University of Applied Sciences Würzburg-Schweinfurt
Faculty of Computer Science and Business Information Systems

Bachelor Thesis

Detection and Analysis of Communities in a large-scale bibliographic Graph Database

submitted to the University of Applied Sciences Würzburg-Schweinfurt in the Faculty of Computer Science and Business Information Systems to achieve the Bachelor of Engineering degree in Computer Science

Simon Weickert

Submitted on: September 14, 2021

First Reader: Prof. Dr. Frank Michael Schleif
Second Reader: M. Sc. Maximilian Münch

Zusammenfassung

Diese Arbeit befasst sich mit verschiedenen Ansätzen zur Handhabung großer graphenbasierter Datensammlungen anhand der bibliografischen Datenbank DBLP. Diese Ansätze zielen darauf ab, die Daten zu reduzieren. Eine Reduktion findet durch die Spezialisierung auf die Top 200 Konferenzen und die dazugehörigen Artikel statt. Der Louvain Algorithmus wird angewandt, um Communitystrukturen in den Daten zu ermitteln. Dadurch wird eine weitergehende Fokussierung auf die Top 10 Communities ermöglicht.

Diese Arbeit stellt überdies einen Ansatz vor, mit welchem sich repräsentative Artikel der Communities ermitteln lassen. Die Extraktion dieser Artikel ist durch die generierte „Dokument zu Thema"- Wahrscheinlichkeitsverteilung θ_d aus dem Latent Dirichlet Allocation Modell möglich. Dieser Ansatz erlaubt es, je nach Anzahl der Themen und der Festlegung der repräsentativen Datenmenge eine signifikante Reduzierung derselben zu erreichen. Somit lassen sich anschließende Analysen leichter durchführen und Visualisierungen können aufgrund der geringeren Datendichte besser verstanden werden. Dabei wird angenommen, dass der Informationsgehalt der Daten stabil bleibt, solange die latenten Themen alle wichtigen Bereiche einer Community abdecken. Es sind jedoch weitere Forschungen notwendig, um diese Annahme validieren zu können.

Des Weiteren kommen deskriptive Visualisierungen zum Einsatz, um einen Überblick über die repräsentativen Daten herzustellen. Das Ende der Arbeit bildet die Überführung der einzelnen Titel mithilfe einer Kombination aus fastText Embedding und einer Gewichtung durch die Inverse Dokumentenhäufigkeit in einen hochdimensionalen Vektorraum. Die Darstellung in einem 2D und 3D Streudiagramm ist durch den t-SNE Algorithmus möglich und zeigt, wie die Titel im Kontext zueinander stehen.

Abstract

This thesis addresses different approaches to handle large graphical data collections using the bibliographic database DBLP. These approaches aim at reducing the data. One reduction takes place by focusing on the top 200 conferences and those articles that were presented at one of them.

Through the Louvain algorithm distinct communities are detected which then allows to further narrow the focus on a subset such as the top 10 communities. This work moreover presents an approach to determine representative articles of the communities. Selecting these is made possible through the generated document-topic probability distribution θ_d from the Latent Dirichlet Allocation model. This approach allows to achieve a significant reduction depending on the number of topics and the specification of the representative data quantity. The implementation of this technique leads to lower data density and therefore to an easier management of continuing analyses and better understanding of visualizations. It is assumed that the information content will remain stable as long as the latent topics cover all the important areas of a community. However, to fully validate this approach more research needs to be done in the future.

Descriptive visualizations subsequently assist to give an overview of the representative data. In the final part of this work the individual titles are transformed into a high dimensional vector space using a combination of fastText embedding and weighting by inverse document frequency. The representation in a 2D and 3D scatter plot is made possible through the t-SNE dimensionality reduction algorithm and shows how the titles compare to each other.

List of Acronyms

DBLP Digital Bibliography & Library Project

GDSL Graph Data Science Library

NLP Natural Language Processing

LDA Latent Dirichlet Allocation

SNE Stochastic Neighbor Embedding

t-SNE t-distributed Stochastic Neighbor Embedding

XML Extensible Markup Language

JSON Javascript Object Notation

HTML Hypertext Markup Language

IDF Inverse Document Frequency

Contents

1 Introduction

This chapter clarifies the motivation and goal of the thesis, stating the research objectives. Subsequently, the organization of the thesis highlights the route of this work.

1.1 Motivation

Processing and analyzing large amounts of data is becoming an ever increasingly important task. The Digital Bibliography & Library Project (DBLP) dataset has huge amounts of data stored by today, as all major publications in the field of science have been stored there for over 20 years. Such large amounts of data require preparation and certain approaches to analyze and process them in a meaningful way. For this work, the data is processed in a graph structure.

1.2 Goal of the Thesis

Using the large-scale bibliographic database of the DBLP, various problems will be investigated, and possible approaches will be examined. The following three issues are the primary focus of this thesis:

- How can a large-scale bibliographic database with a graph structure be managed and separated into communities?
- How can a representative dataset of the communities be formed to reduce memory and computational requirements?
- How can the identified communities and their relations to one another be visually represented?

1.3 Organization of the Thesis

The thesis is separated into four chapters in order to provide answers to the issues mentioned above.

To begin, chapter 2 provides the theoretical foundation necessary for a solid understanding of the processes described in the subsequent chapters. Thus, the fundamentals of graphs, communities and community detection are discussed, followed by the essential aspects of the algorithms and models which are utilized in this thesis.

Subsequently chapter 3 gives an overview of the data and then covers all the necessary data preparation processes.

Eventually, chapter 4 deals with all the practical application steps for the detection of the communities and the continuing analysis.

Finally, the concluding chapter 5 highlights the major points of this work and provides an outlook for future research.

2 Theoretical Background

2.1 Graph and Graph Database

Graphs
Graphs date all the way back to 1736, when Leonhard Euler solved the popular issue of the "Seven Bridges of Königsberg". The issue posed the question of whether it was feasible to visit all four areas of a city linked by seven bridges while crossing each bridge just once. That was not the case. Euler established the foundations of graph theory and related mathematics with the idea that only the links themselves were significant. [11]

The basic building blocks of a graph are vertices and edges. Whereby a vertex reflects an entity, which can form relations called edges to other entities. Graphs can be found in all kinds of real-world things or situations like a street map, the family tree trunk, or any social media such as Facebook or Instagram. Graphs are a near-universal way of thinking about real-world events because they abstract away the items and relationships being displayed, allowing for quick and reliable interpretation of the data's connections.

A simple example of a graph is the route between two points, such as the way from home to the library where several intersections are passed. This graph example is depicted in Figure 2.1.

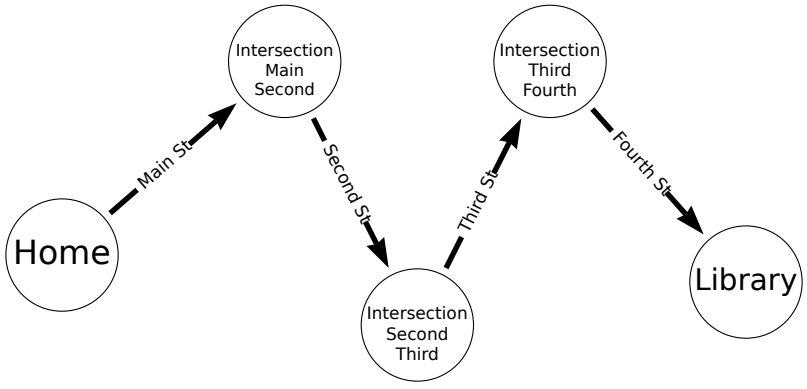

Figure 2.1: Graph depicting the path from home to the library passing multiple streets.

Graph Database

In computing, a graph database (GDB) is a database that displays and stores data using graph structures for semantic problems with vertices, edges, and properties. The graph is the system's core concept. It represents the items in the databases as a set of vertices and their relationships are the edges. These connections enable rapid querying of the nodes associated with the item. Graph databases provide natural visualization of relationships, making them both accurate and efficient for highly interconnected data. Graph databases are a subset of NoSQL data that were created to solve the shortcomings of relational databases. While the graph model demonstrates the dependency of the data nodes, the relationship model, which is compatible with various NoSQL data models, combines the data through the missing link. In other words, relationships are the graph's initial citizens, capable of being written, directed, and given structures. This contrasts with the relational patterns that define and enhance these partnerships throughout practice. The graph database features are comparable to those of the old network database models in that they both represent normal graphs; however the network database model operates at a lower level of abstraction and does not allow for simple edge crossing.

2.2 Neo4j

Neo4j is an open-source graph database managements system and is publicly accessible since 2007. By now it's become a well-known and well documented platform which provides numerous useful tools and functions in addition to storing data in graph structures. The Neo4j Desktop Application is used to manage the database management system and databases. The Neo4j Browser is used to create the queries and see the results. For even faster visualizations and overviews the Neo4j Bloom can be used. Instead of utilizing a query language, the user can select and set the desired nodes, relationships, and attributes via the user interface. Additional filter help to reduce the number of results. This allows anybody to access databases without having a strong knowledge of programming. Many different plugins, utilities, and packages can be installed through an app gallery. The Graph Data Science Playground is a useful tool that enables algorithms to be easily parametrized and executed. This tool allows quick Evaluation whether an algorithm is helpful or not.

Cypher

Cypher is a declarative query language and is used to retrieve and manipulate the data in Neo4j. It is well optimized for graphs and can handle traversal of billions of nodes on reasonably powerful hardware. Furthermore its intuitive structure makes it easy to read.

```
MATCH (p:Person)-[:LIVES_IN]->(c:City)
RETURN p AS Person
```

Code 2.1: Query that retrieves all persons that have the relationship LIVES_IN directed to a city.

The example in 2.1 demonstrates the direct graph structure in the cypher code. Persons that have the relationship `LIVES_IN` to a city are returned.

Graph Algorithms in Neo4j

Many graph algorithms are already built into Neo4j which are accessible via the Graph Data Science Library (GDSL). There are many different algorithm types such as Pathfinding & Search, Centrality, Similarity, Community Detection and Node Embedding algorithms. The scientific research is ever increasing, and new algorithms are being developed all the time. The GDSL includes around 5-8 algorithms for each field, which are utilized according to application and circumstance.

Neo4j Driver

To interact with the database, the Neo4j Driver can be used in most programming languages. The support for the most popular languages come directly from Neo4j and offer good documentation. The connection to the database is done via the so-called bolt protocol.

2.3 Community Detection

What is a Community?

The first issue in graph clustering according to Fortunato [5] is to define a community quantitatively. There is no generally recognized definition. In fact, the definition frequently varies depending on the system and/or application. However, a community may be identified easily and intuitively by the fact that the nodes within the grouping have a stronger connection than those on the outside. For a mathematical overview a Community or more generally a subgraph C of a Graph G is defined with $|C| = n_C$ vertices and $|G| = n$ vertices. The internal degree reflects connections to other vertices inside C and the external degree reflects connections to vertices outside of C. They are defined as k_v^{int} and k_v^{ext} for each vertex $v \in C$. The sum of the degrees internally and externally are k_{int}^C and k_{ext}^C respectively. Thereby all degrees of a subgraph can be summarized with $k^C = k_{int}^C + k_{ext}^C$ The density of the degrees inside a subgraph called the intra-cluster density $\delta_{int}(C)$ and the density to the outside of the community called

the inter-cluster density $\delta_{int}(C)$ are needed to finish the mathematics of the intuitive definition. The community's intra-cluster density $\delta_{int}(C)$ and the density of the total Graph $\delta(G)$ is calculated through a subtraction. The result is expected to be higher than the comparison of $\delta_{ext}(C)$ with $\delta(G)$ [5].

Communities can be easily identified in structures where relationships play an important role as in social structures. For example on social media platforms there are various closely connected groups of people who have similar interests and thus similar friends or the same hobbies listed in their profile. A community can also emerge when people attend the same institution or work in the same profession.

Community Detection

A large network can usually be sensibly divided into communities by utilizing the relationships. Community detection algorithms are used to find groups of vertices that are closely connected to one another but weakly linked to other vertices in the network. These algorithms are utilized in a wide number of areas and across a broad variety of industries and application domains in the real world. Karatas and Sahin [8] provide a summary of the different application domains, few of which are highlighted below:

- In public health, community detection algorithms help in identifying groups at risk of epidemic illness or in identifying new cancer or tumor types.
- In politics, community detection is helpful for tracking politicians' impact on social groups or for detecting bots that manipulate political beliefs.
- In criminology, community detection is used to identify frauds on telecommunication networks or to detect terrorist groups on social media.
- Recommendation systems are improved by the detection of like-minded individuals that are interested in the same products.

Algorithms

There are many different community detection techniques as shown in Figure 2.2. The two basic categories are separating the disjoint and the overlapping techniques, where the naming is determined by the resulting type of the communities, depending on whether they are completely independent and have no overlap with others or do overlap. A comprehensive overview of all the different techniques and their characteristics can be found in the review from Javed et al. [6].

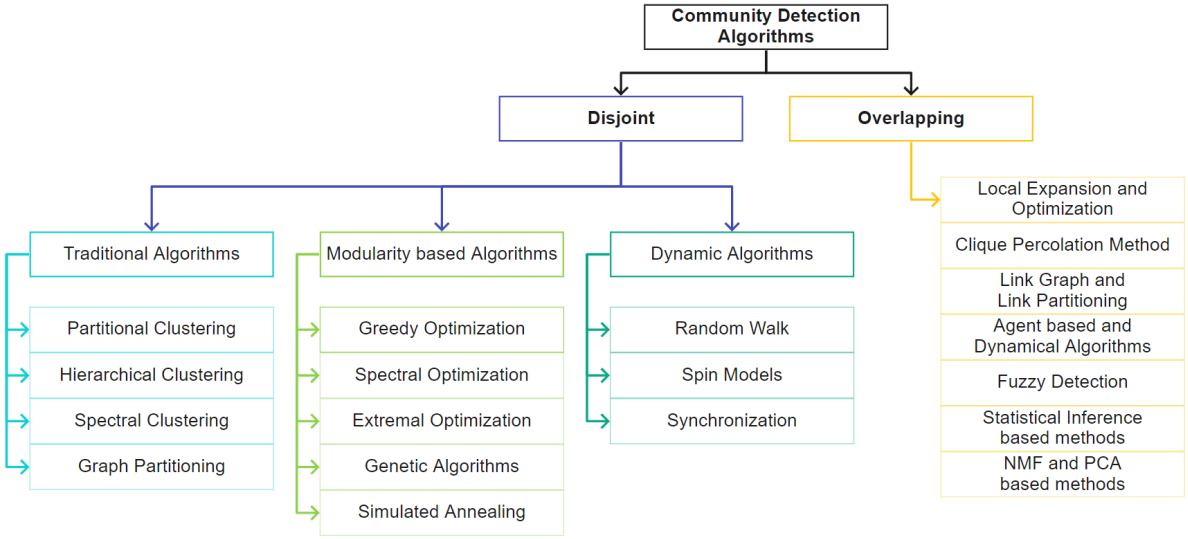

Figure 2.2: Hierarchical overview of the different community detection techniques. Derived from [6].

The Neo4j GDSL contains community detection algorithms, grouped by quality tier (see Table 2.1).

Production Quality	Beta	Alpha
Louvain	K-1 Coloring	Strongly Connected Components
Label Propagation	Modularity Optimization	
Weakly Connected Components		Speaker-Listener Label Propagation
Triangle Count		
Local Clustering Coefficient		

Table 2.1: Available community detection algorithms in the GDSL.

Modularity

Many algorithms use modularity as an objective to be maximized as it's an effective way to describe the quality of a community. The modularity score is a scalar value between -1 and 1 that reflects the density of connections within communities in comparison to the density of connections between communities [3]. Higher values indicate better connected communities inside the network. In the Equation 2.1 the Modularity Q is calculated for an undirected Graph with m edges which is represented by an adjacency matrix A. Here $A_{ij} = 1$ if there is a relationship between vertex i and vertex j and 0 if not. The degree of the vertex i is described by $k_j = \sum_j A_{ij}$. The Kronecker delta function δ_{ij} returns 1

if $i = j$ and $A_{ij} = 0$ if $i \neq j$ and thereby prevents edges from being counted twice [1].

$$\mathbf{Q} = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta_{ij} \tag{2.1}$$

Figure 2.3 displays the modularity scores of differently distributed communities in a graph. Because the scores in these examples are based on intuition rather than calculation, they serve primarily to illustrate the concept. The graph on the left displays a negative modularity score because there are fewer connections between community members compared to other communities. The second graph shows that modularity is zero when there is only one community. Because the third graph has only two communities, the likelihood of having numerous links within them is high, as demonstrated by its modularity score. The graph on the right shows a modularity score which is close to optimal.

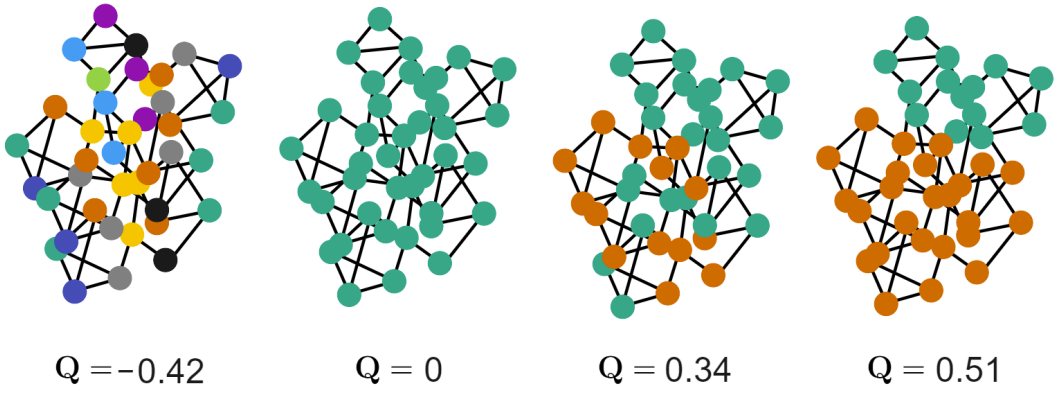

Figure 2.3: Modularity scores of differently distributed communities over the same graph structure. Each color represents a community.

To achieve maximum modularity, optimizations on the modularity function must be performed over all potential communities in order to determine the one with the highest score. Since the number of potential communities of a network is exponentially large, approximation optimization techniques are used to overcome this problem [12].

Louvain Algorithm

The Louvain algorithm is a simple, efficient, and easy way to detect communities on large networks. It has been used on a wide variety of networks and with a size of up to 100 million nodes and billions of links [3]. This method is a greedy way of trying to improve the *modularity* of network partitions. The calculation of the communities

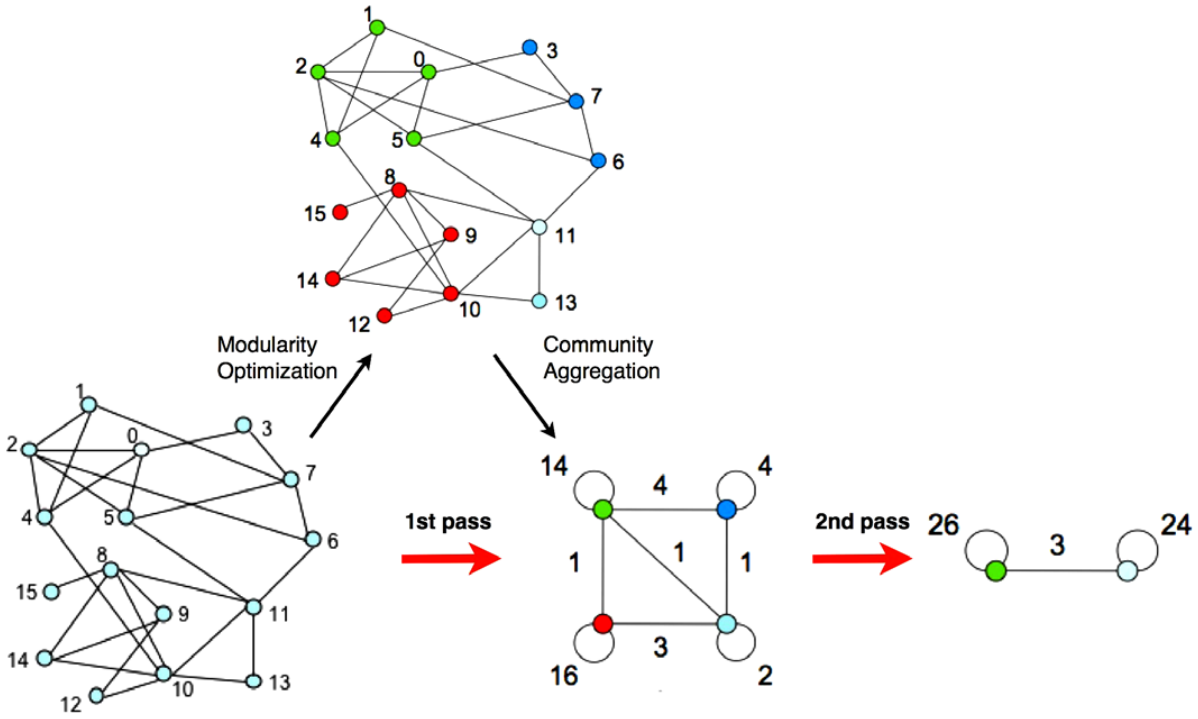

Figure 2.4: Steps of the Louvain algorithm. Source[3].

is done iteratively where each iteration contains two steps called the modularity optimization and the community aggregation (see Figure 2.4). At the start, the number of communities equals the number of vertices, which means that each vertex has its own community.

Modularity Optimization For each vertex i all its neighbors j are considered and it is evaluated whether the modularity increases or decreases when i is removed from its community and placed into the community of j. After the evaluation if a community with a better score was found, then vertex i is placed into the community where the highest increase occurred. This step stops as soon as a local maximum is reached, i.e. no more improvements can be made.

Community Aggregation In the second step every vertex of a newly formed group is combined to a single vertex representing the community.

These steps are repeated until there are no more changes, or a set maximum of iterations is reached. Louvain algorithm is a fast algorithm based on modularity community detection. For a graph that has a total of n vertices and m edges the computational time complexity of the algorithm is $\mathcal{O}(m)$ [6]. In smaller graphs there is also an average growth rate of about n. The demand of storage for large networks tends to be a bigger

limitation than the time cost [3].

Other Algorithms

To provide a more comprehensive overview of community detection methodologies in general, this section showcases two more algorithms that are also available in Neo4j's algorithm library.

Triangle Count

Triangle Count is a very basic algorithm that can find very close-knit communities for example inside a social network. As its name implies it counts the triangles in a given graph. A triangle is also known as a 3-clique in graph terminology which means that three nodes all have a relationship to each other. A primitive example can be seen in Figure 2.5, where the two triangles are highlighted.

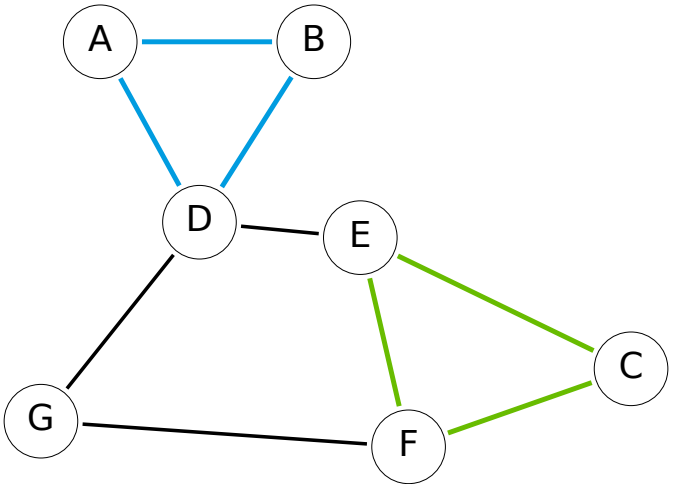

Figure 2.5: A graph displaying two triangles.

Strongly- and Weakly-Connected-Components

The Strongly-Connected-Component (SCC) and the Weakly-Connected-Component (WCC) are essentially two distinct algorithms, though their underlying concept is the same. In a given graph, connected components are searched for. With SCC, the direction of the individual links is important since node pairs are only considered connected if they are mutually linked. Whereas with WCC, the graph is always considered unweighted and thereby every connection results in an inclusion of each node pair.

Figure 2.6 shows a graph with two connected components, though the results are different for both algorithm types. The two weakly connected components are showcased in the blue circles and the strongly connected components are inside the red circles.

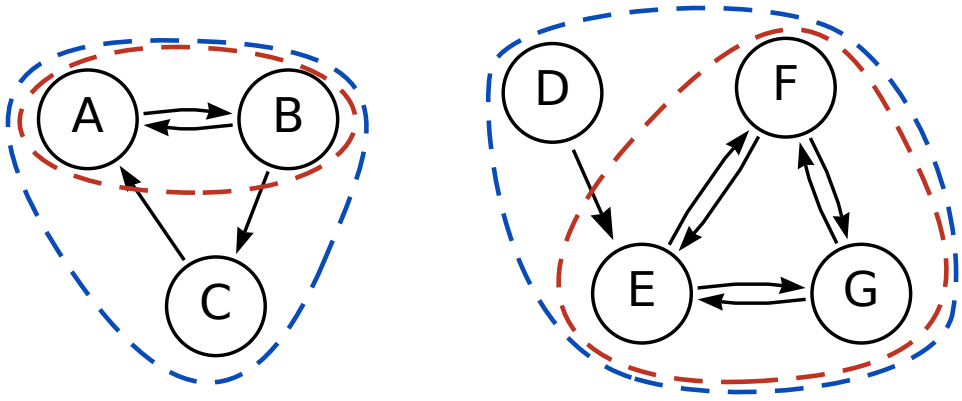

Figure 2.6: Two individual connected components the detected components by WCC are shown in blue and by SCC are shown in red.

2.4 Word Embeddings

Word embedding is a term used in Natural Language Processing (NLP) to refer to the representation of words for text analysis, typically in a vector space where the vectors of similar words are close to each other. In 2013 the popular Word2Vec model was introduced by Mikolov et al. [10] which is an unsupervised learning method that allows very large corpora to be converted into vector space. In 2017 Bojanowski et al. [4] proposed a new method called *fastText* which brought further improvements as it represents n-gram characters in the vector space instead of the whole words. This allows minor word differences to be retained better. It's also very efficient and thereby allows to quickly train models on huge corpora. To give better understanding how the n-gram representation works, a brief explanation about the subword model is given in the following paragraph. Due to time constraints further information about the appliance of the model with Skip-gram and negative sampling may be looked up directly from the article by Bojanowski et al. [4].

Subword Model The so called *subword model* is used to create a bag of character n-grams for each word. For example if $n = 3$ (3-*grams*) is used then the word *writing* will be split into <*wr, wri, rit, iti, tin, ing, ng*> (Note that angular brackets have been added to mark the beginning and ending and count as a character). Every n-gram gets a vector representation associated and each word is represented by the sum of all its n-gram vectors plus the representation for the word itself [4].

2.5 LDA

Latent Dirichlet Allocation (LDA) was first introduced in 2003 by Blei et al. [2] and they define it as "a generative probabilistic model for collections of discrete data such as text corpora" (Blei et al. [2]). It's a topic modelling technique prominently used for finding hidden structures in big data and is frequently used for a variety of NLP, text mining and information retrieval tasks. The applications in the fields e.g. medical sciences, software engineering and political science are well summarized in the work of Jelodar et al. [7]. LDA is premised on two assumptions:

1. Documents are classified by a combination of topics

2. Topics are described by a combination of words

The goal is to find the best Dirichlet parameters to determine the most accurate latent topics for each document. It models a distribution over probability densities controlled by the Dirichlet prior parameters α and β where α controls the document-topic distribution θ_d and β controls the topic-word distribution ϕ_t. Here t denotes a topic though sometimes in the literature a topic is also denoted by z.

This generative process to model a corpus D with M documents d which contain N words is explained in the following steps which are derived from the work of Jelodar et al. [7] and Blei et al. [2]:

1. Selection of a multinomial distribution ϕ_t from Dirichlet distribution with parameter β, one for each topic t

2. Selection of multinomial θ_d from Dirichlet distribution with parameter α, one for each document d

3. For each word w_i in the document:

    a. Selection of a topic t_i from multinomial θ_d

    b. Selection of a word w_i from multinomial ϕ_{ti}

The hidden structure of the topics in LDA is described by the distribution of the latent variables given by the documents: $p(\theta, t|w, \alpha, \beta) = \frac{p(\theta,t|w,\alpha,\beta)}{p(w|\alpha,\beta}$

2.6 t-SNE

The t-distributed Stochastic Neighbor Embedding (t-SNE) is a nonlinear dimensionality reduction technique that is suitable for embedding a high-dimensional dataset into a two- or three-dimensional space. This way the newly mapped data can be visualized in a 2D or 3D scatter plot.

How t-SNE works

Like the classic Stochastic Neighbor Embedding (SNE) the algorithm t-SNE also begins by transforming the high-dimensional Euclidean distances between data points to conditional probabilities representing the similarities [9]. That means that a D-dimensional dataset $X \in \mathbb{R}^D$ gets converted to a low d -dimensional embedding $Y \in \mathbb{R}^d$ which is typically 2 or 3. The goal is that if two points x_i and x_j are close to each other in the input space then their respective points y_i and y_j in the low dimension space should also be close. The conditional probability $p_{j|i}$ of point x_j to be next to point x_i is represented by a Gaussian function centered at x_i with a variance of σ_i(see Equation 2.2 for the mathematical equation) [9].

$$p_{j|i} = \frac{exp(-\|x_i - x_j\|)^2/2\sigma_i^2}{\sum_{k \neq i} exp(-\|x_i - x_k\|)^2/2\sigma_i^2} \tag{2.2}$$

Typically the conditional probabilities $p_{j|i}$ and $p_{i|j}$ are then symmetrized by first adding them and afterwards dividing them by 2 to obtain the joint probabilities p_{ij} (see Equation 2.3).

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2} \tag{2.3}$$

In the embedding space the t-SNE algorithm replaces the Gaussian distribution with its name-giving *Student's t-distribution* to solve the crowding problem and optimization problems [9]. Further explanations about the problems can be obtained through the work of van der Maaten and Hinton in their paper [9]. The joint probabilities q_{ji} of the low d-dim Space are calculated with Equation 2.4.

$$q_{ij} = \frac{(1 + \|y_i - y_k\|)^2)^{-1})}{\sum_{k \neq l}(1 + \|y_k - y_l\|)^2)^{-1}} \tag{2.4}$$

SNE now aims to minimize the difference between the two probability distributions by calculating the cost function C with the usage of the Kullback-Leibler divergence (see Equation 2.5).

$$C = KL(P \,||\, Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{2.5}$$

The cost function gives an indication about how good or bad the embedding is. To optimize this function gradient descent is performed. Van der Maaten and Hinton put

together the complete derivation of that Equation 2.6 in [9].

$$\frac{\partial C}{\partial y_i} = 4 \sum_{j \neq i} (pij - qij)(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1} \tag{2.6}$$

The algorithm combines the probability of pairing conditions and attempts to reduce the amount of probability variation by higher and lower dimension. The algorithm takes a lot of time and is hardly manageable for large real-world datasets as it needs to compute all pairwise interactions which results in a time complexity of $\mathcal{O}(N^2)$. Thereby a lot of further research on the t-SNE has been dedicated to the acceleration of this method. Van der Maaten also published another paper where he reduced the time complexity to $\mathcal{O}(N \log N)$ by applying a variation of the Barnes-Hut algorithm [16].

CRISP-DM

The CRISP-DM cycle represented in Figure 2.7 is a standardized process used to plan data mining projects. The cycle contains six steps which cover all the important aspects of a data science project. The following paragraphs explain these steps as they relate to the thesis.

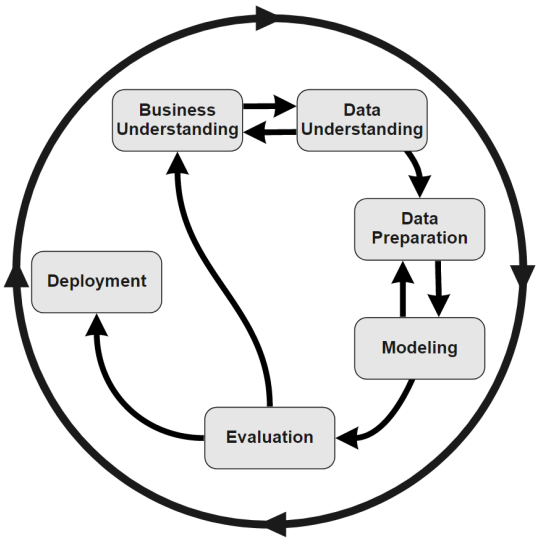

Figure 2.7: The CRISP-DM Cycle.

Business Understanding

The first stage includes collecting information about what is required to meet the professor's or employer's requirements. Therefore a preliminary analysis is the initial step in gaining an overview of this.

Data Understanding

The data understanding phase begins with initial data collecting and continues with actions aimed at familiarizing oneself with the data, identifying data quality issues, gaining early insights into the data, or detecting intriguing subsets to develop hypotheses about hidden information. There is a strong connection between Business and Data Understanding. Formulating the data mining issue and developing the project strategy both need knowledge of the available data.

Data Preparation

The data preparation phase encompasses all actions necessary to construct the final dataset that will be fed into the modeling tools from the raw data. Tasks associated with data preparation are likely to be repeated many times and in no sequence. Selecting tables, records, and characteristics, cleansing data, creating new attributes, and transforming data for modeling tools are all tasks.

Modeling

This phase involves the selection and use of different modeling methods, as well as the calibration of their parameters to their optimum values. Typically, there are many methods for solving the same kind of data mining issue. Data preparation and modeling are closely linked because new issues and discoveries often surface in this phase.

Evaluation

Prior to finalizing the model's deployment, it is important to conduct a comprehensive evaluation of the model and to examine the processes used to build the model to ensure that it meets the business objectives.

Deployment

The project typically ends when the requirements are met and the model is finished. The gathered data is structured and presented in a way that the client can make effective use of it. The acquired information is structured and presented in a way the client can use well.

3 Data And Graph Preparations

3.1 Data Overview

DBLP Dataset

The computer science bibliography of DBLP provides open bibliographic information on major computer science journals and proceedings [13] and by January 2019 it already indexed over 4.4 million publications, published by more than 2.2 million authors, and presented at about 39 thousand conferences or workshops [14]. DBLP provides its datasets as files in the Extensible Markup Language (XML) format but in this work the data has been downloaded as a prepared file in Javascript Object Notation (JSON) format which already excludes some irrelevant data and only contains articles with their properties. The Structure of an article in the JSON-file is described in Code 3.1.

```
{
   "abstract":"Abstract of the article or paper",
   "authors":[
      "Author 1",
      "Author 2"
   ],
   "n_citation":0,
   "references":[
      "ID ref Article 1",
      "ID ref Article 2"
   ],
   "title":"Title",
   "venue":"Conference/Workshop Name",
   "year":2010,
   "id":"ID Article"
}
```

Code 3.1: Structure of an article in the JSON-file of the DBLP dataset.

As seen in this example each article has the following properties that are listed in Table 3.1.

abstract	authors	n_citation	references	title	venue	year	id

Table 3.1: Properties of the article object.

Acquisition

The acquisition of the data was made possible through the work of [15]. The downloaded JSON-file versioned at V10 contains about 3 million articles, 2.5 million venues and around 1.8 million Authors. In total there are more than 37 million relationships between these nodes. Figure 3.1 provides a more detailed overview of these statistics.

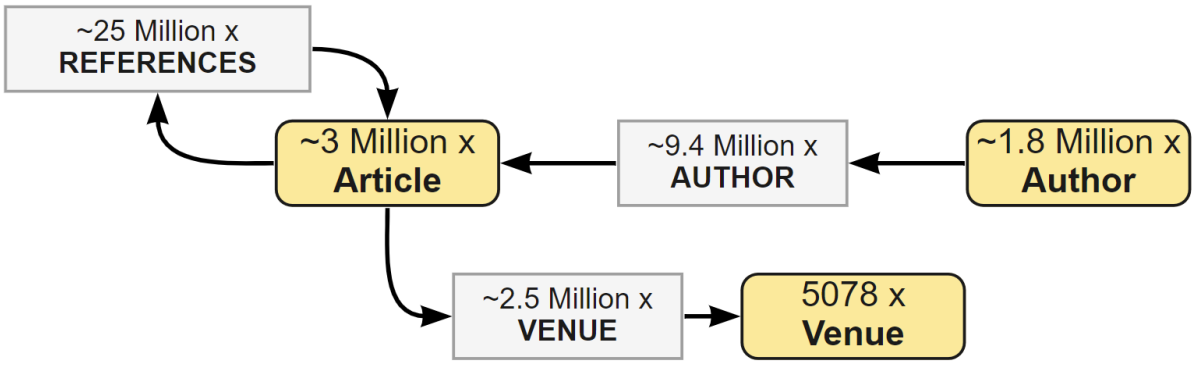

Figure 3.1: Overview of the nodes and relationships acquired.

Quality

The abstracts and titles show inconvenient characters mostly recognized as LaTeX commands. The dataset seems to reliably contain information for each property, only in the abstracts some empty values have been recognized. To see how many articles are missing, a sample set of about 424 thousand articles has been checked. In this sample set about 88% of the articles had no abstract. The cause for this missing data appears to be that some abstracts are not publicly available due to copyright restrictions.

with Abstract	without Abstract	No. of Articles	Abstract Probability
371675	52721	424396	~87,6 %

Table 3.2: Statistics about the percentage of abstracts in a sample set of articles.

3.2 Data Transformation

To generate a graph structure from the JSON-file's data, the relationships must be specified. The articles and venues are the focal point of this thesis. The relationship between them is retrieved directly since the article objects have the *venue* key. It specifies

the conference name at which the paper was presented, and the *references* key contains information about cited articles. Each entity, e.g. articles, venues and authors as well as each relationship gets their own header file, which may be thought of as the entity's blueprint. The header files regarding the articles and venues are listed in the following blocks 3.2 to 3.5.

```
index:ID(Article),title:string,abstract:string,year:int
```

Code 3.2: Article object header CSV-file.

```
name:ID(Venue)
```

Code 3.3: Venue object header CSV-file.

```
:START_ID(Article),:END_ID(Article)
```

Code 3.4: Articles to Article relationship header CSV-file

```
:START_ID(Article),:END_ID(Venue)
```

Code 3.5: Articles to Venue relationship header CSV-file.

A CSV-file is generated for each entity type using the structure from the header. The files are produced using a Python script that extracts data from JSON-files and saves it to the corresponding CSV-file. Additionally, any parentheses included inside the string values are deleted since they would have caused problems later in the import to Neo4j.

Import into Neo4j

Importing data into Neo4j can be done either with Cypher or with the Neo4j Admin Tool. When importing big files, it is recommended to use the import feature of the admin tool. The import command is then performed with the following parameters in the Neo4j Admin Shell. The *–nodes* and *–relationships* parameters get their name assigned and the path of their corresponding header file followed by the path where the generated data is located. To make sure that no duplicates are loaded into Neo4j *–skip-duplicate-nodes=true* is set and additionally *–skip-bad-relationships=true* ensures that the import finishes even if a relationship is missing one end of the connection. The import took about three minutes in total and imported 4.8 million nodes, 37 million connections, and 13.5 million properties successfully.

Graph Schema

To give an overview of the graph schema the `db.schema.visualization()` function in Neo4j can be executed. The Figure 3.2 shows a custom image of that schema which also displays the properties.

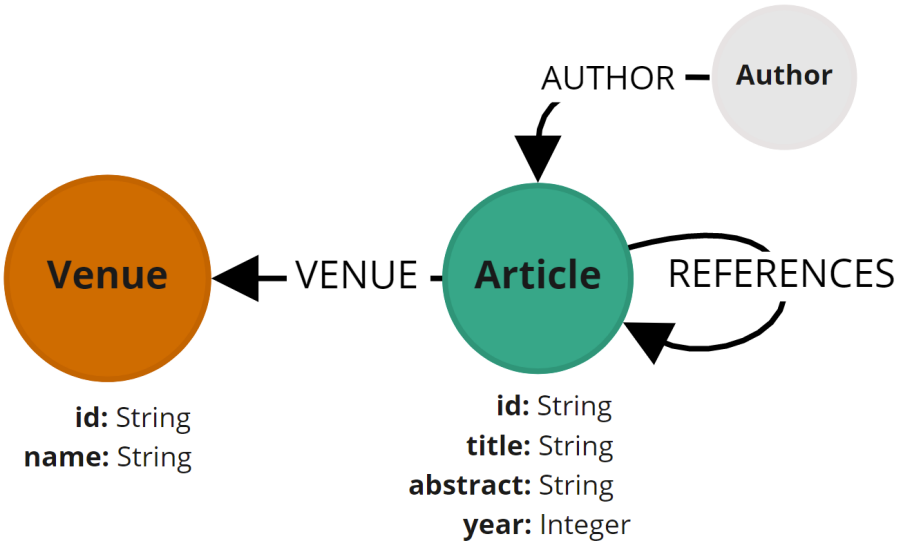

Figure 3.2: Graph schema showing the nodes with their properties and the relationships between them.

3.3 Data Reduction

Initially the attempt had been to only load a portion of the data into the database to reduce memory complexity and loading times. While significantly decreasing the size, it also resulted in a huge fall in the number of connections, since they are totally disregarded in the absence of a partner. This random destruction of the connections is not acceptable as it significantly decreases the quality of the network and therefore this particular approach was abandoned.

Reduction to top 200 Conferences

As the following step a more systematic reduction of those large amounts of data was done by setting the focus on a specific area of the dataset. Therefore the decision to only view the most important 200 conferences was made. In order to do so, the corresponding names of the conferences were needed first. These can be found on the website Guide2Research[1] which contains the most important conferences in the field

[1] website link: https://Guide2Research.com/topconf/

of computer science and electronics sorted by the h5 index and the impact of these venues.

Although these top conferences cannot be exported directly, they can be retrieved using the website's source code. Because the conference names are always saved in the same Hypertext Markup Language (HTML) tags, they can be selected sequentially using the "Find Next Match"-functionality in *Visual Studio Code*. After removing all HTML tags, the list of the names remains. This list, however, cannot be utilized directly since the names do not match precisely to those in the DBLP record. Two options emerged, which were to either program a script that would help to locate the matching names in the DBLP dataset or to manually find the corresponding venues. The second option was chosen due to optimism that locating these venues manually might be quicker. there were many differences spread inconsistently between the names of Guide2Research and the DBLP record as the following list shows.

- Acronyms as prefix
- Abbreviations in brackets
- The publisher mentioned before or after name
- Different word usages (e.g. conference $\Leftrightarrow$ symposium)
- Differences in comma placements
- Empty conference name in DBLP data
- Deprecated naming of conference in DBLP
- Acronym instead of full name used

When no matching name was located after all the alterations, there were still other ways to discover them. The following steps described in Figure 3.3 were taken for this purpose. When the names in the two sources did not match, the venue-specific XML-file was examined to check whether the proper name was declared there. If not, a look into the proceedings of a venue sometimes help to reveal deprecated names, and if that did not work, a peek into the website was the last chance to find any insights. If nothing worked, then that conference was disregarded.

In total there are 52 conference names that could not be found in the DBLP dataset and therefore will be ignored in the oncoming work. Table 3.3 shows the distribution of how many conference names were not matched in intervals of 25. For instance, just one

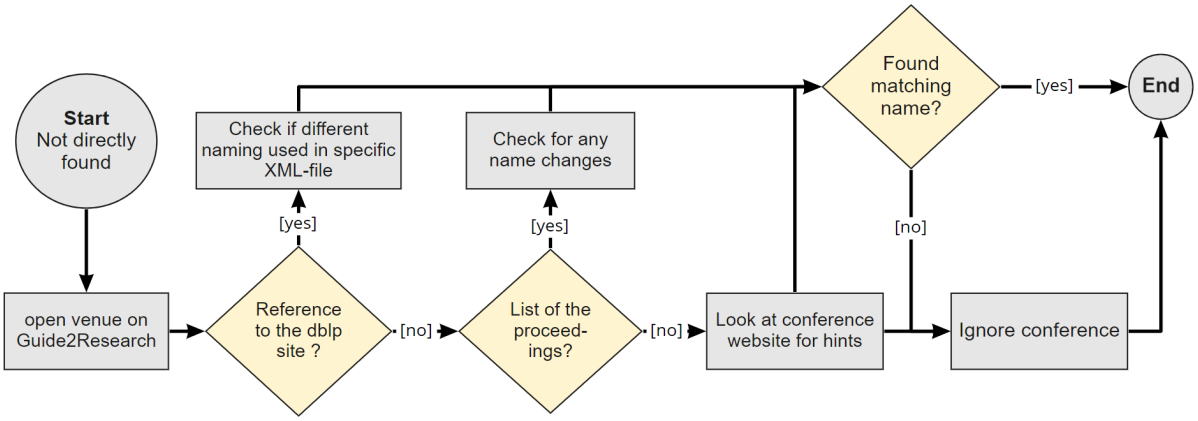

Figure 3.3: Steps to find missing venue names displayed in a flow diagram.

of the top 25 conferences was not accessible. The table demonstrates a clear connection between the conferences' relevance and match rate.

In the range of i-j	1-25	26-50	51-75	76-100	101-125	126-150	151-175	176-200
No. of Venues not found	1	3	5	6	5	7	11	15

Table 3.3: Distribution of missing conference names in intervals of 25.

Filter of the top 200 conferences

To continue working with the view of the top 200 conferences in the graph database, these must be filtered out from the whole set. For this purpose every conference node gets a new label called *Top200* assigned (see Code 3.6). Afterwards, the view with the top conferences can easily be accessed with a simple query.

```
//Load the list of top conferences
LOAD CSV
FROM "file:///venues_top_conferences.csv" AS row
WITH COLLECT(row[0]) AS list

//Match all venues that are represented in the list
MATCH(v:Venue)
WHERE v.name IN list
SET v:Top200
```

Code 3.6: New label for each of the top 200 conferences.

The initial aim was to create a graph projection and then apply the Louvain algorithm onto it. However, there was always the issue of the projection specifying the data subset incorrectly. Additional details will be provided in the section about the application of the Louvain algorithm (see section 4.1).

Removal of unnecessary Nodes and Relationships

To begin with, any venues that do not have the Top200 label are removed. Second, to address the projection issue stated before and to substantially reduce the quantity of data (see Table 3.4), any article nodes that have not been presented at one of the top 200 conferences are also removed. Also the authors are disregarded because they won't be used in the oncoming work.

	Venues	**Article**
No. of Nodes deleted	4.930	2.654.611
No. of Nodes left	148	424.396

Table 3.4: Statistics on the number of nodes eliminated per label and the number that remain.

Reduced Export for Python

Later, after the communities have been detected in Neo4j, which will be shown in section 4.1 Community Detection with Louvain Algorithm, one more step is taken to reduce the data. This is done because the amount of data is still difficult to manage with around 420k nodes. Therefore a further specification of the data is accomplished by only exporting a subset of the graph to Python. Figure 3.4 depicts the different emphasis levels, where the biggest contains all venues and all articles. The second level only includes the top 200 venues and their direct articles and the two small levels are the new subsets. These subsets will be exported from two graphs with different amounts of communities. The top 10 communities will be selected from a graph with many communities, while the articles published between 2011 and 2014 will be selected from a graph with a small number of communities.

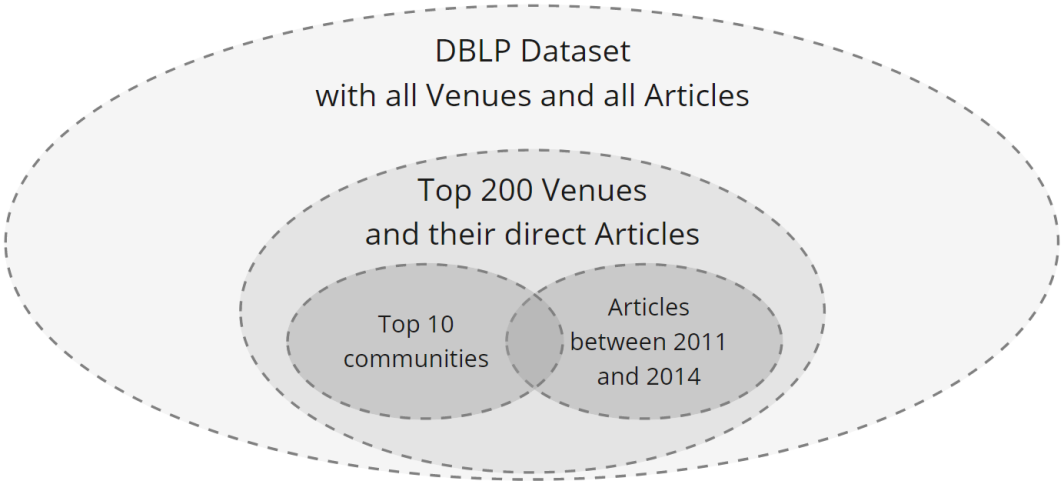

Figure 3.4: Figure showcasing the different emphasis levels on the dataset.

The data is exported to Neo4j using the APOC library's `apoc.export.json.cypher`

function. A batch size can be provided to prevent the Neo4j browser from crashing. By using a Cypher query, this method determines the subset of the graph. The query to retrieve the top 10 communities of the second run is used in the method in Code 3.7.

```
CALL apoc.export.json.query(
    "MATCH (n:Article)-[:VENUE]-(v:Top200)
    WHERE n.community_1 IN $listOfTop10Communities
    RETURN id(n) AS ID_Article, n.community_1 AS communityId,
↪  id(v) AS ID_RelatedVenue , n.title AS title, n.year AS year,
↪  n.abstract AS abstract",
    "top10_communities.json",
    {batchSize:10000})
```

Code 3.7: Exporting the top 10 communities to a JSON-file.

3.4 Text Preprocessing

To perform further analysis on the titles or abstracts, the strings must be preprocessed to remove unnecessary characters and reduce noise. In general, the strings are prepared so that the following models can handle them more precisely and efficiently. Although the text preprocessing will be applied further down this thesis, the steps needed are showcased here already:

1. Removal of any characters that do not contribute to the text's meaning (This primarily relates to LaTeX commands)
2. Cast to lowercase
3. *Tokenization* which means that the individual words are separated
4. Removal of stop words that don't contribute to the contextual information
5. *Lemmatization* which means that the words are converted to their lemma forms

Summary of this chapter

This chapter discussed the data understanding and preparation phase of the CRISP-DM cycle. After acquiring the data, it was briefly inspected, then arranged into a graph structure and subsequently imported into Neo4j. From there, the dataset was reduced to

the top 200 conferences and their related articles. Additionally, it was already discussed how to further specify a subset once the communities are detected. Eventually, the text processing steps were mentioned, which will be important for the analysis of the titles and abstracts in the next chapter.

4 Community Analysis

This chapter discusses community detection and further analyses using a variety of techniques and algorithms as it can be seen in Figure 4.1. The graph was shortened, imported and given the necessary preparation in the preceding chapter. First the communities will be detected by applying the Louvain algorithm. Subsequently a subset of these communities will be analyzed in Python where a new sample reduction approach is attempted. The goal is to determine a representative sample set through the help of topic modelling with LDA. Thereafter descriptive statistics will provide an overview of these representative articles and their venues to gain insights about their communities. To take this a step further, the *fastText* model will be used to embed titles in a high-dimensional vector space. Hereby weights are distributed based on the Inverse Document Frequency (IDF) to increase the relevancy of frequently occurring words. To be able to visualize the resulting high-dimensional data, the t-SNE algorithm is used to transform the data to a 2D and 3D representation.

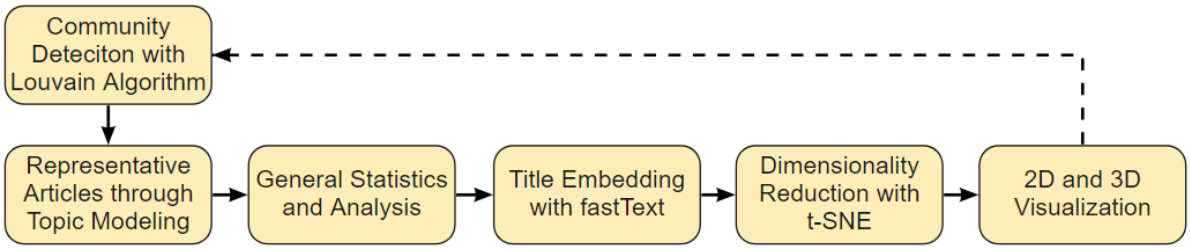

Figure 4.1: Overview of the techniques and methods used for the community analysis chapter.

4.1 Community Detection with Louvain Algorithm

This section describes the application of the Louvain algorithm from Neo4j's GDSL with the corresponding parameterization and the resulting communities.

Graph Projection

The Louvain algorithm is applied only on the subset of the graph with the top 200

conferences and the articles that have a direct relationship to one of them. A graph projection is typically created for this purpose, which means that a view containing the graph is created. In Cypher a graph projection can be defined using either the native projection or the cypher projection. Both types and their issues will be addressed in the following as there have been multiple attempts to get this to work.

1. **Cypher projection**

The node labels and relationship types are specified using appropriate queries. A Cypher statement is passed to the *relationshipQuery* field, requiring each relationship to be determined using only one source node and one target node. It was unclear at the time how such a query could be formed to establish the correct view. Though later a solution was found using the UNION clause (see Code 4.1)

```
CALL gds.graph.create.cypher(
       'top200_with_articles',
       'MATCH (t:Top200) RETURN id(t) AS id
        UNION MATCH (a:Article)--(:Top200)
        RETURN id(a) as id',
       'MATCH (n)-[:REFERENCES|VENUE]-(m)
        RETURN id(n) AS source, id(m) AS target',
        {validateRelationships:FALSE})
```

Code 4.1: Graph with native projection.

2. **Native projection**

The labels and relationship types used can be specified as strings, lists, or maps. However, specifying the labels via the list `["Articles","Top200"]` and the relation types via the list `["REFERENCES","VENUE"]` always resulted in large numbers of communities in the range of 50 thousand to 500 thousand. These appeared to be false because only about 400 thousand nodes are meant to be included for the projection. It was later discovered that the projection had not been constrained correctly to the right subset of the articles, leading to numerous one-man communities.

A workaround for these problems was sought after several attempts with these projections. As a result, the decision was made to delete all nodes from the database that would not be used in the following analysis. Subsequently the Louvain algorithm can be applied directly to the entire graph.

The Louvain algorithm is included in Neo4j's Graph Data Science library and can be used by a method call described in 4.2. The `graphName` field is parametrized with the graph projection and the `configuration` map is provided with the parameters of the

algorithm. The *write mode* is used which means that every node gets its community ID assigned in its new property which is specified in the mandatory `writeProperty` inside the `configuration`.

```
CALL gds.louvain.write(graphName: String, configuration: Map)
YIELD communityCount: Integer,
      ranLevels: Integer,
      modularity: Float,
      modularities: Integer[]
```

Code 4.2: Syntax of the Louvain algorithm in *write mode* with yield of interesting information about the run.

Parameters

To configure the algorithm the following parameters can be set:

- `maxLevels` specifies the maximum number of levels in which the graph is clustered and condensed.
- `maxIterations` defines the maximum number of iterations of the modularity optimization step.
- `tolerance` specifies the threshold of modularity change needed to run another iteration. If it's not reached, then the result is considered stable and thereby the algorithm stops.
- `writeProperty` specifies the name of the new property which will be written to each node and assigned with the corresponding community ID.

Many different evaluation runs were performed with `gds.louvain.stats()` which returns the statistics of the algorithm execution but doesn't make any changes to the database. This was able to provide an insight into the number of communities, generated according to each parameter set. The results varied significantly when the delta of the modularity score exceeded the threshold at times and did not at others. In Table 4.1 there are two runs listed that were written to the graph database with the property named as specified in `writeProperty`.

writeProperty	tolerance	maxIterations	maxLevels	No. of Communities
community_0	0.0001	10	10	13
community_1	0.1	10	5	1978

Table 4.1: Statistics about the Louvain algorithm runs that are written to the database.

First Write Run

By using the *stats mode* with the default parameter assignment, the graph was split into approximately 10 to 20 communities. In a write run, the community_0 property was then assigned to the nodes with the corresponding community IDs. A total of 13 communities were detected. The distribution of the nodes of this run is shown in Figure 4.2. The largest community here comprises over 68 thousand nodes, including 19 venues in addition to the articles.

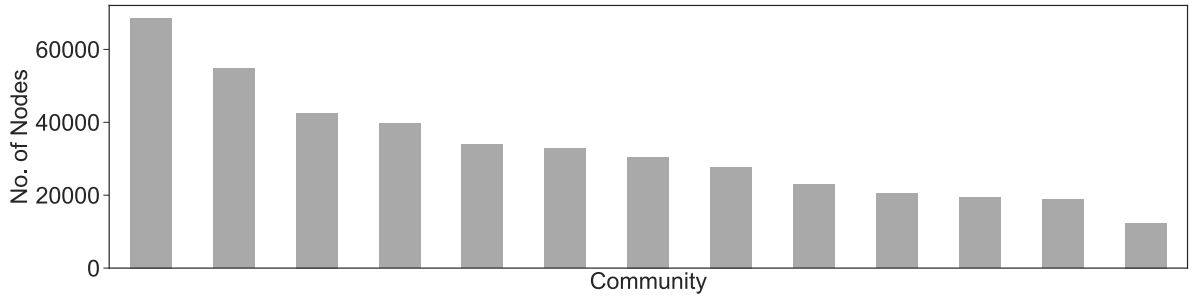

Figure 4.2: Distribution of the communities from the first run.

Second Write Run

By adjusting the parameters (see Table 4.1), 1978 communities were detected in that run. The node distribution of the top 100 communities can be seen in Figure 4.3.

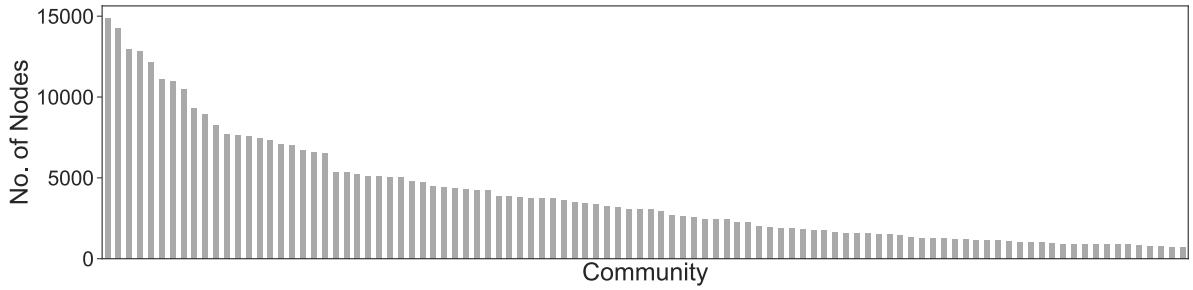

Figure 4.3: Distribution of the top 100 communities from the second run.

Due to the increased `tolerance` of 0.1 the algorithm stopped early and thereby ended up generating many more communities than in the first run. Though around 93% of them contain less than 100 members. In the further course of this work, analyses are run on the generated communities. The second run offers a good opportunity to further specify the dataset on the top 10 communities. This reduces the number of nodes from about 424 thousand to around 118 thousand which is a reduction of approximately 70%.

The updated graph schema with the new properties can be seen in Figure 4.4 which also excludes the *Author* and *Venue* labels whilst the *Top200* label is included since section 3.3 Removal of unnecessary Nodes and Relationships.

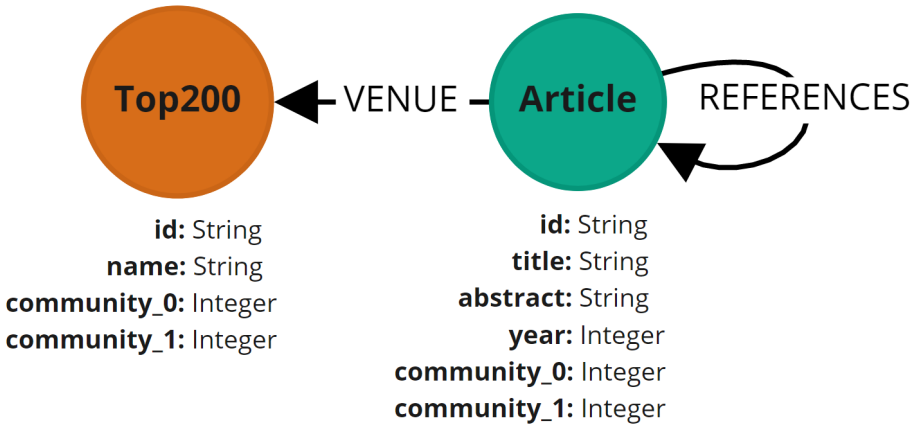

Figure 4.4: Updated graph schema with community properties.

4.2 Representative Articles

The detected communities still contain a large and varying number of articles. Performing analysis on such big data may still require a great quantity of computational resources and time. Additionally the amount of data may also pose two more issues when it comes to displaying the data. Firstly, plots with so many datapoints may become too extremely cluttered, making it difficult to highlight the differences in the visualizations. Secondly, comparing communities with highly varying member counts is inconvenient, as the count differences may mislead some statistics. To solve the issues and alongside further reduce the amount of data, representative articles for each community are examined.

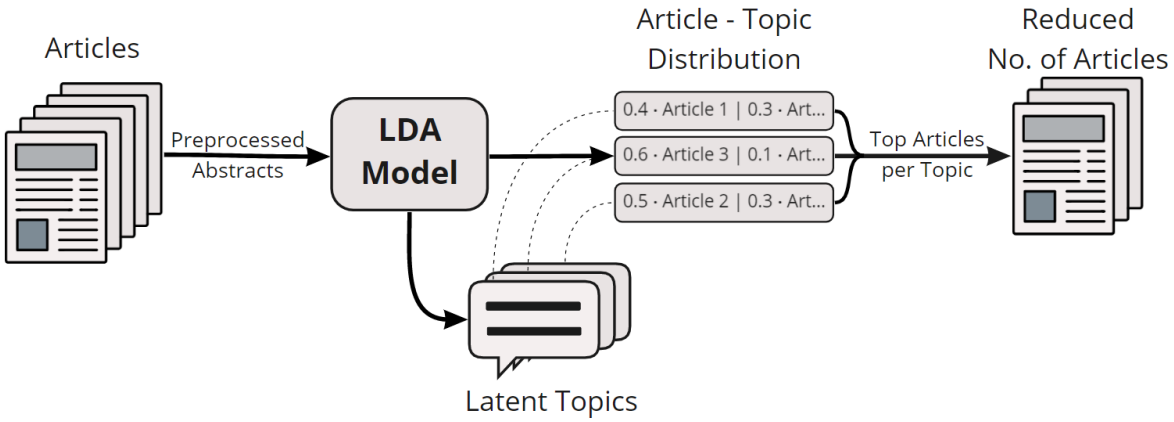

Figure 4.5: The key steps for the proposed approach to examine the representative articles for a community.

This is accomplished by a new proposed approach described in Figure 4.5. First the abstracts are analyzed and categorized into different latent topics using the LDA model. Afterwards, the presumably most relevant documents or articles in this case are extracted from the document-topic distribution θ_d.

These steps are performed independently for each community, and the resulting sets of articles are stored as CSV-files.

Text Preprocessing

First, it is necessary to clean and generalize the abstracts. For this purpose the procedures mentioned earlier in section 3.4 are applied. For each article two data types are needed. An array with the tokenized words in it and a string with the concatenated preprocessed words. One example of this preparation step is demonstrated in Code 4.3.

```
# Raw Abstract
"Utilization of communication technologies such as mobile phones
    ↪ and wireless sensor networks becomes much more common ..."
# Transformed to:
# - Array Representation
["utilization", "communication", "technology", "mobile", "phone",
    ↪ "wireless", "sensor", "network", "becomes", "common", ...]
# - String Representation
"utilization communication technology mobile phone wireless
    ↪ sensor network becomes common ..."
```

Code 4.3: Text preprocessing example of abstract with new array and string representation.

Vectorizing n-grams

The next step is to vectorize the preprocessed words. The goal is to map the concatenated strings into n-grams and eventually get them as vectors in the TF-IDF scheme. Mapping the n-grams should lead to the inclusion of more semantic information of the abstracts. For this, the `CountVectorizer` from the *scikit-learn* package is used in Python with the `ngramm_range` parameter. This range is set to `ngramm_range = (1-3)`, so that the abstracts are represented as a mixture of unigrams, bigrams, and trigrams. The results are term frequencies, because the `CountVectorizer` creates an NxM sparse matrix, where N denotes the number of individual n-grams and M the number of articles. Thus, from each n-gram, the corresponding frequency to the items of each abstract is represented. Here, the vocabulary becomes significantly larger due to the n-grams, but the direct connection of the neighboring words creates a higher semantic information

content. Then, using the *TfidfTransformer*, the TF-IDF model is created and fitted using the previously generated term frequencies. The TF-IDF values for a word w in an abstract are calculated mathematically by multiplying the term frequency with the inverse document frequency. The result are vectors that reflect the relevancy of each n-gram regarding any given abstract.

LDA Model

The topics are then determined by the LDA which is also used from the scikit-learn package and the model is created by with the method:
`LatentDirichletAllocation(n_components=10, max_iter=10)`
Here `n_components` specifies the number of topics that are generated and `max_iter` sets the number of iterations done before the model fitting stops. The less relevant parameters are disregarded here. Afterwards the vector representation of the abstracts can be passed to the `fit` method which gets applied to the LDA Model. The abstracts to topic probability density and the topic-word probability density can be retrieved from this model and are then used to extract the representative articles.

Retrieval of Representative Articles

To retrieve the most representative articles for each Community the method `get_representative_articles` (see Code 4.4) is defined and used. This function retrieves previously produced distributions for the current community and stores the results in a new dataframe. This is extended by a specified number of top articles for each topic. The results presented in this chapter are based on selecting 350 articles for each topic. The document-topic distribution matrix θ_d gives the important information, as each document is represented by a vector of probabilities which in turn reflect how well the document matches with each topic (Note that in this approach a document is equivalent to an abstract). The retrieval of the top documents is accomplished by sorting the document-topic distribution using `argsort` and then reversing the array and slicing the list to the specified amount. The corresponding articles are then appended to the dataframe `representatives_df`. After iterating through all topics, the dataframe containing all representative articles for each topic is returned.

```
def get_representative_articles(topic_word_distr, doc_topic_distr,
    ↪  no_top_documents):
    representatives_df = pd.DataFrame()
    # For each topic get n top docs sorted desc by their value
    for topic_idx, topic in enumerate(topic_word_distr):
        # Sort and reverse the list of the document distribution and
            ↪  slice it to no_top_documents
        top_doc_indices = np.argsort( doc_topic_distr[:,topic_idx]
            ↪  )[::-1][0:no_top_documents]
```

```
        # Add each of the top docs to a new representatives dataframe
        for doc_index in top_doc_indices:
            representatives_df = representatives_df.append(d
                ↪  f_articles_ofCom.iloc[doc_index])
    return representatives_df
```

Code 4.4: Method to get the representatives from the distributions in a community for each topic.

In this work the `no_of_topics` is specified to 10 and `no_top_documents` is set to 350, therefore 3500 representative articles are obtained in the dataframe. One last step is the removal of duplicates, as many documents belong to multiple topic's top documents. After removing them, only between 740 and 1033 remain, depending on the community. In total, this approach causes a reduction from 117878 articles to only 8992, which is a decrease of $\approx 92\%$.

4.3 Descriptive Analysis

Now that all communities are represented by their determined representative articles, an overview can be given by visualizing them with various descriptive analysis. These are primarily done on the top 10 communities from the second Louvain run where in total 1978 communities were detected. To begin with, a plot is created in which the publications of the articles are mapped against the time they were published to provide some chronological information. Figure 4.6 depicts the publications between 1990 and 2015. Generally, a definite upward trend can be noticed with the increase in time. To learn more about the communities, a statistic about the distribution of venues across these communities is shown in Figure 4.7. Because the exported dataset only contains the IDs of the related venues, the original venue names must be traced back. This is accomplished by first exporting all the top 200 venue names that correlate to the communities from Neo4j and then initializing a dictionary with the IDs as the keys and the corresponding abbreviations of the conference names as the values. Subsequently the IDs are replaced by the abbreviations. Figure 4.7 illustrates that the articles of each community are mainly presented at a single venue. This is expected as there are many communities in total and the articles in each of the top 200 are closely connected because they all have the *VENUE* relationship to their specific venue. Interestingly, there are noticeable spikes to be seen. Especially the community with the ID 21349, which is shown in gray, has two such spikes, one in 2004 and one in 2007. The articles of the gray community were presented almost exclusively at the *IGARSS: International Geoscience and Remote Sensing Society*.

To further compare the ten communities, the most relevant 15 words are extracted

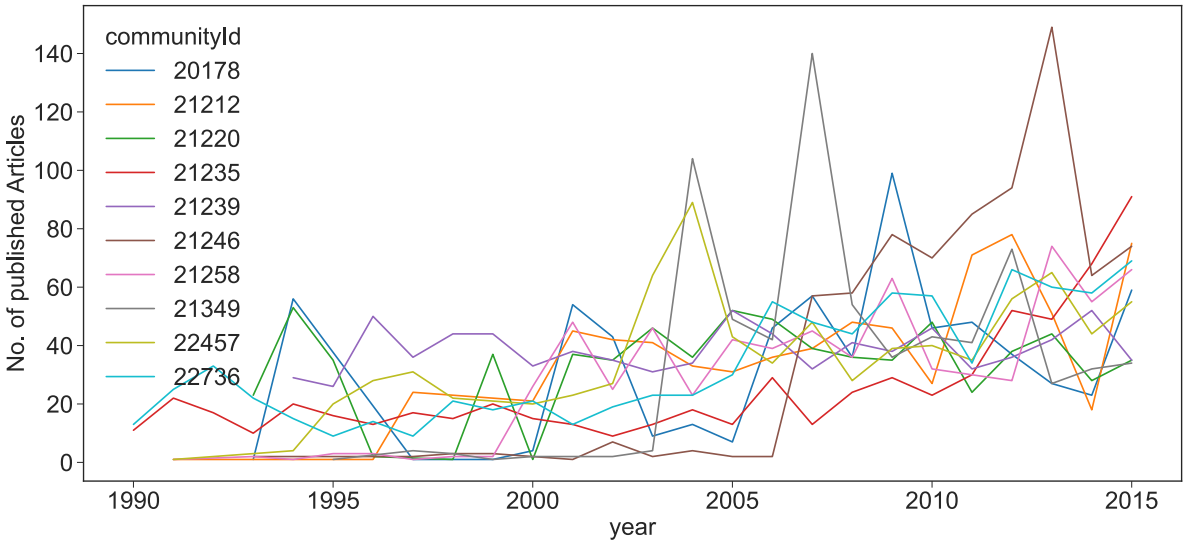

Figure 4.6: Distribution of articles between 1990 and 2017 of each community.

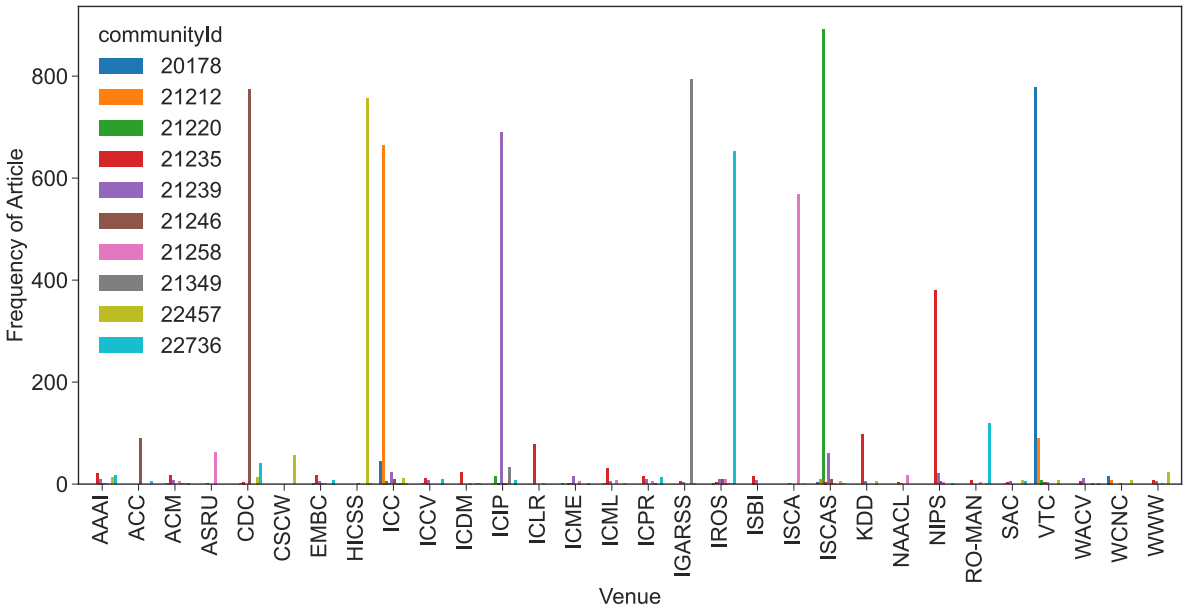

Figure 4.7: Distribution of the articles ordered by the communities against the individual venues.

and the corresponding frequencies in the respective communities are calculated which is shown in Figure 4.8. Most of the words are spread fairly homogeneously. The word

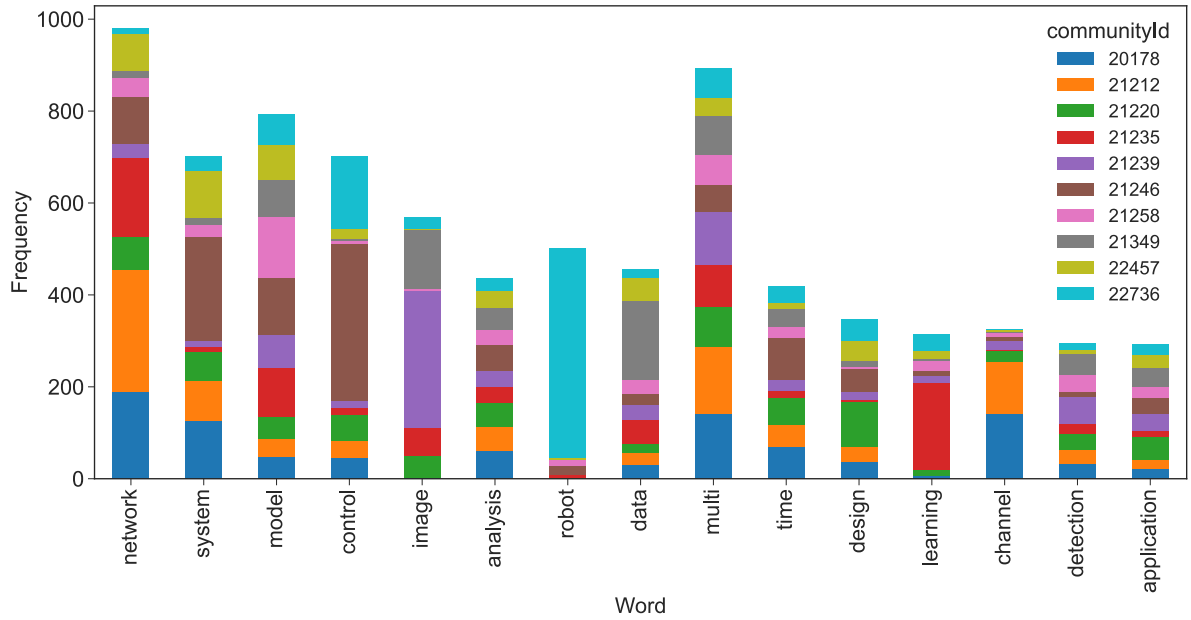

Figure 4.8: The 15 most used words across all communities broken down by the frequency of the respective community.

"robot" appears noticeably one-sided in the cyan community with the ID 22736. Figure 4.7 shows that the articles in this community are, as expected, mainly in the field of robotics, since the venues *IROS: International Conference on Intelligent Robots and Systems* and *RO-MAN: The IEEE International Symposium on Robot and Human Interactive Communication* are the most dominant ones.

4.4 Comparison of the Titles

Next, an attempt is made to reveal the underlying structures by comparing the article titles. To accomplish this, the titles are transformed into a high-dimensional vector space using the *fastText* word embedding model and subsequently t-SNE is used to reduce the data back to two- and three-dimensional representation so that the data can be visualized.

Word Embedding with fastText

There are numerous different types of text embeddings available. The *Word2Vec* word embedding and its extended version *Doc2Vec* are the most well-known. *Doc2Vec* is a

sentence embedding approach and therefore likely to produce superior results because it embeds contextual information, the *fastText* model introduced by Bojanowski et al. [4] was chosen instead. Likewise, it allows the embedding of contextual information due to the use of character n-grams which is a new and effective approach, especially when employing a pre-trained model that may not include all words. A 300-dimensional *fastText* model, that had been pre-trained with a million word vectors from Wikipedia, is utilized. The Python package *pymagnitude* was used to efficiently load the magnitude file[1].

Inverse Documents Frequency Weights

The IDF-scheme is used to compute the vectors for each word in the titles and subsequently assign them a weight in the vector mapping.

This can be seen in the method `idf_fasttext(df)` in Code 4.5. The method gets the representative articles passed, and for each title, every words' corresponding vector from the *fastText* Model is saved in an array. Every word is weighted by its IDF score which is computed in advance and stored in a dictionary. The embedding of the titles are calculated by averaging the products of the individual vectors and the IDF scores.

```
def idf_fasttext(repr_articles_df):
    vectors = []
    for title, comm in tqdm(zip(df.title.values,
        ↪  df.communityId.values)):
        fasttext_vectors = fasttext.query(word_tokenize(title))
        weights = [idf_dicts[comm].get(word, 1) for word in
            ↪  word_tokenize(title)]
        vectors.append(np.average(fasttext_vectors, axis = 0, weights
            ↪  = weights))
    return np.array(vectors)
```

Code 4.5: A method for embedding titles with the use of *fastText* vectors and IDF scores as weights.

Dimensionality Reduction with t-SNE

To visualize the vectorized data of the individual communities, the high dimensionality of 300 must be reduced to either two or three dimensions. The t-SNE algorithm from the scikit-learn package is used for this. A 2D representation was created using a perplexity of 50 and 5000 iteration steps. The result can be seen in the scatter plot in Figure 4.9. Two major data clusters are visible in the two-dimensional scatter plot. The one on the

[1]The download link of the .magnitude file: http://magnitude.plasticity.ai/fasttext/medium/wiki-news-300d-1M-subword.magnitude

right includes all communities, and each has areas of increased concentration. However, there are numerous title embeddings that are positioned far from these main areas.

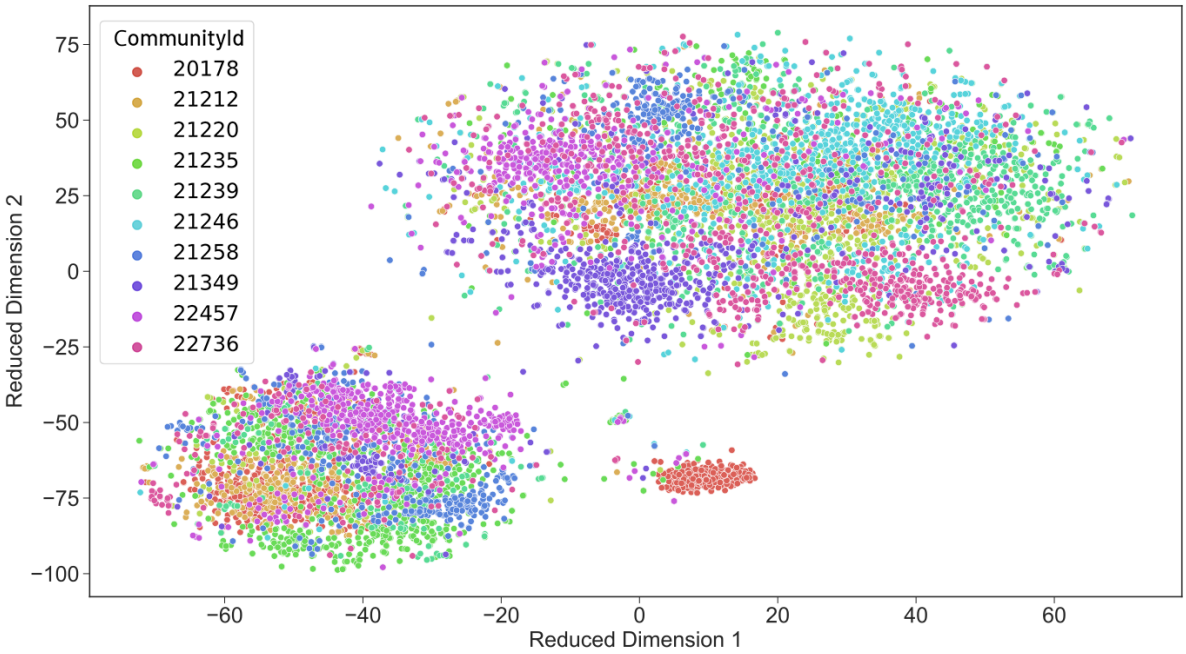

Figure 4.9: Scatter plot showing the title embedding in a two dimensional space.

Likewise, the cluster on the bottom left contains data points from all communities, but only a small number of turquoise and yellow-green communities are represented. Two major data clusters are visible in the two-dimensional scatter plot. The one on the right includes all communities, and each has concentrated areas. However, there are numerous title embeddings that are positioned far from these main areas. Similarly, the cluster on the bottom left contains data points from all communities except for a few turquoise (●) and lime green (●) communities. There is one extra group that contains the titles primarily from the red (●) community. In comparison to the ones on the outside, these title representations appear to be quite diverse.

Additionally, the titles are displayed in a three-dimensional scatter plot. As a result, the number of components in the t-SNE reduction is increased from two to three while maintaining the same parameters for perplexity and iteration steps. An animated[2] visualization can be seen in Figure 4.10. Due to the plot's dense population, it's difficult to perceive individual formations. However, for the majority of communities, it is possible to recognize areas with a heavy proportion of their titles.

[2]Note that Adobe Acrobat may be required to play the animation

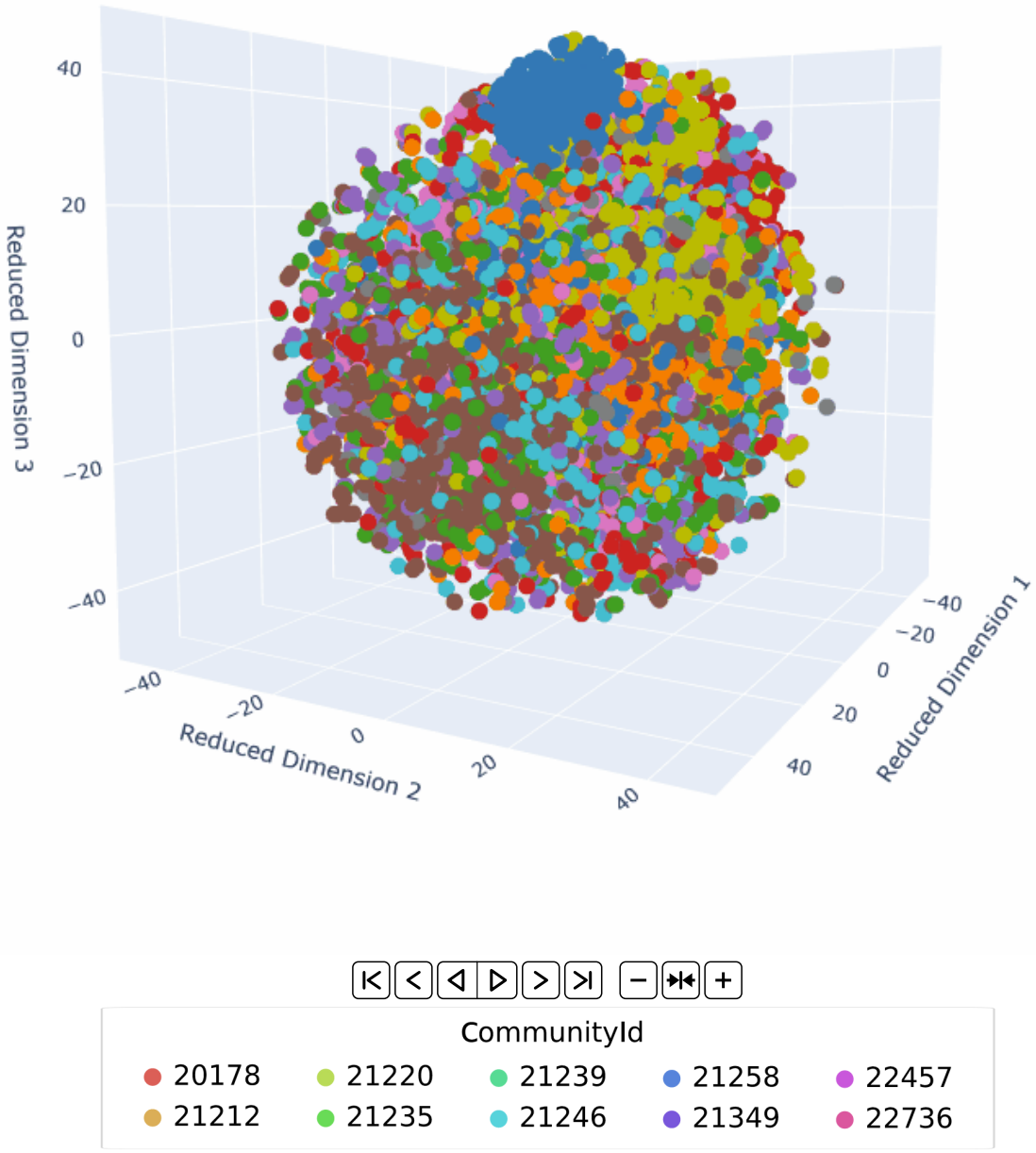

Figure 4.10: Animation of the 3D scatter plot showing the title embeddings.

4.5 Evaluation

In this chapter it was shown that the prepared graph data can be handled and analyzed through various algorithms and models. The application of the Louvain algorithm allowed the segmentation of the data into communities whereby new structural knowledge was meant to be attained. But especially in the second run the gain in knowledge was limited because each community primarily consisted of articles that were linked to a single venue.

After exporting a defined subset of the communities, the new proposed approach was tested. This resulted in a massive decrease of 92%, making subsequent analyses more affordable, manageable, and easier to visualize. The descriptive statistics helped to give an overview of this reduced sample set. Additionally, the quantity of articles per community was balanced. This liberated the statistics from disparity in member counts and facilitated a more direct comparison of other relevant information. Whether the information density remained stable or not is unclear but assuming that the latent topics cover all important areas of a community, the selected articles may also represent the community well. Outliers are very likely to be ignored with this sample reduction technique but may also be used as an advantage.

The subsequent analysis of the embedded titles was made possible through the dimensionality reduction performed with the t-SNE algorithm. This procedure enabled a direct comparison of the titles in a 2D and 3D scatter plot. The 2D plot provided a good overview of the different communities but the 3D plot was very cluttered and therefore it was difficult to extract useful information. Understanding such visualizations is generally difficult though, because the representation of high-dimensional data in a low dimensional space is prone to be misinterpreted. These results may lead to new experiments though. Investigations on the reasons behind the characteristics of the formed clusters described in section 4.4 may yield interesting insights into the data.

Reducing the data this heavily through specification and sample reduction seems to be very helpful because the plots already contain a great deal of cluttered information; if all articles were used, these scatter plots would have 13 times the amount of data points. Comprehending them might have been even more challenging.

5 Conclusion

An overview of the study is provided at the summary of this thesis followed by an outlook on future research possibilities.

5.1 Summary

In the two previous chapters, the data was first acquired, then prepared by transforming it into a graph structure and subsequently reduced in size and complexity to make it more manageable. This reduction was accomplished by limiting the scope, which was chosen to be the top 200 conferences and all the articles that have a direct connection to any one of them. Thereby only 148 conferences of the top 200 from *Guide2Research* were identified within the dataset.
From there on, the graph analysis started by dividing all the entities into communities using the Louvain algorithm. Multiple runs with different parameters were written to the Neo4j database but the focus was set on the second run in which 1978 communities were created in total. A further narrowing of the dataset was then accomplished by exporting only the top 10 communities to Python.

Following that, the problem of a representative dataset was addressed using the newly proposed approach where the representative articles were extracted from the document to topic distribution θ_d of the LDA Model. Although the extracted articles have not been verified in their accuracy describing all the data yet, these articles are at least highly representative to the ten generated latent topics. A further decrease of around 92% in the number of articles was achieved through this sample reduction approach, which allows the data to be managed and comprehended easier when plotted.

Communities were then put into perspective by giving an overview over temporal and contextual information with the help of descriptive statistics. Afterwards the titles were transformed into high-dimensional vector representations to facilitate a discrete comparison of the articles in a single figure. To be able to visualize the data the vector space was then reduced to a 2D, and 3D space by applying the t-SNE algorithm.

5.2 Outlook

To conclude this thesis, the last section addresses unresolved issues and potential areas of further research.

It became clear that the top ten communities of the second run, which the primary focus was placed on, was limited by the detected communities typically consisting of only one venue and its articles. As a result, the increase in structural knowledge is also limited. In fact, comparable findings may be obtained by using only the top ten venues and their articles. The more interesting combination of communities happened during the first run, as there are 13 communities in total and all 148 venues are distributed among them. Due to time constraints, the emphasis was restricted to a single run, which happened to be the second one. Eventually the execution and preparation of the plots were completed in the very final stages of this work, but the deadline unfortunately did not allow for an integration into the main part of the thesis so they had to be added to the appendix. Further realizations may come from observations and studies of these plots.

The proposed sample reduction approach based on LDA is yet to be verified for its accuracy in describing all of the data of each community. Future research may compare this approach to other sample reduction techniques. Furthermore, the number of topics and representative documents in this work were arbitrarily set to ten topics and 350 articles for each community. Additional research should be conducted to determine which parameters are optimal in this regard keeping in mind that this may be highly context-dependent.

Another way to evaluate this approach may be to run parallel analysis on the representatives and the entire dataset and then compare the results. Afterwards the results may be investigated to see what the differences between those are.

The 2D and 3D visualizations of the titles may be improved because it is difficult to recognize structural coherence of the communities' data points. This may be the result of a poor vector representation of the titles. Experiments may be conducted using Sentence Embedding techniques rather than Word Embedding techniques.

Bibliography

[1] Andrew Allman, Wentao Tang, and Prodromos Daoutidis. "Towards a Generic Algorithm for Identifying High-Quality Decompositions of Optimization Problems". In: *13th International Symposium on Process Systems Engineering*. Vol. 44. Computer Aided Chemical Engineering. Elsevier, 2018, pp. 943–948.

[2] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. "Latent Dirichlet Allocation". In: *J. Mach. Learn. Res.* Vol. 3 (2003), pp. 993–1022.

[3] Vincent D. Blondel et al. "Fast unfolding of community hierarchies in large networks". In: *Journal of Statistical Mechanics: Theory and Experiment* (2008).

[4] Piotr Bojanowski et al. "Enriching Word Vectors with Subword Information". In: *Transactions of the Association for Computational Linguistics* Vol. 5 (2016).

[5] Santo Fortunato. "Community detection in graphs". In: *Physics Reports* Vol. 486 (2009), pp. 75–174.

[6] Muhammad Aqib Javed et al. "Community detection in networks: A multidisciplinary review". In: *Journal of Network and Computer Applications* Vol. 108 (2018), pp. 87–111.

[7] Hamed Jelodar et al. "Latent Dirichlet allocation (LDA) and topic modeling: models, applications, a survey". In: *Multimedia Tools and Applications* Vol. 78 (2019), pp. 15169–15211.

[8] Arzum Karataş and Serap Şahin. "Application Areas of Community Detection: A Review". In: *2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism*. 2018, pp. 65–70.

[9] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* Vol. 9 (2008), pp. 2579–2605.

[10] Tomás Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *1st International Conference on Learning Representations*. 2013.

[11] Mark Needham and E. Hodler Amy. *Graph Algorithms*. O'Reilly, 2019.

[12] M. E. J. Newman. "Equivalence between modularity optimization and maximum likelihood methods for community detection". In: *Physical Review E* Vol. 94 (2016).

[13] Digital Bibliography & Library Project. *dblp: computer science bibliography*. URL: `https://dblp.org/` (visited on 08/01/2021).

[14] Digital Bibliography & Library Project. *dblp: What is dblp?* URL: `https://dblp.org/faq/1474565.html` (visited on 08/01/2021).

[15] Jie Tang et al. "ArnetMiner: extraction and mining of academic social networks". In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data*. ACM, 2008, pp. 990–998.

[16] Laurens Van Der Maaten. "Accelerating t-SNE using tree-based algorithms". In: *The Journal of Machine Learning Research* Vol. 15, no. 1 (2014), pp. 3221–3245.

List of Figures

List of Tables

Code and Listings

Appendix

1 Additional Visualizations of the Second Run

There are some more analyses done from the first run that are not included in the main part of the thesis due to time constraints or because they don't add much value.

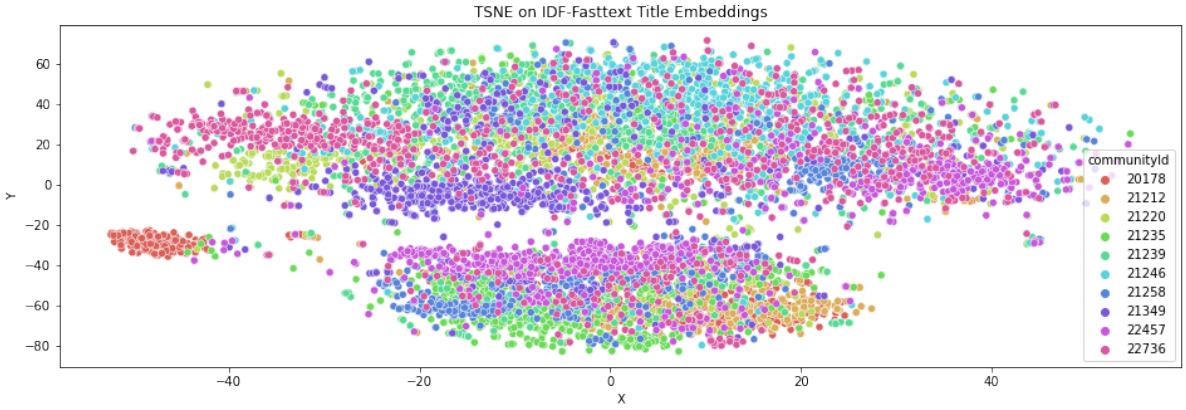

Figure 1.1: Smaller 2D scatter plot of title embeddings reduced with t-SNE

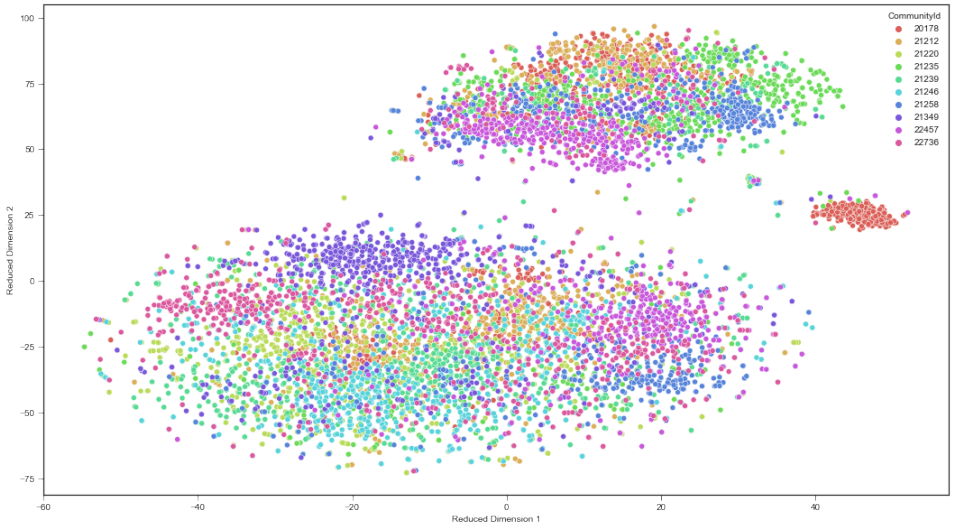

Figure 1.2: 2D scatter plot of title embeddings reduced with PCA.

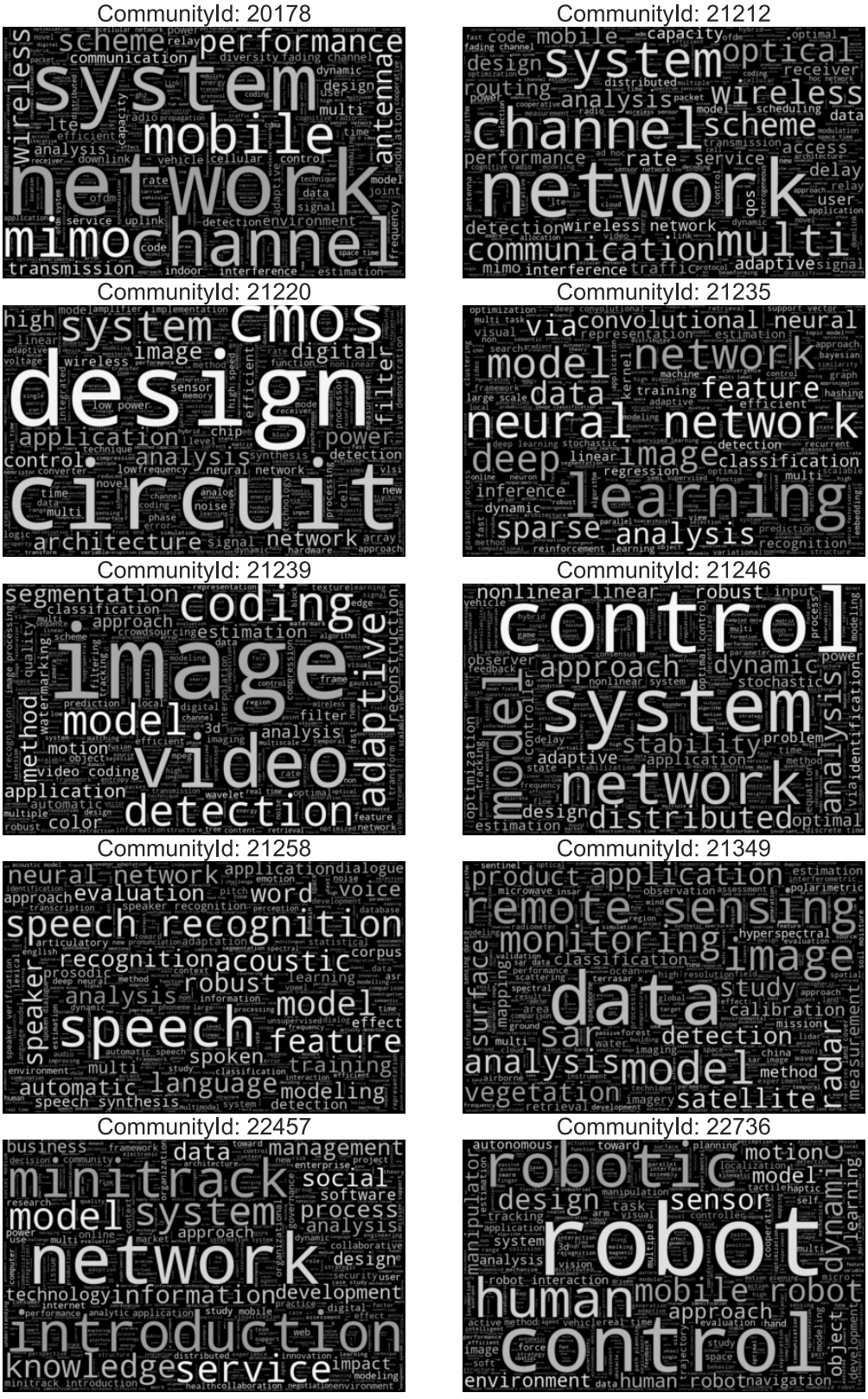

Figure 1.3: Word clouds showcasing the frequencies of the words from the titles. The bigger the word the more frequent.

2 Visualizations of the First Run

In the very last stages of the thesis the plots could be performed and prepared but it was too late to integrate them, so they are are just appended here. All communities' articles that were published between 2011 and 2014 were used for these analyses and visualizations.

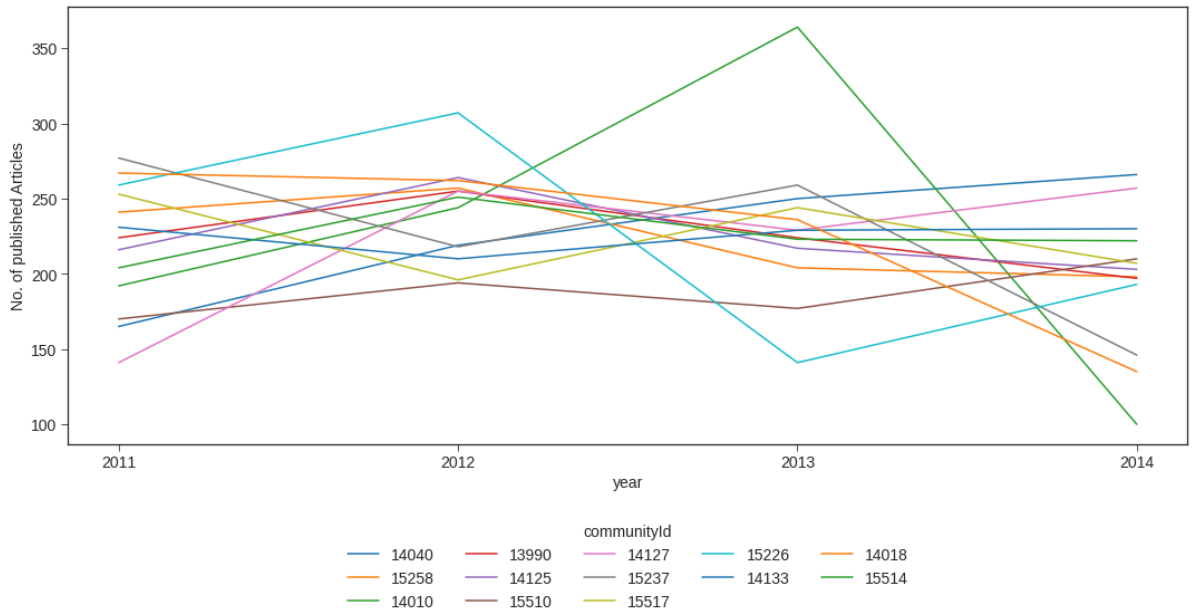

Figure 2.1: Publications over time.

Figure 2.2: Articles per venue.

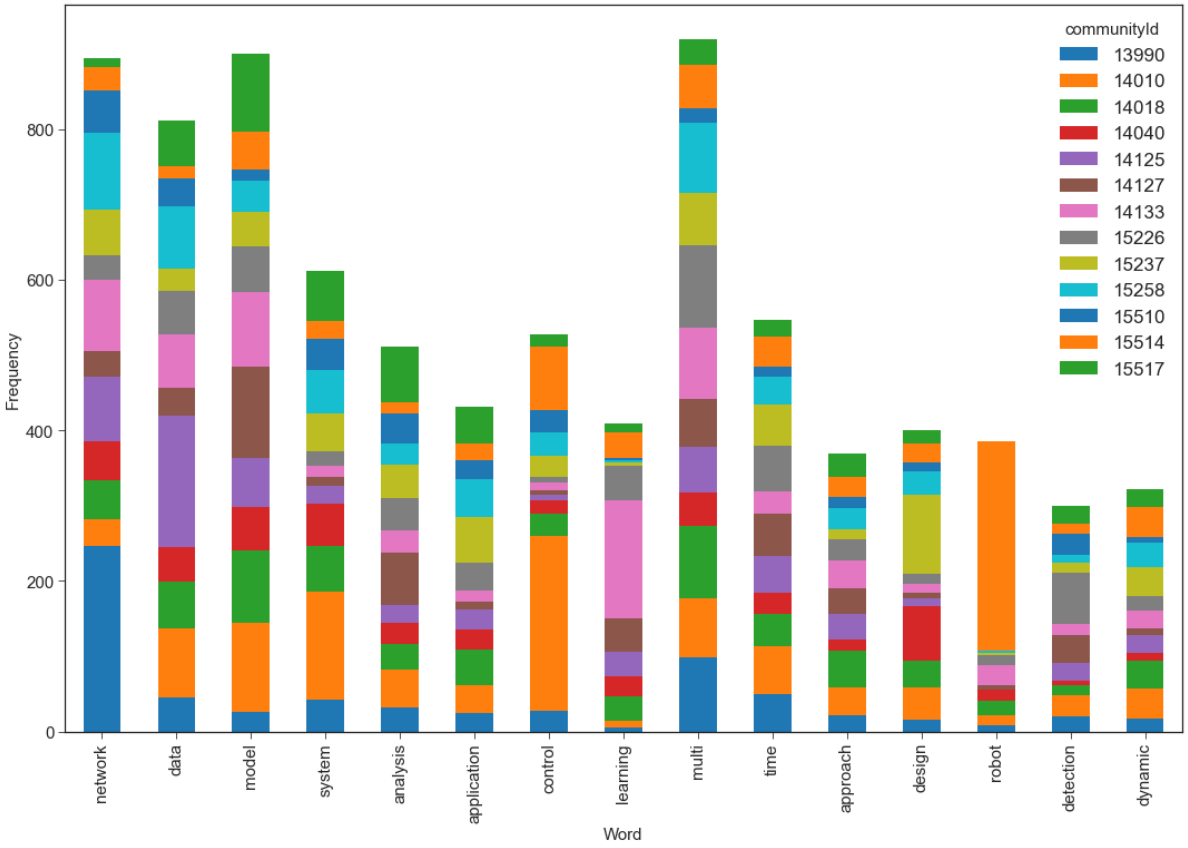

Figure 2.3: Top 15 words.

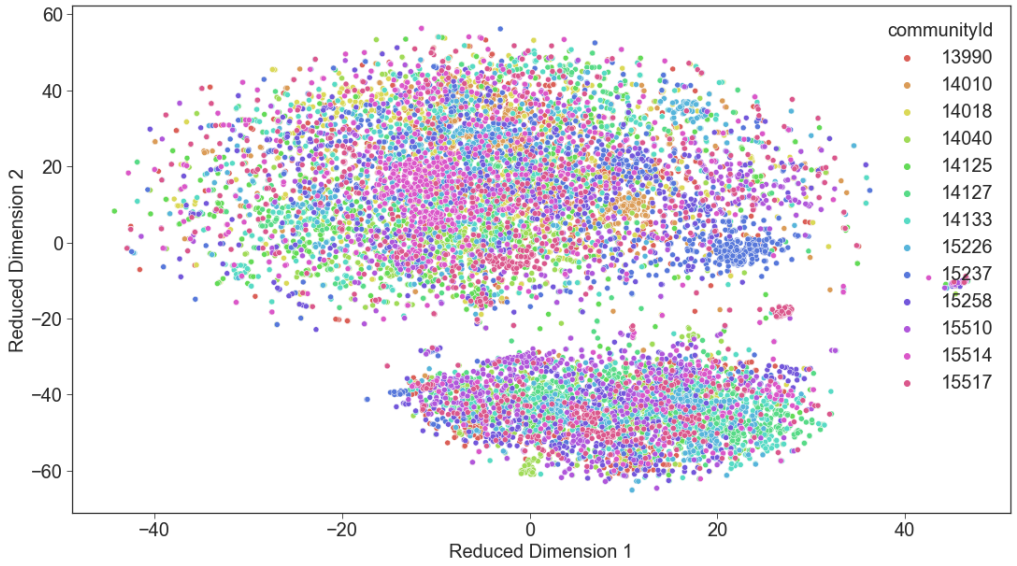

Figure 2.4: 2D scatter plot of title embeddings reduced with t-SNE.

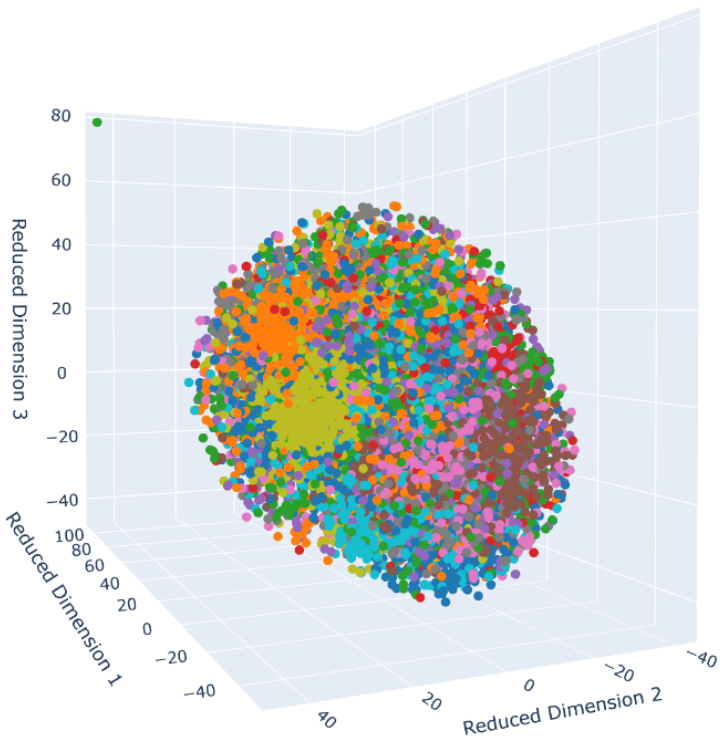

Figure 2.5: 3D scatter plot of title embeddings reduced with t-SNE.

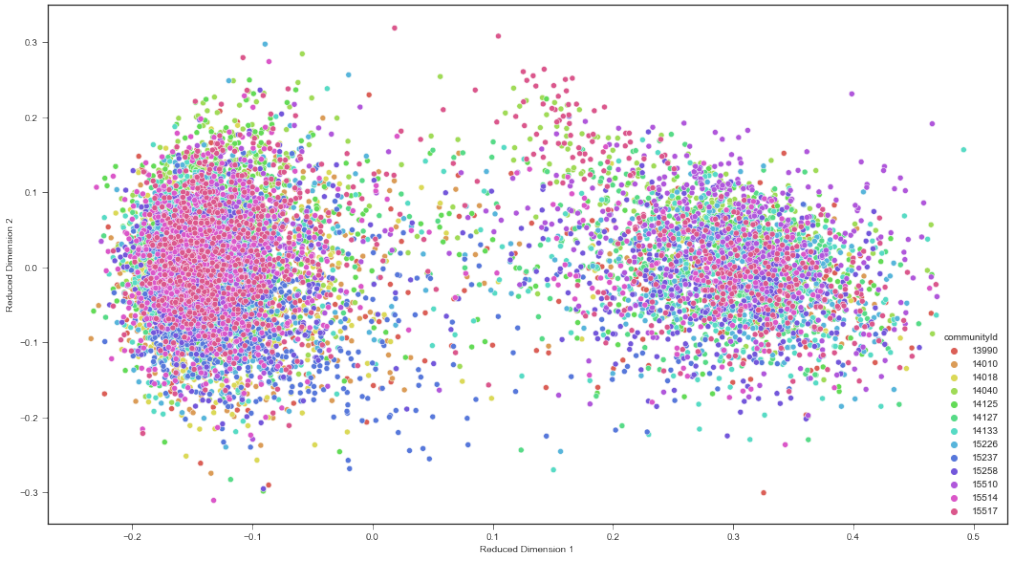

Figure 2.6: 2D scatter plot of title embeddings reduced with PCA.

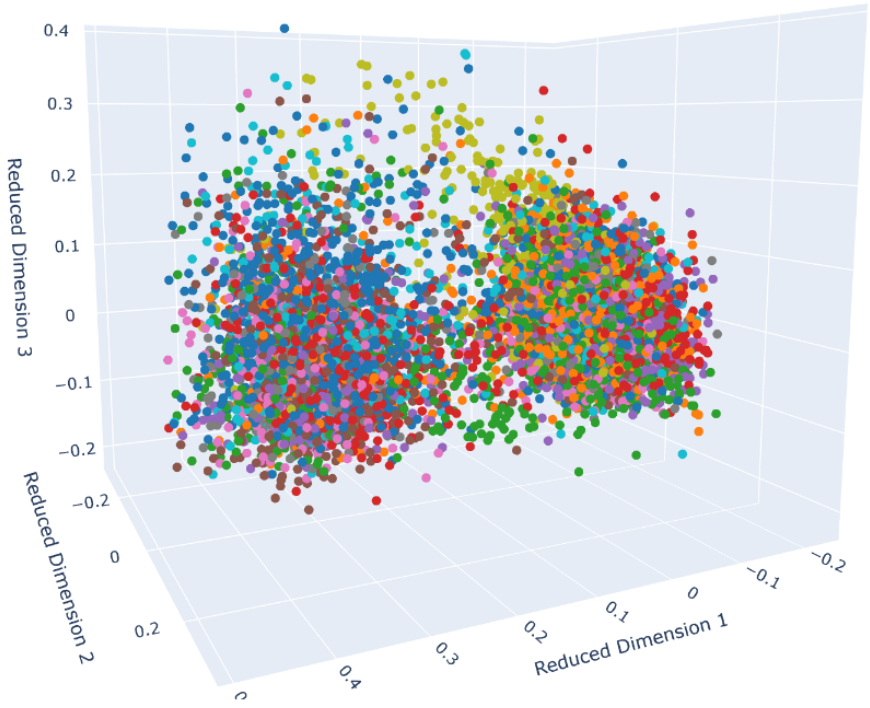

Figure 2.7: 3D scatter plot of title embeddings reduced with PCA.

Figure 2.8: Word clouds showcasing the frequencies of the words from the titles. The bigger the word the more frequent.

Acknowledgment

Finally I would like to express my gratitude to Prof. Dr. Schleif and M.Sc. Münch for their guidance and support.

I owe a debt of gratitude to my partner, who has always encouraged me when I have had doubts and reminded me to take breaks when I have been in a state of confusion.

In addition I would like to thank my family and friends for being supportive.

Also I would like to thank my employer for being patient with me when the thesis required all attention.

Last but not least I want to thank my two cats for being the best stress relievers.

Affidavit

I hereby affirm that I have independently written the submitted bachelor thesis and have not yet submitted it elsewhere for examination purposes. All sources and aids used are indicated, literal and analogous quotations are marked as such.

Simon Weickert, September 14, 2021

Approval for Plagiarism Check

I hereby agree that for the purpose of checking for plagiarism my submitted work will be transmitted in digital form to PlagScan (www.plagscan.com) and stored temporarily (max. 5 years) in the database maintained by PlagScan and that personal data, which are part of this work, will be stored there.

Consent is voluntary. Without this consent, plagiarism checks cannot be prevented by removing all personal information and observing copyright regulations. Consent to the storage and use of personal data can be revoked at any time by declaration to the faculty.

Simon Weickert, September 14, 2021