity measure is used to compare and match communities in different time steps. However, this is only done for a subset of snapshots. For a given time window, a set of community features is calculated, such as the number of nodes in the community and the ratio of the sum of the degrees of the nodes in the community to the total number of nodes in the community. Then, they build a model to be able to predict these features in subsequent time steps. The features of this model are the community features and the labels are the community events that take place. This model is then used to predict the community events of a partition for subsequent time steps.

A different way to approach the matching process is to identify *core nodes* for communities and use these for matching communities, instead of using a similarity measure. These core nodes can be selected based on a centrality measure, for example. This is done in [Wang et al., 2008], where a centrality measure is used to select core nodes. Matching rules are defined based on the mutual presence of these core nodes in communities in different time steps.

[Chen et al., 2010] introduces a similar approach, where the core nodes are called *community representatives*. A community representative of a community is the node that has the minimum number of appearances in other communities in the graph. This definition relies on the assumption that is made that nodes can belong to multiple communities in a single time step. Again, matching rules are used that are partially based on the appearance of these community representatives.

3.3.3 Temporal Trade-Off CD

In [Aynaud and Guillaume, 2010], a slightly adapted version of the Louvain algorithm is used, in order to tackle the community instability problem. Instead of initializing the algorithm such that each node is its own community, the resulting partition from the last time step is used as the initial partition for the algorithm. Intuitively, this steers the algorithm in the direction of the local maximum it found in the previous times step, increasing the stability of the results.

In [Alvari et al., 2014], the dynamic CD problem is approached from an entirely different perspective. Instead of globally optimizing a performance measure, the problem is approached from a game theory perspective. Each node is seen as an agent that continuously tries to maximize its own utility. In each snapshot, agents iteratively have the opportunity to perform actions, such as becoming a member of a different community. They do this by using a utility function and choosing the action that maximizes this utility value. The utility function uses similarity values between pairs of vertices. An agents community membership from the last snapshot is also taken into account.

In this manner, a Nash equilibrium will be reached at some point in each snapshot. The actions agents take with regards to their community membership constitute the results of the dynamic CD.

As was previously stated, dynamic CD can also be done in an online manner. In [Rossetti et al., 2017], such an online method is introduced, the TILES algorithm. Starting with an empty graph, network events, such as the addition of nodes or edges, are streamed to the algorithm. Then, based on a set of rules, the partitioning of the graph is updated. The algorithm makes a distinction between *core nodes* and *peripheral nodes*. Peripheral nodes are not allowed to propagate their community membership to nodes they interact with, while core nodes do have that ability. After a predetermined interval, the algorithm returns a partition. After doing this for the lifetime of the graph, community events can be observed.

In [Folino and Pizzuti, 2010], the authors attempt to smooth out the evolution of communities by incorporating the stability of the communities into the quality function. More precisely, the quality function incorporates Snapshot Costs (SC) and Temporal Costs (TC) and is defined as follows:

$$\alpha \times SC + (1 - \alpha) \times TC \tag{3.7}$$

where α is a weight value, determining the importance of either cost values. SC measures the presence of community structure at a certain time step, while TC measures the similarity of the partition to the partition of the previous time step. In this case, a measure introduced in [Pizzuti, 2008] called *community score* was used as the SC and Normalized Mutual Information (NMI) was used as the TC.

In [Wang et al., 2018], the *topology potential field* is used to find community structure. The topology potential field is based on the interaction between nodes. Each node has a potential field around it and the potential fields of all nodes create the topology potential field. It has previously been used to identify essential proteins in protein interaction networks [Li et al., 2014].

The authors construct a static overlapping community detection algorithm using the topology potential field, after which a rule based approach is taken, where rules are determined based on the topology potential field. Community membership and community events are then determined by these rules.

The DynaMo algorithm, introduced in [Zhuang et al., 2019], maximizes modularity in order to find dynamic communities, but does it in an incremental way. In the first snapshot, the standard Louvain algorithm is performed in order to find a partition. Then, for all network events, such as node additions and edge deletions, the partition is updated in order to maximize modularity. After a while, if modularity does not exceed a certain threshold, the Louvain algorithm is applied again, to prevent the algorithm from getting stuck in a local maximum.

3.3.4 Cross-Time CD

In [Aynaud and Guillaume, 2011], a *sum graph* is created from the snapshot graph. This sum graph is the union of all snapshots, where the weight of each edge is the total time that edge has existed in the dynamic graph. This results in a weighted static network. The Louvain algorithm is then applied to this graph and the resulting communities are regarded as dynamic communities. The downside of this method is that it does not record community events, such as merges or splits. Additionally, this method assumes that nodes cannot switch their community membership.

Likewise, in [Jdidia et al., 2007], a static graph is created using the snapshot graph. However, this is done in a different way. Two types of edges are added between nodes: *identity edges* and *transversal edges*. An identity edge is added between nodes if the same node is present in subsequent time steps. A transversal edge is added between node u in time step t and node v in time step $t + 1$, if there exists a node w that is connected to v in time step t and is connected to u in time step $t + 1$. The Walktrap algorithm is then applied to this static network, resulting in dynamic communities [Pons and Latapy, 2005].

Chapter 4

Methods

In this section, the methods we employ in order tot analyse the data will be described. First, some data transformations are done. Then, two methods of community detection are applied, namely static community detection and dynamic community detection. Finally, some simple descriptions of the resulting communities are generated.

4.1 KG to HIN

As was discussed in section 2.2, KGs are quite similar to HINs. Our method revolves around converting the HIN into a homogeneous graph, based on meta-paths. Because of this, the KG does not have to perfectly adhere to the definition of a HIN. The KG is constructed in a way, such that the meta-path based conversion to a homogeneous graph can be easily executed.

This is done as follows. First, we have to construct a KG that is a conglomeration of multiple datasets, including, for example, the STCN and the Rijksmuseum Library datasets. This is done in a manner as to construct a simple network schema that satisfies the constraint that an edge type can only exist between two particular object types and that every object only has one type. In a KG, this means that every predicate only exists between two particular object types. The network schema of this KG remains simple, only containing that which will be needed for the conversion to a homogeneous graph. More precisely, it contains all object types and predicates that may be a part of the meta-paths that will be used for the conversion. The *type* predicate will not be regarded as an edge, but simply as the declaration of the type of the node. Every object in this simple schema only has one type.

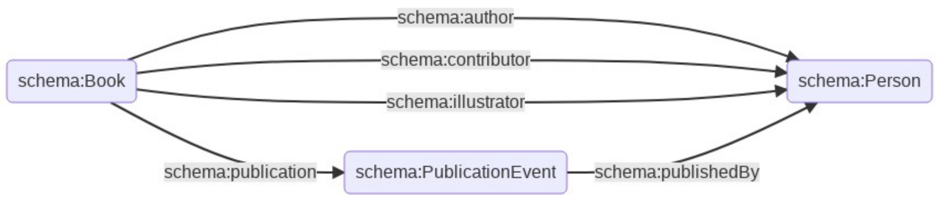


Figure 4.1: Network schema.

4.1.1 Network schema

The KG that is created has a simple schema (see figure 4.1). There are three types of objects: *Person*, *Book* and *PublicationEvent*. A *Person* can be an *author*, a *contributor*, an *illustrator* or a *publisher* (defined by the *publishedBy* relation). This is a simple schema that adheres to the definition of a HIN and contains only the necessary data to construct the meta-paths we need for community detection. It can also be easily expanded in the future, if it becomes apparent that more/other information is needed.

Apart from this subset of the data, organized in a KG with a simple schema, there is much more data available to us. This is stored separately and can be accessed by querying this dataset using URIs from objects in the constructed KG. This extra information, such as book titles and author names, can be seen as attributes, which would make the combination of the datasets similar to an AHIN, such as is used in [Li et al., 2017].

4.2 Meta-paths

From this simple schema, multiple interesting meta-paths can be defined. In this schema, the role of a *Person* regarding a *Book* is defined in the relation it has with *Book*. As there are multiple possible types of edges between *Book* and *Person*, it is necessary to define these relations in the meta-path if we want to distinguish between them. Furthermore, we assume that there exists an inverse relation for all edge types (or predicates). In reality, this is not the case and this results in having to define multiple separate paths in order to define one meta-path. However, by using this assumption, defining a meta-path based on co-authorship can be done as follows: $Person \xrightarrow{author} Book \xrightarrow{author} Person$. This notation can be further simplified by writing the role of $Person$ as if it is a direct subtype of $Person$. The same meta-path can then be written as follows: $Author - Book - Author$.

Now, the following meta-paths can be defined for this network schema:

- **Co-author**: $Author - Book - Author$
- **Co-contributor**: $Contributor - Book - Contributor$
- **Co-illustrator**: $Illustrator - Book - Illustrator$
- **Co-publisher**:$Publisher - Book - Publisher$

It is also possible to define a meta-path with which we do not distinguish between possible relations a *Person* can have with a *Book*. This meta-path would be $Person \xrightarrow{author/contributor/illustrator/publisher} Book \xrightarrow{author/contributor/illustrator/publisher} Person$. We can also write this more simply as:

- **Co-acip**: $Person - Book - Person$

Moreover, any combination of roles could be used to construct such a meta-path. However, using an arbitrary combination of roles could hurt the interpretability of the results.

	Co-author	Co-illustrator	Co-contributor	Co-publisher	Co-acip
Meta-path instances	0	700	326	20246	28589
Nodes	0	231	286	1163	2876
Edges	0	558	271	15432	21646
LCC (nodes)	0	188	12	881	2519
LCC (edges)	0	512	14	15201	21333

Table 4.1: Properties of the homogeneous graphs. (LCC: Largest Connected Component)

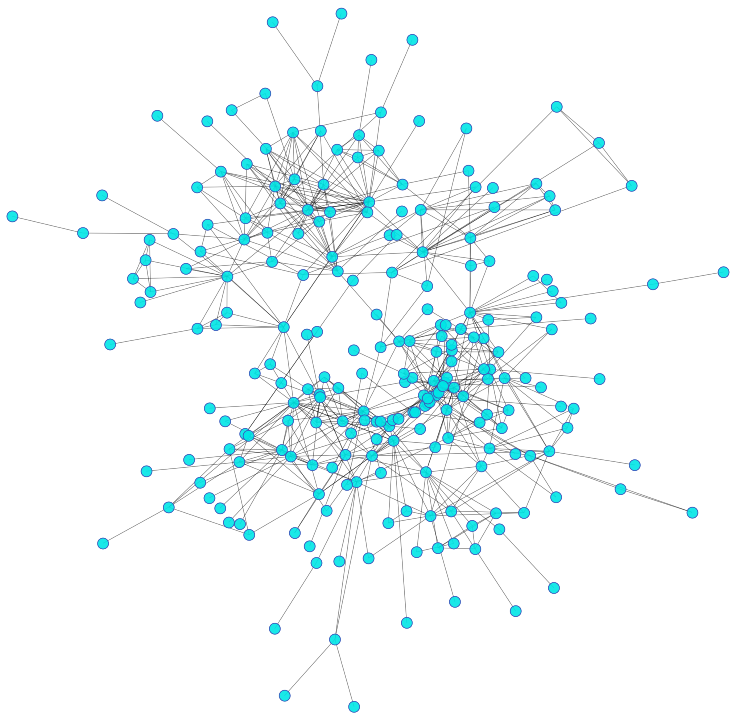


Figure 4.2: Co-illustrator network.

4.3 HIN to HG

Using these meta-paths, homogeneous graphs are created. This is done in the manner as described in section 3.1.6. We create weighted graphs, where the weights of the edges correspond to the number of instances of a meta-path between two nodes. In this case, this means that if two $Person$ nodes have cooperated on multiple books, the edge between them in the homogeneous graph has a higher weight.

Figure 4.2 contains a visualisation of the homogeneous graph constructed with the $Illustrator - Book - Illustrator$ meta-path. Table 4.1 presents an overview of some properties of the networks that were created. One of the properties listed here is the Largest Connected Component (LCC). At first glance, the LCCs for all three networks seem fairly small. The co-contributor network has a very small LCC, it consists of only

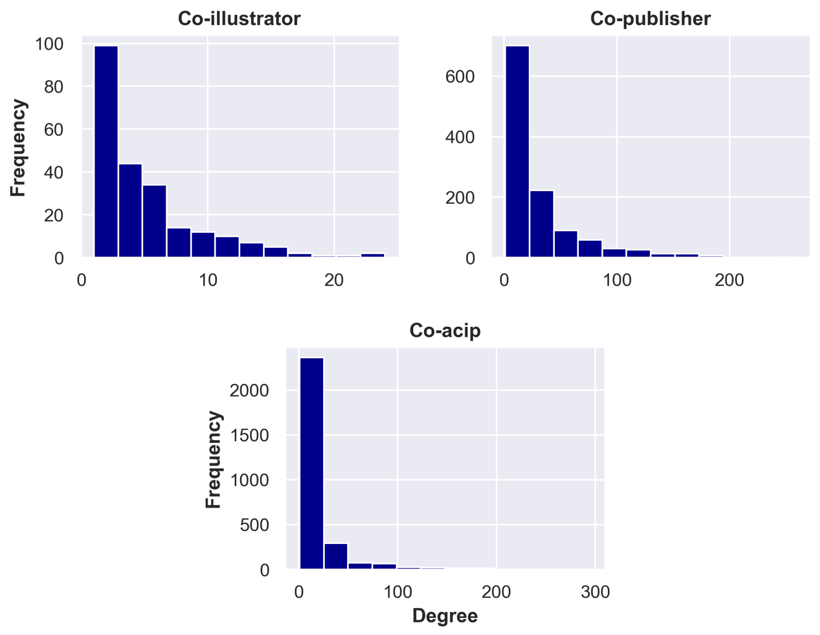


Figure 4.3: Degree histograms for the co-illustrator, co-acip and co-acip networks.

12 nodes, while there are 286 nodes in the co-contributor network. This is a ratio of 4.2%, which indicates that the network is a collection of small components, of which the size does not exceed 5% of the size of the network. Because of this, we will not be considering this network further. The co-publisher network is the second largest network that was constructed. It also has a large LCC, consisting of 881 nodes, which is 76% of the nodes of the full network. A visualization of the LCC of the co-publisher network can be seen in figure 4.4. The co-author network does not return any meta-path instances. This is because, in the dataset that authors are retrieved from, the STCN, only one author is connected to a book at all times. Because of this, we have to use a meta-path using combinations of roles to investigate author relations. The co-acip meta-path will be used for this purpose, which is the largest network that was constructed.

This leaves us with three HGs that we will be analysing and performing static and dynamic CD on, namely the co-illustrator HG, the co-publisher HG and the co-acip HG.

Figure 4.3 displays the degree histograms for the three constructed networks. In principle, these are social networks. As you would expect from social networks, the degree sequences of the networks seem to follow a power law, where there are a large number of nodes with a small degree and a small number of nodes with a very large degree. These nodes with high degrees are also called *hubs*.

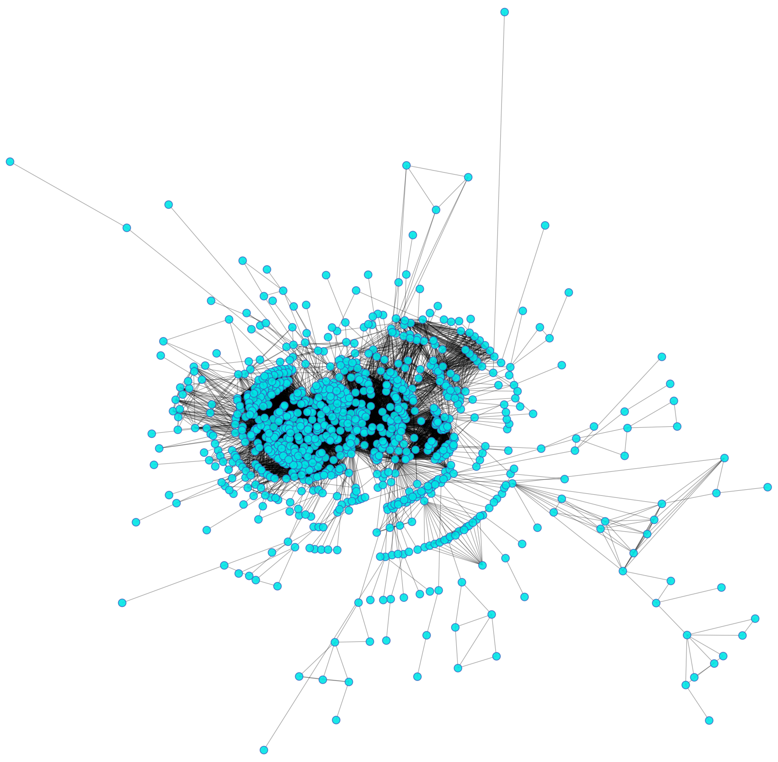


Figure 4.4: LCC of the co-publisher network.

4.4 Static Community Detection on HG

Now, in the pursuit of answering research question 1, static community detection can be performed on these resulting HGs. Multiple static community detection algorithms were tested and some internal evaluation measures were calculated, in order to determine which one performs best on our dataset.

4.4.1 Static CD algorithms

The following algorithms were used:

- **The Louvain algorithm**: See section 3.2.1. As the HGs we create are weighted graphs, a slight variation of modularity is used. When calculating the sum of edges in a graph or a group of nodes, or when determining the degree of a node, instead of simply counting the number of edges, the sum of their weights is taken.
- **Label Propagation (LP)**: See section 3.1.6.
- **Walktrap**: Walktrap is an algorithm that uses random walks to calculate distances between nodes. The idea behind the approach is that distances between nodes in the same community are smaller than distances between nodes from different communities [Pons and Latapy, 2005].

- **Infomap**: Infomap is also an approach that it based on random walks. Infomap measures information flow between nodes using random walks and uses this to partition the network [Rosvall and Bergstrom, 2008].

The algorithms were evaluated using the following internal evaluation measures:

- **Modularity**: See section 3.2.1.
- **Z-Modularity**: Z-Modularity is a variation of modularity that takes the statistical rarity of a partition into account by measuring the Z-score of the partition [Miyauchi and Kawase, 2016]. One of the main advantages of this variant of modularity is that it attempts to mitigate the *resolution limit* problem [Fortunato and Barthelemy, 2007]. The resolution limit is a limitation of modularity that may cause it to fail to identify small communities, depending on the size of the network.
- **Average Embeddedness**: The embeddedness $e(n, c)$ of a node n belonging to community c is the ratio of the degree of n within community c and the overall degree of node n. That is:

$$e(n, c) = \frac{k_{n,c}}{k_n} \tag{4.1}$$

  where $k_{n,c}$ is the degree of node n in community c and k_n is the overall degree of node n. The average embeddedness is calculated by taking the sum of the embeddedness for all communities and all nodes and normalising it by the number of communities. If this value is 1, then all communities are simply disjoint sets of nodes and the clustering results will not be interesting, because you would not need any community detection to find these sets of nodes. The average embeddedness tells us something about the structure of the network and by doing that, it helps with interpreting clustering results.
- **Conductance**: The conductance of a community within a graph is the ratio of external edges of the community with regards to the total number of internal edges of the community. It is defined as follows:

$$f(c) = \frac{e_{out,c}}{2 \times e_c + e_{out,c}} \tag{4.2}$$

  where $e_{out,c}$ is the total number of external edges of community c and e_c is the total number of internal edges of community c. The conductance of a partition can then be calculated by taking the mean of the conductance for all communities in the partition.
- **Cut Ratio**: The cut ratio of a community measures the ratio of the number of external edges of the community with regards to the total number of possible external edges. The total number of possible external edges is calculated as follows:

$$sum_{out}(G, c) = n_c \times (n_G - n_c) \tag{4.3}$$

where n_c is the number of nodes of community c and n_G is the number of nodes in the graph. Then, the cut ratio of a community c in graph G is defined as follows:

$$cr(G, c) = \frac{e_{out,c}}{sum_{out}(G, c)} \tag{4.4}$$

Again, to calculate the cut ratio of a partition, the mean of the cut ratio for all communities in the partition is taken.

Tables 4.2, 4.3 and 4.4 display the results of the experiments for the co-illustrator, co-publisher, and co-acip networks, respectively. The algorithms were applied to the full networks and to the LCCs of the networks. For the co-illustrator network, the Louvain algorithm outperforms the other algorithms in terms of modularity, for the full network and the LCC. This was expected however, because Louvain tries to maximize modularity. For Z-Modularity, LP performs best for both versions of the network, with Louvain being a close second. Looking at average embeddedness, conductance, and cut Ratio, the Infomap algorithm outperforms the other algorithms, performing quite well in conductance.

When looking at the co-publisher network, the results are similar. However, the Louvain algorithm outperforms LP in Z-Modularity. The Infomap algorithm performs significantly worse on this network, both on the full network and the LCC, achieving modularity scores of 0.048 and 0.018, respectively. The Z-Modularity scores are also significantly lower, being 0.208 and 0.111 for the full network and the LCC, respectively. The co-acip network is the largest network of the three. The results for this network are very similar to the results for the co-publisher network.

As it is difficult to define what makes a partitioning 'correct', we will choose an algorithm that performs well for a set of evaluation measures. Considering these results, the Louvain algorithm seems the right choice for static CD on the HGs. It performs significantly better regarding modularity, and is a close runner up for almost all other evaluation measures we looked at.

4.5 Transforming a Static Graph to a Dynamic Graph

To address research question 2, dynamic community detection must be performed. In order to perform dynamic community detection, we need to transform the static graph to a dynamic graph. As was explained in section 2.3, there are two ways to model a DG, namely as a temporal graph or as a snapshot graph. A temporal graph is created using birth and death timestamps of nodes in a graph. A snapshot graph can be created using specific timestamps of nodes or edges. Our method is focused on revealing more information about the collaboration networks in the data. Because of this, using timestamps of the edges, which represent books that nodes collaborated on, is a better approach from an analytical standpoint.

		Louvain	Infomap	Walktrap	LP
Full Network	# of Communities	23	17	31	37
	Modularity	**0.714**	0.549	0.682	0.666
	Z-Modularity	2.019	1.103	1.792	**2.060**
	Avg. Embeddedness	0.958	**0.988**	0.956	0.856
	Conductance	0.066	**0.003**	0.132	0.202
	Cut Ratio	0.002	**0.000**	0.002	0.003
LCC	# of Communities	8	2	15	21
	Modularity	**0.683**	0.478	0.646	0.632
	Z-Modularity	1.786	0.928	1.579	**1.810**
	Avg. Embeddedness	0.881	**0.981**	0.829	0.764
	Conductance	0.188	**0.026**	0.242	0.333
	Cut Ratio	0.006	**0.002**	0.004	0.007

Table 4.2: Evaluation measures of static community detection algorithms for the **co-illustrator** network.

		Louvain	Infomap	Walktrap	LP
Full Network	# of Communities	116	103	314	162
	Modularity	**0.550**	0.048	0.486	0.226
	Z-Modularity	**1.289**	0.208	0.9732	0.507
	Avg. Embeddedness	0.982	**1.000**	0.425	0.895
	Conductance	0.023	**0.000**	0.584	0.127
	Cut Ratio	0.001	**0.000**	0.001	0.003
LCC	# of Communities	15	3	206	54
	Modularity	**0.539**	0.018	0.477	0.203
	Z-Modularity	**1.290**	0.111	1.002	0.469
	Avg. Embeddedness	0.872	**0.992**	0.157	0.731
	Conductance	0.173	**0.011**	0.849	0.318
	Cut Ratio	0.007	**0.000**	0.003	0.001

Table 4.3: Evaluation measures of static community detection algorithms for the **co-publisher** network.

		Louvain	Infomap	Walktrap	LP
Full Network	# of Communities	152	138	489	395
	Modularity	**0.633**	0.373	0.544	0.548
	Z-Modularity	**1.537**	0.724	1.214	1.117
	Avg. Embeddedness	0.982	**0.997**	0.688	0.767
	Conductance	0.024	**0.004**	0.348	0.271
	Cut Ratio	**0.000**	**0.000**	**0.000**	**0.000**
LCC	# of Communities	20	5	345	256
	Modularity	**0.627**	0.347	0.535	0.539
	Z-Modularity	**1.491**	0.673	1.180	1.084
	Avg. Embeddedness	0.858	**0.919**	0.573	0.648
	Conductance	0.203	**0.119**	0.475	0.408
	Cut Ratio	0.001	**0.000**	0.001	0.001

Table 4.4: Evaluation measures of static community detection algorithms for the **co-acip** network.

Furthermore, creating a temporal graph using birth and death timestamps of nodes is unfeasible in this case. The ratio in which the birth and death timestamps of nodes are present in the data is too low. For the co-illustrator network, 70% of nodes have birth timestamps. For the co-publisher and the co-acip networks, only 22% and 29% of nodes have a birth timestamp, respectively. These percentages are much too low to be used to construct a DG. The edges do have timestamps that can be used to create a DG. All edges in all three HGs have timestamps. These timestamps represent publication dates. The timestamps are modeled as follows: each book has a *Publication* and each *Publication* has an *EarliestBeginningTimestamp* and a *LatestEndingTimestamp*. Upon further inspection, most of the books have an *EarliestBeginningTimestamp* and *LatestEndingTimestamp* that are identical, but there are some books for which these values can differ up to 100 years. This suggests that these publication timestamps are not exact and that there can be some uncertainty regarding the exact publication dates. As most of the timestamps of the books give an exact publication date, we do not have birth and death timestamps for the edges. We can use time intervals, along with the exact publication dates, to create snapshot graphs.

Apart from the fact that using the timestamps of edges makes more sense analytically, using time intervals to create a snapshot graph will also make the uncertainty of the timestamps of the books a smaller issue, because the intervals introduce some margin of error for the timestamps. Additionally, online CD algorithms, for which temporal graphs are often used, are not of much use to us, because we are working with historical data that has a set start and end time. Altogether, we will be converting the HGs to snapshot graphs.

4.5.1 Creating snapshot graphs

To create the snapshot graphs, the HG is partitioned into multiple smaller HGs, based on time intervals. Algorithm 1 gives a succinct description of the algorithm used to do this. First, a set of time intervals for the HG must be initialized.

Algorithm 1: Create snapshot graph

```
   input  : $HG$, $intervalsize$, $shift$
   output: $SG$
1  Determine set of intervals based on the $HG$, $intervalsize$ and $shift$
2  Initialize $SG$ as list of empty graphs
3  for all intervals do
4      for all edges in HG do
5          if timestamp(edge) falls in interval then
6              add edge (and corresponding nodes) to snapshot
7      Apply a static CD algorithm to snapshot
8      Calculate modularity for resulting partition
9      if $modularity == 0$ then
10         remove snapshot from $SG$
```

In order to do this, two parameters are needed, the interval size and the shift. The set of intervals is then determined by first determining the birth and death timestamps of the full HG. This is simply done by taking the earliest *EarliestBeginningTimestamp* and the latest *LatestEndingTimestamp* present in the graph. Then, starting from the starting timestamp of the graph, we slide a window through the timeline of the graph, where the window size is the interval size and the step size is the shift. Each step is recorded and this results in a set of intervals that can be used to generate a snapshot graph.

For each time interval, a snapshot is initialized as an empty graph. Then, all edges that have a timestamp that falls in this interval are added to the snapshot, along with the nodes they are connected to. As was mentioned above, the timestamp of an edge is determined by two values, *EarliestBeginningTimestamp* and *LatestEndingTimestamp*. These values are often identical, but in the case that they are not, the average of the values is taken. After a snapshot has been created for an interval, the snapshot is partitioned using a static CD algorithm.

In figure 4.5, the modularities of the partitions, along with the number of edges in the snapshots, have been plotted for all intervals of a generated SG. The SG was generated from the co-illustrator HG, using an interval size of 20 and a shift of 10. The Louvain algorithm was used to partition each snapshot. Looking at these figures, it looks like there is some correlation between the number of edges in the graph at a given interval and the modularity of the partition found in that interval. In the first and last few intervals of the graph, there are no or very few edges in the snapshot. These empty or very sparse graphs result in partitions with modularity values of 0.

Because these parts of the SG are not interesting, the intervals for which the partition has a modularity of 0 are removed from the SG. Doing this simply cuts off the first and last intervals of the SG, because they are either empty or too sparse.

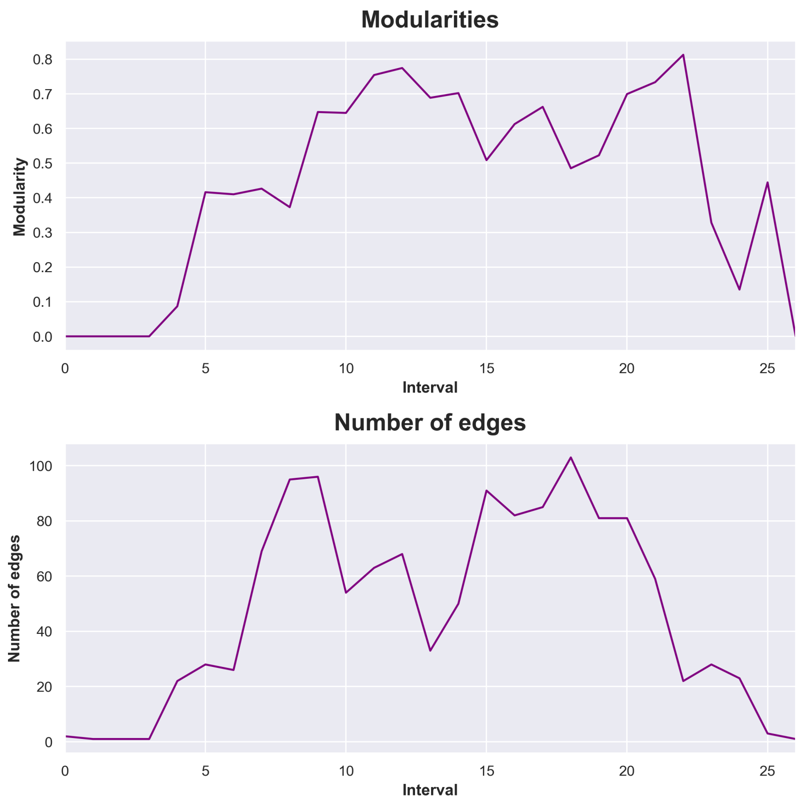


Figure 4.5: The modularities and the number of edges plotted for every interval, using the co-illustrator SG.

4.6 Community Detection on Dynamic Graphs

Given the snapshot graph, dynamic community detection is performed. The method we employ is mainly inspired by [Greene et al., 2010]. The algorithm in [Greene et al., 2010] is an *Instant-Optimal CD* method, because it performs static community detection at different time steps and attempts to match the partitions in consecutive time steps using a similarity measure, while making the assumption that partitions in different time steps are not dependent on each other. Our proposed method assumes that the partitions in different time steps are dependent on each other, making our approach fall into the category of *Temporal Trade-off CD* algorithms.

4.6.1 Algorithm overview

In algorithm 2, the pseudocode for our Dynamic Community Detection algorithm can be seen. The algorithm can be briefly described as follows: first, a static CD algorithm is applied to the first snapshot. For each community that was returned, a Dynamic Community (DC) is initialized. Every DC has a *head*, which is the last community that has been added to the DC. The *heads* of DCs are also updated when the DCs are updated. Then, at each consecutive snapshot, static CD is performed. For every community c_t in time step $t \neq 0$, the Jaccard similarity is calculated between it and the *heads* of all DCs. Based on these values, a matching threshold and a set of matching rules, communities are matched, the set of DCs is updated and the community events that have taken place are recorded.

Algorithm 2: Dynamic Community Detection

```
   input  : $SG$, $\theta$
   output: $DC$, $Events$
1  $DC \longleftarrow \{\}$
2  $Events \longleftarrow \{\}$
3  Apply static CD to first snapshot
4  for snapshot in SG do
5      Apply static CD to first snapshot
6      for community in partition do
7          Calculate Jaccard Similarity for all DCs
8          if $JaccardSimilarity \geq \theta$ then
9              Add community to matching communities
10         Update $DC$ with all matching communities
11         Apply Merge/Split rules, update $Events$
```

4.6.2 Community events

In order to compose a set of dynamic communities, we have to record community events. The following community events are recorded:

- **Birth**: a community appears for the first time.
- **Death**: a community appears for the last time.
- **Continuation**: a community is the continuation of a community from a previous time step, it remains mostly unchanged.
- **Merge**: multiple communities merge into a single community.
- **Split**: a community splits into multiple communities.

4.6.3 Matching rules

The following matching rules are used to detect these events:

- **Birth**: If community c_t matches with no DC, c_t is the start of a new DC.
- **Death**: If a DC has not been updated for more than the maximum community lifetime, *clt* time steps, it will not be updated anymore.

- **Continuation**: If community c_t matches with exactly one DC, and the DC matches only with community c_t, c_t is a continuation of the DC and becomes the *head* of that DC.
- **Merge**: If community c_t matches with more than one DC, c_t is a merge of those DCs. This event is recorded in *Events* and c_t is added to all those DCs, which means that they will be identical from this point on.
- **Split**: If multiple communities in time step t match with one DC, that DC has split into those communities in time step t. For all but one matching community in time step t, new DCs are created that have the same community history as the DC that was split.

The matching rules are applied after the Jaccard similarity has been calculated between a community c_t and a subset of the *heads* of all DCs, namely all the DCs that have not 'died' yet. As the *death* rule states, a DC that has not been updated for more than *clt* time steps, will not be updated anymore. Hence, when attempting to match community c_t to existing DCs, only the DCs of which the *heads* fall within *clt* time steps of c_t, are considered. During the process of calculating the similarities between c_t and the *heads* of the DCs, a set of matches is composed. This set of matches consists of all DCs, which are not *dead*, of which the *heads* have a Jaccard similarity with c_t that exceeds θ.

The application of the *birth* rule is trivial. The assumption is made that a community always belongs to a DC. Consequently, if a community c_t does not belong to any of the existing DCs, according to the matching rules, a new DC is created, of which c_t is the first community. The application of the *continuation* rule is straightforward as well. If community c_t matched with only one DC, and the DC matched only with community c_t, it is simply added to that DC. Naturally, as it is the last community that has been added to that DC, it also becomes the *head* of the DC.

The *merge* rule is applied if the list of matches for c_t contains more than one DC. c_t is then added to all these DCs, which results in these DCs being identical from time step t on. The *split* rule must be applied if multiple communities in time step t match with one DC. New DCs are created that have an identical history to the DC that has been split, up until the time step of the split. The event is also recorded in the set of *events*, for the purpose of easily being able to produce where splits have occurred in the timeline. This is also done for all *merge* events. The criteria for the *merge* event are checked first, after which those for the *split* event are checked. This means that if both occur simultaneously for the same set of DCs, the *merge* event has the priority and gets registered.

After applying the *birth*, *continuation*, *merge* and *split* rules as described above, the *death* rule is implicitly applied as well. This is a result of the usage of the maximum community lifetime, *clt*, when matching and applying those rules. A DC is not considered for the *continuation*, *merge* and *split* rules if it has not been updated for *clt* time steps. In this manner, after a DC has not been updated for *clt* time steps, it is never updated again.

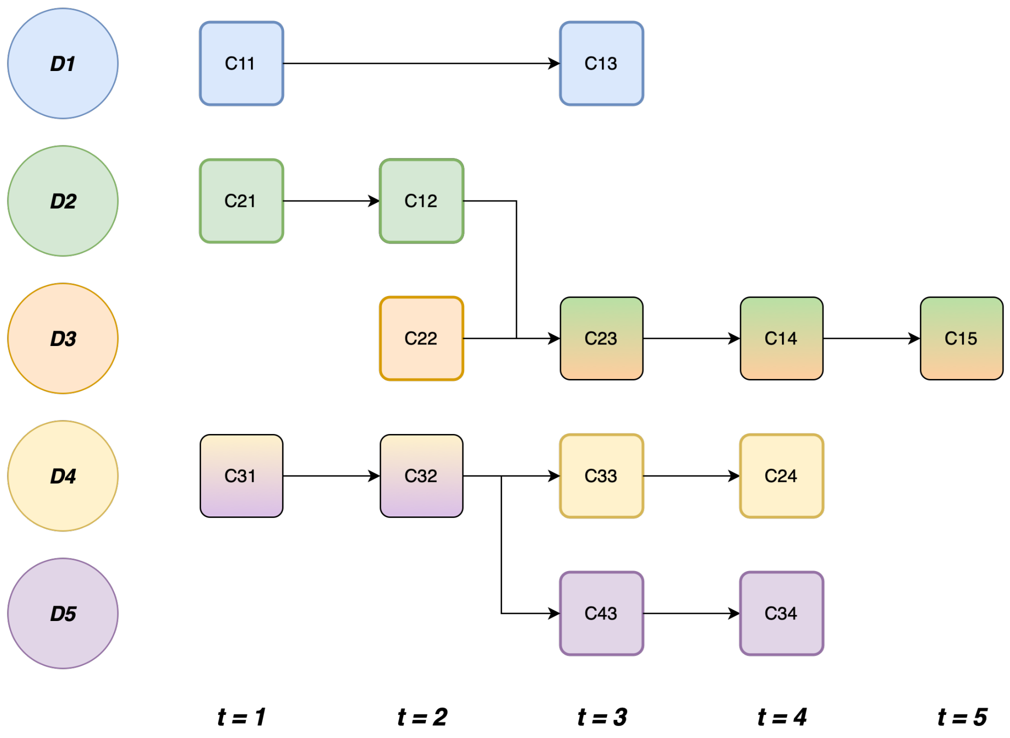


Figure 4.6: An example of a possible timeline of a set of DCs.

4.6.4 DC timeline

The result of the matching process is a set of DCs and a set of events, which together form a DC timeline. In figure 4.6, a visualization of a DC timeline can be seen. In this timeline, there are 5 DCs. Every DC is simply a list of communities. The communities are written as Cxy, where y denotes the time step of the community and x is a unique identifier for the community in that time step. The DCs in figure 4.6 can be written in list-form as follows:

D1 : $[C11, C13]$

D2 : $[C21, C12, C23, C14, C15]$

D3 : $[C22, C23, C14, C15]$

D4 : $[C31, C32, C33, C24]$

D5 : $[C31, C32, C43, C34]$

For this DC timeline, the following events were registered:

t=3 $\{merge : D2,\ D3\}$

t=3 $\{split : D4 \rightarrow (D4,\ D5)\}$

Looking at the timeline, one can see that $D1$ consists of two communities, $C11$ and $C13$. This is an example of a DC not being present in a time step, but still continuing to exist, because it is not yet dead. Here, the *continuation* event took place at $t = 3$. Let us say that the threshold for the death of a DC is set to 2, meaning that if a DC has not been updated for 2 time steps, it is considered dead. Using this threshold, $D1$ is considered dead after $t = 5$, because it has not been updated for 2 time steps after that point.

At time step $t = 3$, DCs $D2$ and $D3$ merge. In the timeline visualization, they continue as one DC. In practice, $D2$ and $D3$ are identical from this point forward, as can be seen in the lists of communities the DCs consists of shown above.

At time step $t = 3$, $D4$ splits into $D4$ and $D5$. In the timeline visualization, the two DCs start as one, splitting into two at $t = 3$. In practice, after the construction of the lists of DCs is completed, $D3$ and $D4$ are identical up to and including $t = 3$, after which their timelines become distinct. This can also be seen in the lists of communities the DCs consists of shown above.

4.6.5 Choosing the right static CD algorithm

On the HGs, the Louvain algorithm seemed to perform best. However, for an algorithm such as Louvain, which finds a local maximum for the modularity measure, community instability may be an issue. To tackle this problem, we experimented with a slightly adjusted version of the Louvain algorithm, as introduced in [Aynaud and Guillaume, 2010] (see section 3.3.3). To briefly recap, instead of initializing the algorithm with each node belonging to its own community, this version of the Louvain algorithm initializes it with a given partition, the partition from the previous time step. We will call this variation of the algorithm Louvain Initialize Partition, or LIP in short.

The Louvain algorithm also has a stochastic element. The first step of the algorithm involves merging nodes based on the increase in modularity resulting from this merge. This has to be done in a certain order, which is randomized. Another way to attempt to stabilize the algorithm is to execute it multiple times on the same graph and choose the partition that achieved the highest modularity. This variation of the algorithm, from here on called Louvain Randomized (LR), was implemented using 10 rounds, meaning that the Louvain algorithm is run 10 times before the best preforming partition is chosen.

In order to choose which variation of the algorithm to use, we looked at the quality and the stability of the partitions. First, let us look at the modularities of each partition in the timeline for the different variations of the Louvain algorithm. To do this, the means of the modularities for each variant of the algorithm were calculated. The SG that was used was generated using an interval size of 20 and a shift of 10. Table 4.5 displays the results of these experiments. There is no difference between the standard Louvain algorithm and LIP, regarding the mean modularity values they produce. This was to be expected, because LIP simply starts its search from a different starting point, which may lie closer to a local maximum than a random point in the search space, but there is no reason to believe that the local maximum that is found will be higher

	Co-illustrator	Co-publisher	Co-acip
Louvain	0.541	0.605	0.659
LIP	0.541	0.605	0.659
LR	0.542	0.606	0.661

Table 4.5: Mean modularity values for partitions of an SG, generated by Louvain and its variations.

than that of the standard Louvain algorithm. LR, however, seems to perform slightly better than both the standard Louvain algorithm and LIP. It intuitively makes sense that if you take the maximum of multiple runs of the algorithm, that that can result in a partition with a higher modularity score. However, the increase here is so small that it is negligible.

We looked at the stability of the partitions by calculating a number of similarity measures between every consecutive partition in the timeline. The similarity measures that were used are:

- **Normalized Mutual Information (NMI)**: Mutual Information (MI) is a measure used to compare two partitions, based on the amount of information from one partition that can be inferred from the other partition. NMI is simply a variation of MI that normalizes its values, so that they fall between 0 and 1.
- **Adjusted Mutual Information (AMI)**: AMI is a variation of MI that accounts for chance. The MI tends to be higher for partitions with more clusters, because they tend to share more information. AMI takes this into account. The AMI has an expected value of 0 for independent partitions and a value of 1 for identical partitions.
- **Adjusted Rand Index (ARI)**: The Rand Index (RI) measures the similarity between two partitions by looking at all possible pairs of nodes and how often these pairs belong to the same cluster across two partitions. The ARI is a variation of RI that accounts for chance, similarly to the AMI. The ARI also has an expected value of 0 for independent partitions and a value of 1 for identical partitions.

An SG was generated using the co-illustrator network, with an interval size of 20 and a shift of 10. However, in order to calculate the similarity measures, the SG had to be generated slightly differently. To be able to compare two partitions, the partitions must contain the same set of nodes. Our method only adds nodes to a snapshot if they are connected to an edge that has a timestamp that falls in the interval of the snapshot. For the purpose of the upcoming experiments regarding partition stability, all nodes of the graph are added to all snapshots. If the edges of these nodes do not fall in the interval of a snapshot, the node becomes a separate component of that snapshot.

All three Louvain variations were used to partition all snapshots and the proposed similarity measures were calculated between consecutive snapshots. In figure 4.7, the

NMI can be seen for all three variations. The stability curve looks almost identical for all variations. The value of the NMI also remains fairly high through the timeline, having a minimum of around 0.95 for all three variations.

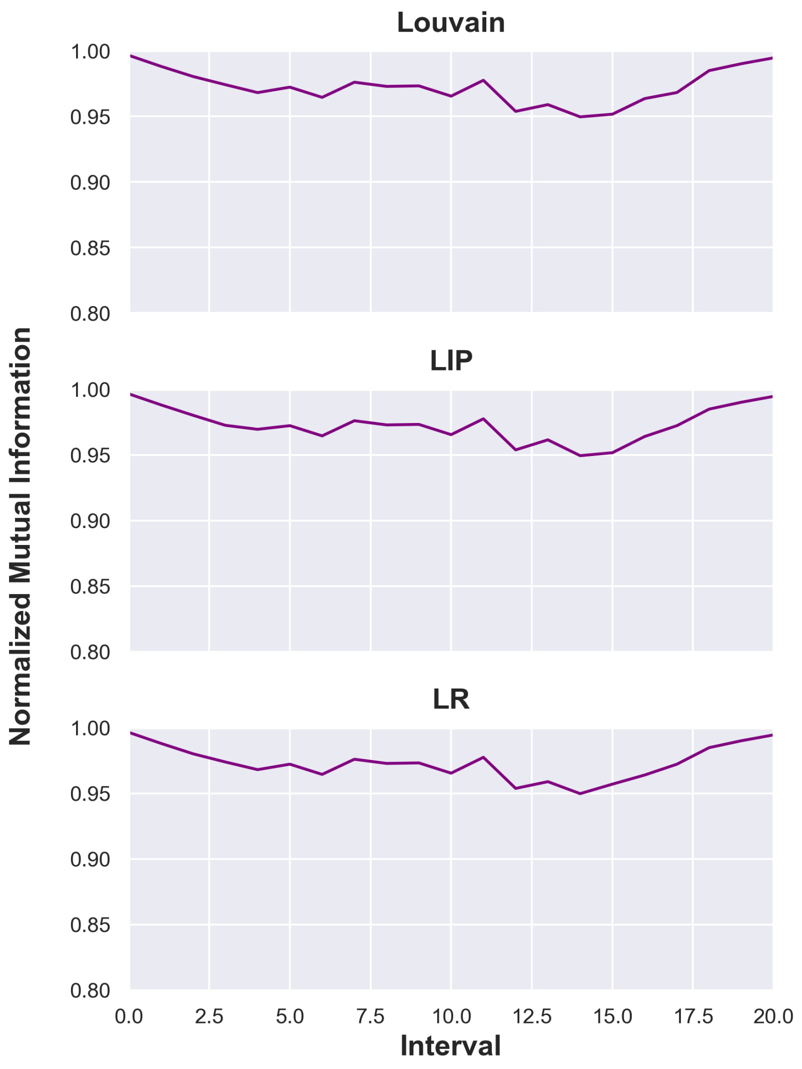


Figure 4.7: The stability trends for all Louvain variations, using the co-illustrator SG (interval size = 20, shift = 10).

To confirm that these values are not this high solely because there is a time-overlap of 50% between consecutive snapshots, we also calculated the NMI for the same SG, but generated using an interval size of 20 and a shift of 20, which means that there is no time-overlap between consecutive snapshots. In figure 4.8, these results can be seen. Albeit being slightly lower, the NMI still remains fairly high, not dipping under 0.90

In tables 4.6, 4.7 and 4.8, the NMI, AMI and ARI can be seen for the co-illustrator, co-publisher and co-acip SGs, all generated with an interval size of 20 and a shift of 10. Here, one can see that the AMI and the ARI are significantly lower than the NMI for all three networks and for all Louvain variations. This makes sense, as these measures

	Louvain	LIP	LR
NMI	0.973	0.973	0.973
AMI	0.440	0.437	0.442
ARI	0.355	0.356	0.355

Table 4.6: Similarity measures for co-illustrator SG (interval size = 20, shift = 10).

	Louvain	LIP	LR
NMI	0.976	0.976	0.976
AMI	0.484	0.482	0.484
ARI	0.380	0.387	0.387

Table 4.7: Similarity measures for co-publisher SG (interval size = 20, shift = 10).

are adjusted for chance. The values for AMI are still between 0.400 and 0.500 and the values for ARI are still between 0.300 and 0.400, which indicates that the partitions are still fairly similar, even when adjusted for chance.

Looking at the difference between the variations of the Louvain algorithms, the results are very similar. The NMI is identical, up to three decimals, for all three variations, for all three SGs. Some difference in AMI and ARI can be observed, but these are slight differences. For example, LR scores higher (or equal to, in one case) than Standard Louvain and LIP in AMI in all three SGs. LIP scores higher than LR in ARI in the co-illustrator SG, but lower than LR in the co-acip SG. Altogether, LR seems to outperform the other two variations slightly in most cases. However, the difference is too small to base our choice on. On account of this, we will base this choice on other criteria.

The main consideration when choosing which algorithm to use is maximizing the probability that the algorithm finds the same local maximum for subsequent snapshots. We find the LIP variation of the Louvain algorithm best suited for this, because the algorithm's starting point in the search space is the last partition. Intuitively, this should help the algorithm find the same local maximum more easily. The randomized version of Louvain, LR, could also be a consideration here, but from a different perspective. It could be argued that by running the algorithm multiple times and choosing the partition with the highest modularity score, the algorithm may find the optimal solution in subsequent snapshots. However, if this would be the case, one would expect

	Louvain	LIP	LR
NMI	0.976	0.976	0.976
AMI	0.457	0.460	0.462
ARI	0.320	0.323	0.328

Table 4.8: Similarity measures for co-acip SG (interval size = 20, shift = 10).

a significant difference between the mean modularity that LR achieves, as opposed to the other two variants. As was pointed out previously, this is not the case, as can be seen in table 4.5. There is a difference, but it is very slight. Hence, we will disregard this perspective, and use the LIP algorithm from this point on.

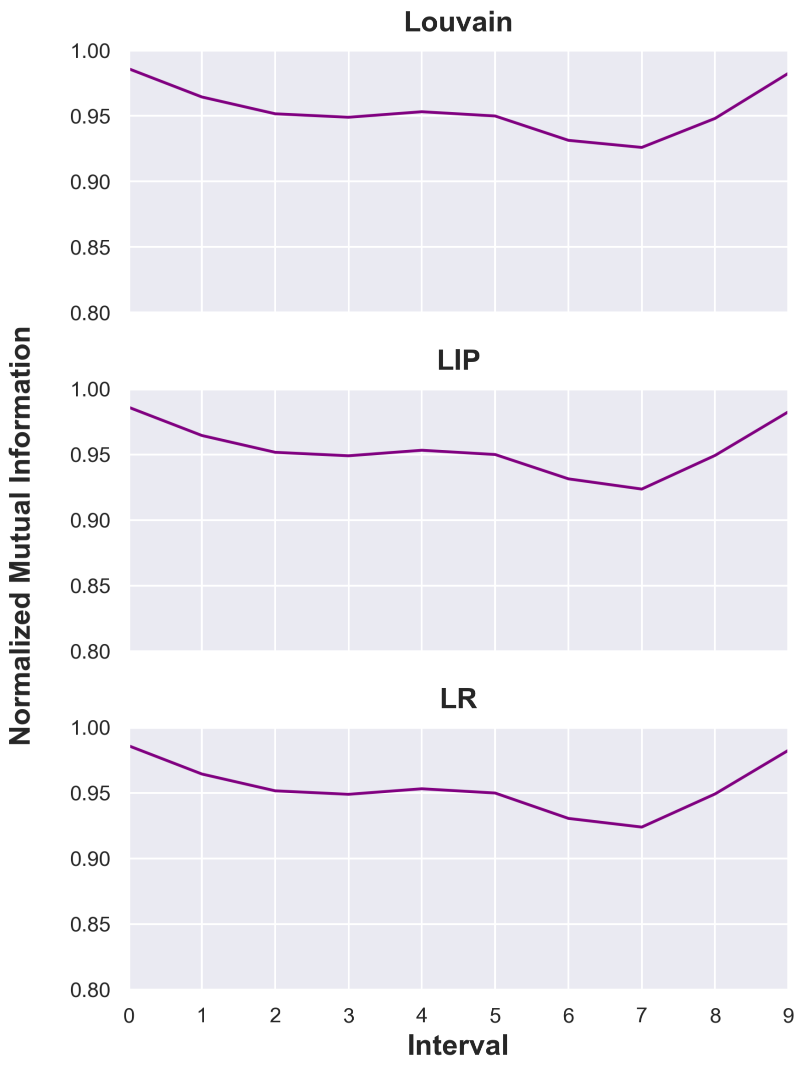


Figure 4.8: The stability trends for all Louvain variations, using the co-illustrator SG (interval size = 20, shift = 20).

4.6.6 Determining interval size and shift

To choose an interval size and shift, we make multiple considerations. First, we can take a look at the stability of the partitions when using different values for interval size and shift, in the same manner as was done when choosing a static CD method. Additionally, the number of DCs and the average size of the DCs will be taken into account. Furthermore, an important consideration is the input of the historians. As they are familiar with the data and the time period of the data, they can have a certain preference regarding the interval size and shift that should be used. This preference is also taken into account.

	NMI	AMI	ARI
10_1	0.996	0.870	0.828
10_5	0.986	0.491	0.414
10_10	0.976	0.117	0.087
20_2	0.992	0.849	0.780
20_10	0.973	0.444	0.354
20_20	0.954	0.059	0.038
40_4	0.983	0.801	0.727
40_20	0.947	0.384	0.277
40_40	0.921	0.045	0.027

Table 4.9: Mean NMI, AMI and ARI for different combinations of interval sizes and shifts (represented as: intervalsize_shift), for the co-illustrator SG.

Community stability

If the shift is equal to the interval size, there is no time-overlap between consecutive snapshots. However, having some time-overlap between consecutive snapshots can help tackle the aforementioned community instability problem.

Using a smaller shift, and thus having a higher time-overlap between consecutive snapshots, increases the expected number of nodes that are present in both snapshots, thus increasing the expected similarity between the snapshots. Consequently, if two consecutive snapshots are more similar, there is a smaller chance of a static CD algorithm finding significantly different partitions. Primarily, this is the result of the two networks themselves being more similar. Secondly, let us take a deterministic static CD algorithm that finds a local maximum for a certain measure. For two networks, if the starting points of the algorithm in the search space lie close to each other, there is a higher probability of the algorithm finding the same local maximum, as opposed to having starting points that are further apart from each other. Furthermore, if partitions are less stable, the number of DCs may increase as a result. More unstable communities can have the effect that there are less matches between time steps. As was explained in section 4.6.3, if a community in a time step is not matched with any DC, a new DC is created containing only that community. In this manner, the number of small DCs, including DCs containing only one community, may increase. It goes without saying that a large set of very small DCs is not a desirable result.

Three different interval sizes were considered, namely 10, 20 and 40 years. For every interval size, a set of shift values was used, based on percentages of the interval size. The following percentages were used: 10%, 25%, 50% and 100%. Table 4.9 displays the mean NMI, AMI, and ARI, for all combinations of interval sizes and shifts, for the co-illustrator SG. For all three measures, the values decrease both when the shift increases, as well as when the interval size increases. Therefore, the smallest interval size, 10, performs best in terms of these stability measures. Additionally, the smallest shift, 1, also performs best in terms of these measures.

	Co-illustrator	Co-publisher	Co-acip
10_1	0.578	0.576	0.649
10_5	0.551	0.577	0.648
10_10	0.541	0.600	0.649
20_2	0.569	0.584	0.646
20_10	0.541	0.605	0.659
20_20	0.523	0.596	0.655
40_4	0.557	0.625	0.676
40_20	0.513	0.634	0.686
40_40	0.466	0.657	0.666

Table 4.10: Mean modularity for different combinations of interval sizes and shifts (represented as: intervalsize_shift), for the all three SGs.

	# of snapshots
10_1	261
10_5	55
10_10	27
20_2	146
20_10	30
20_20	15
40_4	74
40_20	15
40_40	8

Table 4.11: The number of snapshots in the SGs for different interval sizes and shifts (represented as: intervalsize_shift).

Mean modularity

Table 4.10 displays the mean modularity values for different interval sizes and shifts, for all three SGs. Here, for the co-illustrator SG, there are two trends that can be noticed. The modularity decreases as the shift increases and as the interval size increases. However, these trends are not observed for the co-publisher and co-acip SGs. The combination of interval size and shift that achieves the highest modularity for the co-illustrator SG is 10_1. For the co-publisher SG, 40_40 achieves the highest mean modularity and for the co-acip SG, 40_20 achieves the highest mean modularity.

Interpretability

A different consideration to make when choosing the interval size and shift values is the interpretability of the results. In table 4.11, the number of snapshots is displayed for the different interval size and shift combinations. As expected, when using a small interval size, the number of intervals in the SG increases. This may offer more granularity, but also makes the results less easily interpretable. The same goes for the value of

	Co-illustrator		Co-publisher		Co-acip	
	# DCs	avg DCL	# DCs	avg DCL	# DCs	avg DCL
10_1	99	20.98	308	13.77	768	15.01
10_5	84	3.10	280	2.66	636	2.31
10_10	87	1.39	270	1.32	574	1.24
20_2	96	15.29	267	11.31	603	9.72
20_10	74	2.46	231	2.25	494	1.96
20_20	74	1.16	217	1.15	429	1.12
40_4	71	12.66	254	9.33	546	8.95
40_20	15	2.02	201	1.90	389	1.71
40_40	8	1.13	285	1.09	297	1.03

Table 4.12: The number of DCs (# DCs) and the average length of the DCs (avg DCL), for different interval sizes and shifts (represented as: intervalsize_shift), for all three SGs.

shift, where a smaller shift also results in more snapshots. For example, using the 10_1 combination, the SGs will have 261 snapshots, while the 10_10 combination results in 27 snapshots and the 20_10 combination results in 30 snapshots. A high number of snapshots means that the information is much more difficult to analyse. This makes the search for interesting information in the data, which can lead to interesting insights, more difficult.

The same can be said for the number of DCs and the average length of DCs in an SG. In table 4.12, the number of DCs and the average length of the DCs, for all three SGs, for different interval sizes and shifts, is displayed. Increasing the interval size and increasing the shift both lead to a decrease of the number of DCs in an SG and the average length of the DCs in the SG. Using the same line of reasoning, it is not desirable to have a high number of DCs, as this makes analysis more difficult. The average length of the DCs is higher when the interval size or shift is lower, because consecutive snapshots are more similar in that case. This can be explained by looking at the stability measures displayed in table 4.9, in which lower interval sizes and shifts result in more stable SGs. However, having only very large DCs, as is the case with the 10_1 combination, might also not be preferable. The actual interval a DC spans may be the same interval it would span when detected with an SG created with the 20_10 interval. This means that the the data is simply more granular, a DC is cut up into more intervals. The same information would then be displayed in a way that is more difficult to analyse, which would not be preferable.

Finally, to choose the values for these parameters, we took all the aforementioned aspects into consideration. Firstly, the domain experts, the historians, voiced that they prefer a smaller interval size. Their reasoning for this preference was that smaller intervals would give them more granular information in terms of timestamps. Furthermore, we do not observe a specific value for interval size or shift that consistently achieves the highest mean modularity value for all three SGs. As was mentioned previously, we also want to limit the number of snapshots, the number of DCs and the average length of the DCs, which decrease as the interval size and shift increase. As the preference of the historians is a smaller interval size, we disregard the interval size of 40. Using shifts of 10% for both the interval sizes of 10 and 20, 10_1 and 20_2, generates many snapshots, 261 and 146, respectively. As a result, these are also disregarded. The shifts of 100% for both remaining interval sizes, 10_10 and 20_20, do result in a moderate number of snapshots and number of DCs, but the average length of the DCs remains below 2 for both, which means that many of the DCs only consist of one community. As was explained before, this could be because the SGs created with these combinations are less stable. Hence, these are also disregarded.

This leaves two combinations, namely 10_5 and 20_10. 10_5 results in a slightly higher mean modularity for the co-illustrator SG, but for the co-publisher and the co-acip SGs, 20_10 achieves a higher mean modularity. 20_10 also results in much less snapshots than 10_5, 55 vs 30, respectively. The number of DCs and the average length of the DCs are fairly similar, 10_5 generating 84 DCs vs 20_10 generating 74 DCs for the co-illustrator SG. For the co-publisher SG, these values are also fairly similar. For the co-acip SG they do differ significantly, being 636 for 10_5 and 494 for 20_10. The average lengths of the DCs are 3.10 for 10_5 and 2.46 for 20_10, for the co-illustrator SG. For the co-publisher and co-acip SGs, the differences are similar. Considering that for two out of the three SGs, 20_10 achieves a higher mean modularity, and that the stability of the partitions of both SGs are similar, but 10_5 generates almost double the number of snapshots as 20_10, we will be using 20_10 for our analysis.

4.6.7 Determining θ

The final parameter that we need to set is θ. As was explained in section 4.6.1, θ is the similarity threshold for determining whether two communities match. Choosing a low θ means that communities match very easily, even if they are not that similar. Intuitively, one would say that this would result in a small number of large DCs. Using the same intuition, choosing a high θ should result in a large number of small DCs, as few communities will match with each other, so many new DCs are created that remain small. Evidently, this also impacts the number of merges and splits that occur. A smaller θ would increase the number of merges and splits, and a higher θ would decrease them.

Experiments

In [Greene et al., 2010], various experiments were done to determine the best θ value. They generated multiple DGs using benchmarks, in which they introduced community events. In this manner, a ground truth was generated with which the results of their algorithm could be compared. This comparison was done by calculating the NMI between the output of their algorithm and the ground truth. Based on the experiments, a θ of 0.3 was deemed to best capture all community events.

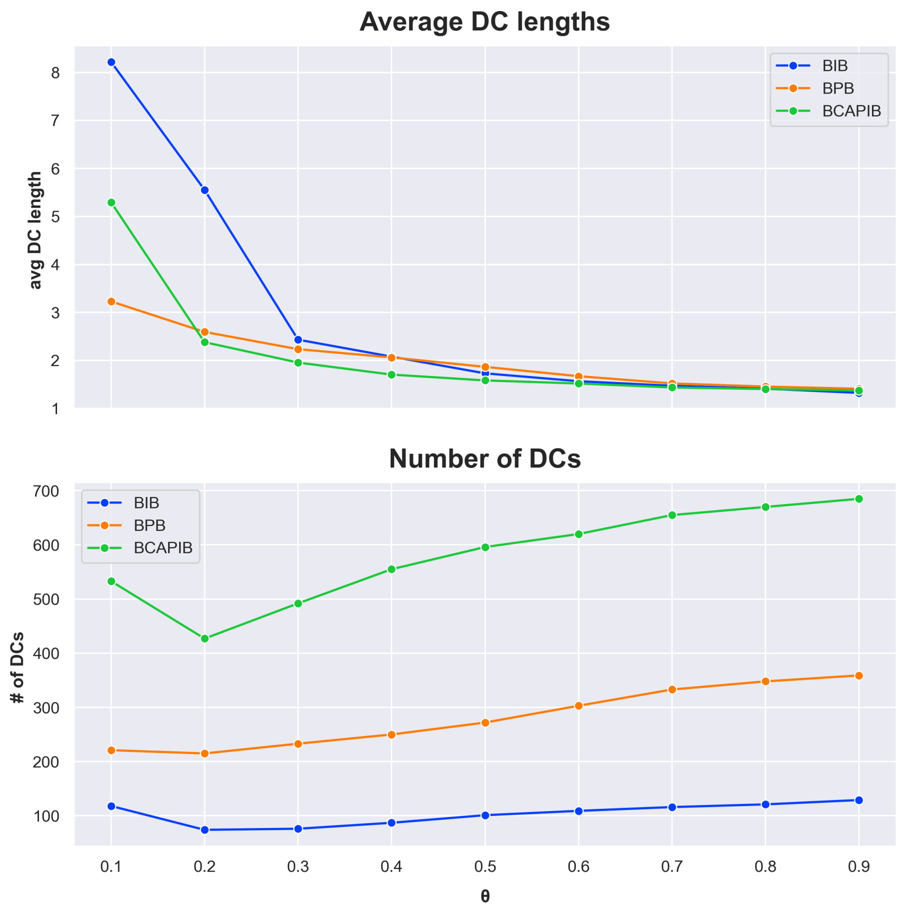


Figure 4.9: The number of DCs and their average length, using various values for θ, in both figures.

DC properties

In figure 4.9, the number of DCs and the average length of DCs are plotted, after dynamic CD was performed using various values for θ. The values that were used were values in the interval $\left[0.1, 0.9\right]$, using a step size of 0.1. For all SGs, the average DC length starts off high and decreases quickly. Then, there is a point after which the average length keeps decreasing, but at a slower pace. This point is around 0.2 and 0.3. A good argument to make here is to avoid the range that displays a high amount of volatility, which is the lower range of θ, 0.1 or 0.2.

The number of DCs starts off high with the co-illustrator and the co-publisher SGs, then decreases quickly, after which it slowly increases again. The high starting point and quick decrease could be the result of many splits occurring at 0.1, which causes new DCs to be created. This effect may then die down at 0.2, after which the number of DCs starts increasing, because communities are less likely to match given a higher θ. Again, one could argue that it is better to avoid the big fluctuations that occur using

	Co-illustrator	Co-publisher	Co-acip
Birthdate	0.70	0.22	0.29
Birthplace	0.87	0.25	0.33
Religion	0.37	0.13	0.17
Occupations	0.90	0.29	0.36
Occupational adress	0.90	0.29	0.35
Parents	0.69	0.20	0.25
Children	0.35	0.12	0.13
Spouses	0.65	0.25	0.28

Table 4.13: Availability of attributes for all SGs.

the lowest values. However, as was mentioned before, we also want to limit the number of DCs that are returned, so it is not advisable to use the highest values of θ .

Taking these arguments into consideration, we will be using the same value as [Greene et al., 2010] did, namely 0.3. It avoids the high fluctuations of the lowest values of θ, and it has some experimental backing.

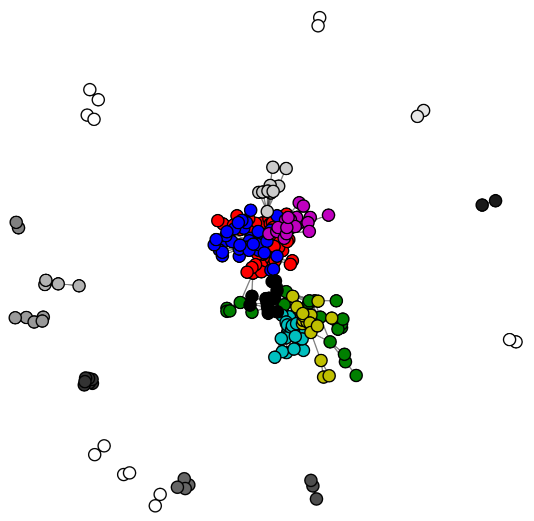
4.7 Generating Descriptions

Lastly, in order to increase the interpretability of the returned DCs, we generated descriptions of the DCs using attributes, addressing research question 3. These attributes were present in an external datastore, which we could query using the URIs of the nodes in the graph. The attributes were attributes of the nodes, which represent people. The attributes that were used are:

- **Birthdate**: date
- **Birthplace**: categorical
- **Religion**: categorical
- **Occupation**: categorical
- **Occupational address**: categorical
- **Family relations**: categorical (in practice, the family relation attribute is composed of three attributes: parents, children and spouses. We simply add all these together to get the family relations attribute)

4.7.1 Missing data

One problem we encountered was the fact that not all attributes were present for all nodes. Table 4.13 displays the ratio in which the attributes are present in the SGs. The availability of the attributes varies a lot between the different attributes and even


(a) Full network

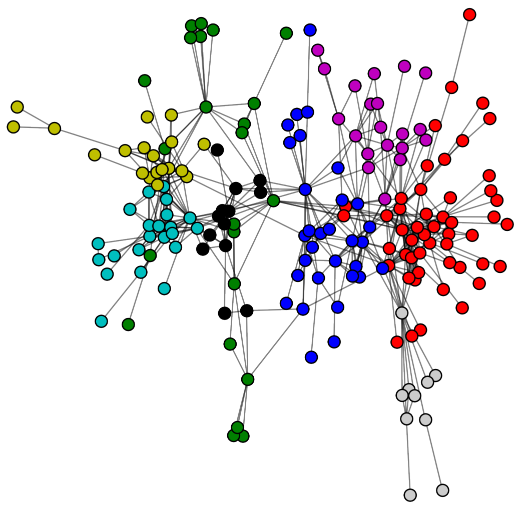

(b) Largest connected component (LCC)

Figure 4.10: Partitioned co-illustrator network.

between the different SGs. The lowest availability is that of the children attribute for the co-publisher SG, namely 0.12, and the most frequently available attributes are occupation and occupational address, which both have an availability of 0.90. For the co-publisher and the co-acip SGs, there are significantly less attributes available than for the co-illustrator network. Overall, there is enough data available such that we can generate descriptions.

4.7.2 Method

To generate a description for a DC, we first take the union of all communities a DC consists of. This gives us a list of unique nodes. Then, we simply take the most frequent value for the categorical attributes and we take the median of the year of birth for the birthdate attribute. We simply ignore the missing values. Moreover, when presenting the description, for the categorical attributes, we also present the frequency with which that value is present in the DC.

4.8 Formatting results

The results are analysed by domain experts. The goal of this analysis is to find out whether domain experts can find interesting information in the results. To facilitate this process, we summarized the results.

4.8.1 Static CD

First, for the results of the static CD, we generated full lists of communities, containing all nodes in the communities, and a list of summaries of every community, for every partitioned HG, accompanied by a visualisation of the partitioned network. In figure

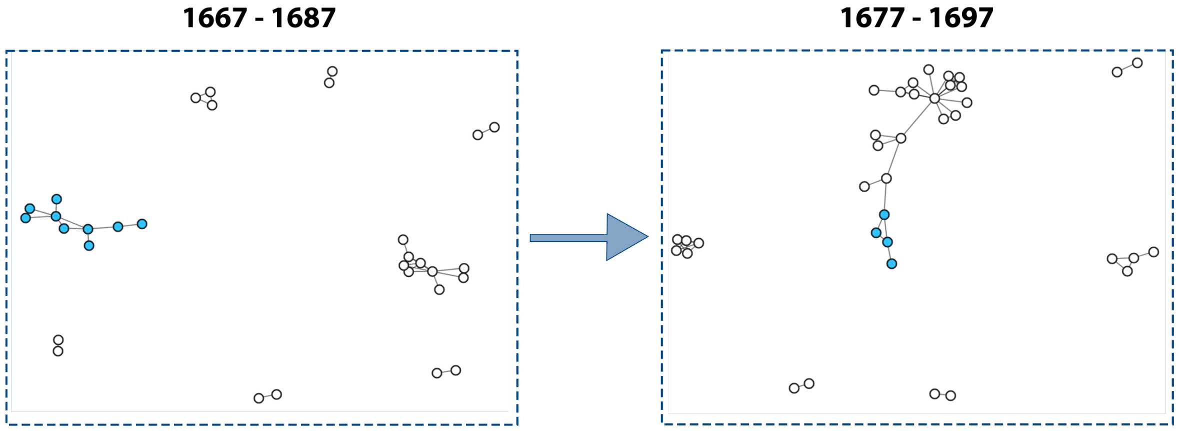


Figure 4.11: Example of DC evolution in the co-illustrator SG.

4.10, such a visualization is displayed for the co-illustrator network. Both for the full lists of communities and for the summaries, the lists of nodes are ordered based on degree centrality, which is simply calculated by determining the number of edges incident to a node and normalizing that by the maximum possible degree a node can have in the HG. More precisely, the degree centrality of a node v is calculated as follows:

$$C_{degree}(v, G) = \frac{k_v}{n_G - 1} \tag{4.5}$$

where n_G is the number of nodes in graph G and k_v is the degree of node v. For summaries, a top 5 of the nodes in the community is given, which is based on the degree centrality. Furthermore, the number of people in the community is also shown.

4.8.2 Dynamic CD

Presenting the dynamic CD results is more difficult, as there is much more information to be presented. For every DC, there is not just a list of nodes in that DC, but also a community evolution timeline. Furthermore, the list of DCs can be long, containing 74 DCs for the co-illustrator SG up to 494 for the co-acip SG. A first way to reduce the search space is to remove DCs that only exist in one interval. The goal of dynamic CD is to follow communities through time and if a DC is only present in one interval, this is not possible, and the DC is deemed uninteresting. After removing DCs of length 1, the number of DCs for the co-illustrator SG drops from 74 to 67. For the co-publisher SG it drops from 231 to 183 and for the co-acip SG, it drops from 494 to 329.

Even though we reduced the number of DCs in the results, it is still too much information to just present all at once. To solve this problem, we created two search functions, so that the results can be interactively analysed. The first search function lets the user enter a name. Then, this person can be followed through time by finding all DCs that it has been part of. A person can only be part of one DC in every interval, so this list gives a timeline of DC memberships of the person. For every interval, a small visualisation of the partition in that interval is given, where only the nodes that belong to the DC that the person belongs to are coloured. Figure 4.11 gives an example of this visualization for two intervals of a DC in the co-illustrator SG. Furthermore, a list of all people that are part of the DC in that interval is given, ordered based on degree centrality. Lastly, if any community events occurred in that interval, this is presented to the user, along with the DCs that were part of the community event. So, if a merge occurred, the DCs that were merged to create the DC that the person is part of, are presented by listing the unique IDs of the DCs and the names of all people in the DCs. For a split, the DC that was split from and the other DCs emerging from the split are presented in a similar manner.

The second search function allows the user to search based on DC IDs. Given an ID of a DC in the SG, the full timeline of that DC is returned. This is done in a similar manner as with the other search function. For every interval, a visualization is given, and a list of nodes is presented. If community events have occurred, these are also presented in the same manner.

One way of using this system, and the idea that it was based upon, is that domain experts start with searching names that they are interested in. Then, through analysing the evolution of the DC memberships of this person, they may come across DCs that seem interesting. Then, they can look at the DC timeline of those DCs by using the other search function.

Chapter 5

Evaluation

In this chapter, we evaluate the results of the static and dynamic CD in two ways. First, by using internal evaluation measures. Second, we externally evaluated the dynamic CD by having domain experts fill in a questionnaire while presenting them with the results of dynamic CD applied to the co-illustrator network, which were formatted as was discussed in section 4.8.2. The co-illustrator network was chosen for this purpose, because the co-illustrator meta-path has a very specific semantic definition, which gives a clear context to the results, facilitating the domain experts' analysis.

5.1 Internal Evaluation

5.1.1 Static CD

Tables 4.2, 4.3 and 4.4 were previously used to choose which of the tested static CD algorithms performed best on the HGs, by presenting a number of evaluation measures for the resulting partitions (see section 4.6.5). The conclusion of this comparison was that the Louvain algorithm performed best. Now, we look at these evaluation measures for the partitions generated by the Louvain algorithm, to determine whether there is inherent community structure in the data and whether the Louvain algorithm finds it.

All three partitions have a modularity value higher than 0.5, the partition of the co-illustrator network even reaching a modularity of 0.714 for the full network. All networks also have a very high value for average embeddedness, the minimum being 0.958. This is the result of the networks containing many small disjoint groups of nodes. We can look at the evaluation measures for the LCCs to investigate to what extent this property of the graphs influences the values of the evaluation measures. Looking at the LCCs, the modularity and average embeddedness remain high, the minimum of the modularity of the LCCs of all three HGs being 0.539. The average embeddedness for the LCCs of the HGs stays above 0.858 for all HGs. These values are a bit lower than those for the full networks, but this was to be expected. The fact that they do not drop by a large amount indicates that the structure that is found in the HGs is not a result of the disjoint components in the graph, at least not according to these evaluation measures. In other words, these components seem to have a small impact on the modularity and the average embeddedness of the HGs. The same also goes for

Z-modularity, which, likewise, is not much lower for the LCCs of the HGs compared to the full networks.

The conductance and the cut ratio are both measures that are based on the number of edges going out of the communities. For the full networks, these values are very small, the cut ratio often being 0. The highest cut ratio is found for the partition of the co-illustrator network, only being 0.002. The conductance stays below 0.066. For the LCCs of the networks, the cut ratio and the conductance also take on slightly worse values. The difference is bigger than for the other evaluation measures, but the values for the LCCs still suggest that there is community structure in the partition.

5.1.2 Dynamic CD

Table 4.5 was used to look at the difference in the mean modularities of partitions generated by the different Louvain variations. Tables 4.6, 4.7 and 4.8 display various similarity measures for the SGs. The SGs that were used for all these evaluations were created with an interval size of 20 and a shift of 10. We can look at the mean modularity values and the similarity measures for the partitions created by the LIP algorithm to internally evaluate whether the resulting dynamic CD method captures any temporal community structure in the data.

The mean modularity values for all three SGs are higher than 0.5, the highest one being for the co-acip network, which is 0.659. Tables 4.2, 4.3 and 4.4 display the modularity values for partitions of the HGs, partitioned by the Louvain algorithm. The mean modularity for the co-illustrator SG is 0.541, as opposed to 0.714 for the partitioned HG. For the co-publisher and the co-acip SGs, the mean modularity is even higher than that of the partitions of their respective HGs, namely 0.605 vs 0.550 and 0.659 vs 0.633, respectively. These values indicate that the community structure that was found by partitioning the HG is not lost when transforming the HG to an SG and performing CD on every snapshot of that SG.

As was discussed in section 4.6.5, the partitions of the SGs seem to be fairly stable, based on the fairly high values of the stability measures. This, in combination with the high modularity values, seems to indicate that community structure is found in the SG and that the selected method is able to track that community structure through time.

5.2 External Evaluation

Appendix A contains the questionnaire that was used. The goal of the questionnaire was to evaluate four aspects of the results, namely:

- The communities found by the algorithm.
- The evolution of those communities.
- The meaningfulness of the generated descriptions.

- The general usefulness of the system.

The questionnaire was filled in by two historians and one data expert who is deeply familiar with the dataset. As was previously mentioned, the co-illustrator SG was used, created using an interval size of 20 and a shift of 10. A θ of 0.3 was used for the matching process. The questionnaire consists of three main parts. The first one instructs the respondents to enter three DC IDs and answer questions about the community structure, the DC timeline, and the descriptions of those DCs. The DC IDs that they had to use were determined based on the length and the size of the DCs. We selected DCs that had a higher-than-average length and that contained a sufficient number of nodes. The second part focuses on the name-based search function. The respondents are instructed to enter a name of a person they deem interesting and then answer some questions about the evolution of that person through the DCs it's part of and the centrality-based position of the person in those DCs. Finally, some general questions are asked about the usefulness of the system.

The results of the questionnaires are then analysed by grouping the answers and selecting the most important and most interesting remarks and then summarizing them. In some cases, a practical example is discussed by the respondents, which can be very helpful in understanding the results and the conclusions that the respondents draw from the results. Hence, these examples are also included.

5.2.1 Community structure

The first question that was asked was whether the combination of names in the communities makes sense. In some cases, the respondents could pinpoint why a community is composed as it is. For example, one answer we received to the question of why a community is composed as it is, was that it is a community of engravers/cartographers that were active in Amsterdam. However, this is a very general description, especially given that Amsterdam is over-represented in the data. There were also DCs where the respondents did not know why it was composed as such. In these cases, they voiced that more information about the DCs might have helped them to figure it out.

Displaying the degree centralities of the nodes in the communities seems to have merit. However, they do not represent the real world perfectly. In some cases, respondents pointed out that they found that the centralities of certain nodes should be higher or lower. On the other hand, two respondents pointed out that Romeyn de Hooghe, which according to them was a central figure, also has high centrality values in the results, confirming their assumption. In one case, using the search function, the person Bernard Picard was searched by a respondent. This person was present in four intervals. In the first interval, he had the lowest centrality in the community. In the subsequent intervals, he had the highest centrality in the communities he was in. The respondent explained this: Picard becomes a central figure in the network of illustrators of Amsterdam, when he moves to Amsterdam, which was around 1711, which was at the start of the second interval (1707-1727). Apart from this, he also moves to a different DC in interval 2, which, given the respondents explanation, makes sense. This observation validates the centrality-based positioning and the community evolution of Bernard Picard.

A remark that was made a few times was that a person in a community had already died. According to a respondent that pointed this out, this is probably the result of a work of that person being published after their death.

5.2.2 Community evolution

Regarding the evolution of communities, we asked whether the starting points and ending points of the DCs made sense and if the respondents knew why the DCs had these starting and ending points. In most cases, the answer to the first question was yes. The starting and ending points seem to correspond with the years in which the people in the community lived.

Expanding on the example of Bernard Picard, according to the same respondent that entered Picards name in the search function, Picard had a pupil, Frans van Bleyswyck. As the respondent expected, they can be seen together in Picards third and fourth interval in the SG. After the second interval, Picard becomes a member of a different DC, after which he remains in the same DC for the fourth interval. This master-pupil relation perfectly explains their collective presence in the same DC for Picards third and fourth interval in the SG. Based on this observation, the respondent posed the idea that it might be interesting to look at master/pupil relations by looking at how often they coexist in the same DC.

Another observation that was made, again relating to Picard, was that in the first DC he exists in, Joseph Mulder and Matthys Pool are present, being first and second in terms of centrality. According to the respondent, these two people have made drawings inspired by a drawing of Picard. However, the respondent pointed out that these drawings were from a later date, so there must be a reason or some shared attributes that causes them to be (the most central nodes) in the same DC so early. This is an interesting question that arises from this observation and that needs more investigation, according to the respondent.

5.2.3 Community descriptions

According to one respondent, the descriptions seem to have merit, only when the frequency of the values of the attributes are high. Furthermore, Amsterdam is frequently seen in descriptions. However, as a respondent pointed out, Amsterdam is over-represented in the data. Another respondent mentioned that religion is an important attribute. However, there is some missing data for this attribute, which reduces its usefulness.

5.2.4 General remarks

In general, there is consensus among the domain experts that the method is useful as a tool for exploration. The results they have seen so far seem valid and some interesting questions have also risen from them. In essence, the results for people that are well known seem to be in line with what is expected. The results for people that are less well known give a first impression about the person's collaboration network and can give rise to interesting ideas for further investigation on that person. There are multiple observations that indicate that the system-generated evolution of communities is correct, such as the observations made when analysing Bernard Picards evolution in the network.

5.2.5 Suggested improvements

Betweenness centrality

One respondent suggested that it would be useful to look at nodes that play an important role in connecting two communities or even perhaps connecting sets of nodes within a community. This could be analysed using betweenness centrality. Betweenness centrality is a centrality measure that calculates, for a given node, the number of shortest paths between pairs of nodes in the graph that go through that node. It can be used to measure the amount of information that flows through a node. Given two communities, one can search all nodes that are not part of those communities, to find the node with the highest betweenness centrality, where only the shortest paths between pairs of nodes from the two communities are considered. This could reveal, as the respondent suggested, nodes that are important in connecting two communities.

Descriptions

According to the respondents, the descriptions sometimes seemed to be quite trivial. One way of improving these descriptions would be to improve the data quality, as one respondent pointed out. This could be done by reducing the amount of missing data, for example. As was pointed out by multiple respondents, religion is one of the most interesting, if not the most interesting attribute in the descriptions. However, the religion attribute is not available for many nodes in the networks, as can be seen in table 4.13. Improving the availability of the attribute could increase its semantic value in the descriptions.

A different way to improve the data quality is to add more attributes or group attributes in a more sophisticated manner. For example, one respondent mentioned that the family relation attribute can be improved by not only using first-degree relatives, but also second-degree (siblings) and fifth-degree (cousins) relatives. He pointed out that these are more important when analysing cooperative relationships.

Timeline characteristics

There was a desire to analyse the data from a different perspective, a more general perspective. To do this, the suggestion was made to also look at characteristics of the DC timeline, instead of focusing on the composition of DCs. The respondents wanted to be able to see appearances and deaths of DCs in the timeline or see how many DCs are active at a given time. According to one respondent, the number of DCs that are active simultaneously can say something about the competitiveness of a market. Elaborating on this, the respondent remarked that if, at a given time, many DCs are active, this could mean that there is a competitive market at that time, which would result in a varied supply and accelerated innovation in the market. Having a more top-down view of the DC timeline could thus assist domain experts in analysing the data in this way, considering hypotheses such as this one.

Chapter 6

Conclusion

The goal of this research was to develop a method for community detection in knowledge graphs. This was done in order to help the Golden Agents project gain interesting insights from the historical data they are analysing. We have achieved this by addressing a number of research questions, the first one being:

Research Question 1. Is it possible to find interesting clusters in historical data by performing community detection on KGs?

To address this research question, we first structured the data in a way that facilitates the application of community detection methods. This was done initially by structuring the KGs in a way that adheres to a more formal definition of networks, specifically the HIN data structure. Then, using meta-paths, the HIN was transformed to multiple homogeneous graphs. This allowed us to use many of the existing static community detection methods. A number of widely used and promising static community detection methods were tested and we concluded that the Louvain algorithm is the best performing method for this dataset.

The second research question was:

Research Question 2. Is it possible to track the evolution of communities in the network through time?

To address this, we additionally performed dynamic community detection by first transforming the HIN to snapshot graphs, after which we performed static community detection on every snapshot and employed a matching algorithm to create a set of dynamic communities and detect community events.

Finally, we addressed the following research question:

Research Question 3. Is it possible to find descriptions of clusters based on node attributes to support the interpretation and validation of the clusters?

We generated the community descriptions by using the most frequently observed values of the categorical attributes, and the median of the numerical attribute.

We then summarized the results and evaluated them by looking at internal evaluation measures and by performing external evaluation, using the domain experts' opinions. From the internal evaluation, we concluded that there is community structure in the data and that the selected method is able to find that structure. Concerning the external evaluation, the consensus among the domain experts was that the results they have seen are valid, and that they consider the method a useful tool for exploration.

6.1 Future Work

To conclude, we consider a number of research avenues that may be pursued to improve the methods we propose in this thesis.

6.1.1 Refining community descriptions

Improving the interpretability of the results is a good way to improve their usefulness. In order to do this, it would be interesting to refine the descriptions of the communities. One way to do this would be to simply improve the data quality, as was discussed before. It would also be interesting to employ a different, more advanced, method for generating these descriptions. One could use a decision tree algorithm, such as CART [Breiman et al., 2017], where the classes are the community IDs and the features are the attributes of the nodes. Then, the descriptions of the communities could be generated by taking a path down the tree to every leaf of the tree and taking the union of all rules that are encountered in that path. That set of rules could then be used as a description for the community ID that is classified in the leaf that the path results in. This method could find distinguishing characteristics of communities, while the method we use in this research uses the most frequently observed attribute values.

6.1.2 Adding community metadata

In the external evaluation, we received the request for more information to go along with the communities, more than once. It would be interesting to explore what information we could further extract from the partitions as they are generated, in order to facilitate the process of interpreting the communities. An example of such metadata would be the idea of finding nodes that act as bridges between communities, using their betweenness centrality. Betweenness centrality is a centrality measure that can be used to measure the amount of information that flows through a node. For any pair of communities, one can find the node that lies between these communities that has the highest betweenness centrality. This method could reveal nodes that are important in connecting communities.

6.1.3 Timeline characteristics

Lastly, an interesting, and very natural, follow-up research avenue would be to look at the results of the dynamic community detection from a more general perspective. One can look at the number of DCs that are active simultaneously, for example. According to the historians, this could say something about the competitiveness of the market, or the rate of innovation in the market.

Bibliography

[Goo, 2012] (2012). Knowledge graphs. https://blog.google/products/search/introducing-knowledge-graph-things-not/. Accessed: 2022-04-25.

[RDF, 2014] (2014). RDF specification. https://www.w3.org/TR/rdf11-concepts/. Accessed: 2021-11-03.

[Gol, 2021] (2021). Golden agents. https://www.goldenagents.org. Accessed: 2021-12-15.

[Airoldi et al., 2008] Airoldi, E. M., Blei, D., Fienberg, S., and Xing, E. (2008). Mixed membership stochastic blockmodels. *Advances in neural information processing systems*, 21.

[Alvari et al., 2014] Alvari, H., Hajibagheri, A., and Sukthankar, G. (2014). Community detection in dynamic social networks: A game-theoretic approach. In *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*, pages 101–107. IEEE.

[Aynaud and Guillaume, 2010] Aynaud, T. and Guillaume, J.-L. (2010). Static community detection algorithms for evolving networks. In *8th International symposium on modeling and optimization in mobile, Ad Hoc, and wireless networks*, pages 513–519. IEEE.

[Aynaud and Guillaume, 2011] Aynaud, T. and Guillaume, J.-L. (2011). Multi-step community detection and hierarchical time segmentation in evolving networks. In *Proceedings of the 5th SNA-KDD workshop*, volume 11.

[Barabási, 2016] Barabási, A.-L. (2016). *Network Science*. Cambridge University Press, 1 edition.

[Blondel et al., 2008] Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008.

[Breiman et al., 2017] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (2017). *Classification and regression trees*. Routledge.

[Chen et al., 2010] Chen, Z., Wilson, K. A., Jin, Y., Hendrix, W., and Samatova, N. F. (2010). Detecting and tracking community dynamics in evolutionary networks. In *2010 IEEE International Conference on Data Mining Workshops*, pages 318–327. IEEE.

[Clauset et al., 2004] Clauset, A., Newman, M. E., and Moore, C. (2004). Finding community structure in very large networks. *Physical review E*, 70(6):066111.

[Derényi et al., 2005] Derényi, I., Palla, G., and Vicsek, T. (2005). Clique percolation in random networks. *Physical review letters*, 94(16):160202.

[Dong et al., 2017] Dong, Y., Chawla, N. V., and Swami, A. (2017). metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 135–144.

[Folino and Pizzuti, 2010] Folino, F. and Pizzuti, C. (2010). Multiobjective evolutionary community detection for dynamic networks. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 535–536.

[Fortunato and Barthelemy, 2007] Fortunato, S. and Barthelemy, M. (2007). Resolution limit in community detection. *Proceedings of the national academy of sciences*, 104(1):36–41.

[Girvan and Newman, 2002] Girvan, M. and Newman, M. E. (2002). Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826.

[Greene et al., 2010] Greene, D., Doyle, D., and Cunningham, P. (2010). Tracking the evolution of communities in dynamic social networks. In *2010 international conference on advances in social networks analysis and mining*, pages 176–183. IEEE.

[Grover and Leskovec, 2016] Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864.

[Hogan et al., 2021] Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., Melo, G. D., Gutierrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., Neumaier, S., et al. (2021). Knowledge graphs. *ACM Computing Surveys (CSUR)*, 54(4):1–37.

[Hopcroft et al., 2004] Hopcroft, J., Khan, O., Kulis, B., and Selman, B. (2004). Tracking evolving communities in large linked networks. *Proceedings of the National Academy of Sciences*, 101(suppl 1):5249–5253.

[İlhan and Öğüdücü, 2015] İlhan, N. and Öğüdücü, Ş. G. (2015). Predicting community evolution based on time series modeling. In *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 1509–1516. IEEE.

[Jdidia et al., 2007] Jdidia, M. B., Robardet, C., and Fleury, E. (2007). Communities detection and analysis of their dynamics in collaborative networks. In *2007 2nd International Conference on Digital Information Management*, volume 2, pages 744–749. IEEE.

[Lee and Wilkinson, 2019] Lee, C. and Wilkinson, D. J. (2019). A review of stochastic block models and extensions for graph clustering. *Applied Network Science*, 4(1):1–50.

[Li et al., 2014] Li, M., Lu, Y., Wang, J., Wu, F.-X., and Pan, Y. (2014). A topology potential-based method for identifying essential proteins from ppi networks. *IEEE/ACM transactions on computational biology and bioinformatics*, 12(2):372–383.

[Li et al., 2019] Li, X., Kao, B., Ren, Z., and Yin, D. (2019). Spectral clustering in heterogeneous information networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4221–4228.

[Li et al., 2017] Li, X., Wu, Y., Ester, M., Kao, B., Wang, X., and Zheng, Y. (2017). Semi-supervised clustering in attributed heterogeneous information networks. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1621–1629.

[Liu and Li, 2021] Liu, D. and Li, L. (2021). A label propagation based node clustering algorithm in heterogeneous information networks. *IEEE Access*, 9:132631–132640.

[Liu et al., 2020] Liu, D., Li, L., and Ma, Z. (2020). A community detection algorithm for heterogeneous information networks. *IEEE Access*, 8:195655–195663.

[Luce and Perry, 1949] Luce, R. D. and Perry, A. D. (1949). A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116.

[Michail, 2016] Michail, O. (2016). An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280.

[Miller, 1995] Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

[Miyauchi and Kawase, 2016] Miyauchi, A. and Kawase, Y. (2016). Z-score-based modularity for community detection in networks. *PloS one*, 11(1):e0147805.

[Padgett and Ansell, 1993] Padgett, J. F. and Ansell, C. K. (1993). Robust action and the rise of the medici, 1400-1434. *American journal of sociology*, 98(6):1259–1319.

[Palla et al., 2007] Palla, G., Barabási, A.-L., and Vicsek, T. (2007). Quantifying social group evolution. *Nature*, 446(7136):664–667.

[Pizzuti, 2008] Pizzuti, C. (2008). Ga-net: A genetic algorithm for community detection in social networks. In *International conference on parallel problem solving from nature*, pages 1081–1090. Springer.

[Pons and Latapy, 2005] Pons, P. and Latapy, M. (2005). Computing communities in large networks using random walks. In *International symposium on computer and information sciences*, pages 284–293. Springer.

[Raghavan et al., 2007] Raghavan, U. N., Albert, R., and Kumara, S. (2007). Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76(3):036106.

[Rossetti and Cazabet, 2018] Rossetti, G. and Cazabet, R. (2018). Community discovery in dynamic networks: a survey. *ACM computing surveys (CSUR)*, 51(2):1–37.

[Rossetti et al., 2017] Rossetti, G., Pappalardo, L., Pedreschi, D., and Giannotti, F. (2017). Tiles: an online algorithm for community discovery in dynamic social networks. *Machine Learning*, 106(8):1213–1241.

[Rosvall and Bergstrom, 2008] Rosvall, M. and Bergstrom, C. T. (2008). Maps of random walks on complex networks reveal community structure. *Proceedings of the national academy of sciences*, 105(4):1118–1123.

[Schneider, 1973] Schneider, E. W. (1973). Course modularization applied: The interface system and its implications for sequence control and data analysis.

[Sun et al., 2011] Sun, Y., Han, J., Yan, X., Yu, P. S., and Wu, T. (2011). Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, 4(11):992–1003.

[Sun et al., 2009] Sun, Y., Han, J., Zhao, P., Yin, Z., Cheng, H., and Wu, T. (2009). Rankclus: integrating clustering with ranking for heterogeneous information network analysis. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pages 565–576.

[Sun et al., 2013] Sun, Y., Norick, B., Han, J., Yan, X., Yu, P. S., and Yu, X. (2013). Pathselclus: Integrating meta-path selection with user-guided object clustering in heterogeneous information networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 7(3):1–23.

[Wang et al., 2008] Wang, Y., Wu, B., and Pei, X. (2008). Commtracker: A core-based algorithm of tracking community evolution. In *International Conference on Advanced Data Mining and Applications*, pages 229–240. Springer.

[Wang et al., 2018] Wang, Z., Li, Z., Yuan, G., Sun, Y., Rui, X., and Xiang, X. (2018). Tracking the evolution of overlapping communities in dynamic social networks. *Knowledge-Based Systems*, 157:81–97.

[Wei et al., 2021] Wei, S., Yu, G., Wang, J., Domeniconi, C., and Zhang, X. (2021). Multiple clusterings of heterogeneous information networks. *Machine Learning*, pages 1–22.

[Yang and Leskovec, 2013] Yang, J. and Leskovec, J. (2013). Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 587–596.

[Zhang et al., 2021] Zhang, B., Gong, M., Huang, J., and Ma, X. (2021). Clustering heterogeneous information network by joint graph embedding and nonnegative matrix factorization. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(4):1–25.

[Zhuang et al., 2019] Zhuang, D., Chang, J. M., and Li, M. (2019). Dynamo: Dynamic community detection by incrementally maximizing modularity. *IEEE Transactions on Knowledge and Data Engineering*, 33(5):1934–1945.

Appendix A

Questionnaire

The purpose of this questionnaire is to evaluate the results that the dynamic community detection produces. Three aspects are discussed, namely the presence of community structure in the results, the evolution of the communities and the correctness of the generated descriptions.

A.1 Search by dynamic community ID

In cell 5, under the heading: "RESULTS: follow a dynamic community based on the dynamic community id, edit "dyn_com_id", it is possible to look up a specific dynamic community and see information about it. A dynamic community is presented as a list of communities that are connected to each other over time. Each community in this list describes the composition of the dynamic community in that period. A dynamic community has a start year and an end year. Multiple dynamic communities can be merged into one dynamic community and a dynamic community can also split into multiple dynamic communities.

You can look up a dynamic community by changing the variable "dynamic_com_id". In the output of the cell you can see how long the community has existed, a small visualization of it and the people in the dynamic community in each period in which it exists. If any merges or splits have occurred, that will also be displayed.

Answer the following question for the following three dynamic communities: *c1, c2, c3*. Enter these numbers into "dynamic_com_id" and run the cell to see the results for that dynamic community.

A.1.1 Community structure

- The people shown for each interval of the dynamic community are sorted by centrality. This is based on the number of connections a person has with other people in the network. Does this ranking correspond to the importance attributed to this person by historians?
- Do you have an idea about why the dynamic community is composed in this way?

- Could you give a description of the dynamic community based on characteristics of the dynamic community?

A.1.2 Evolution of communities

- The dynamic community has a starting point and an ending point. These years give indications and are not exact starting points or ending points. Can you explain why the community has this starting point and ending point, considering the makeup of the dynamic community?
- Have communities been merged? If so, could you describe why you think this happened?

A.1.3 Description

If you run the last cell, under the heading: "RESULTS: produce a description of a dynamic community, edit "dyn_com_id" ", you will see a description of the dynamic community based on some characteristics that are common in the community. The percentage of community members who have this attribute is also included.

- Do you think the description is a characteristic description for the community, or could the given description just as well refer to another community?

A.2 Search by name

With the search function you can enter a name and follow that person through time. You can see which dynamic communities that person belonged to and in what time interval that was. You'll also see a list of names of all the other people who belonged to that dynamic community. This list of names is also sorted by centrality, with the most central person (the person with the most connections to other people) at the top.

Enter in cell 6, under the heading: "RESULTS: follow a person through time" , for "name", a name of an important person, a person who had a central role. Also write down what name this is below. Then answer the following questions.

A.2.1 Centrality

- What position does the person have in the list of people in the community? Does this match what you expected?

A.2.2 Evolution of community membership

- Can the changes taking place in the person's membership of the dynamic communities be substantiated? If so, could you provide this justification?
- Did seeing this development lead to interesting insights about this person?

A.3 General

These questions are about the results in a general sense.

- Did you gain certain insights about the data by using the system? If so, can you name them?
- Do you think this system can be useful for historical research? If so, in answering what type of research questions could the system be helpful? Please provide a concrete example, if possible.