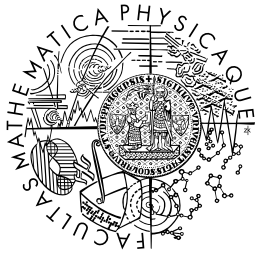


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

DOCTORAL THESIS

Mgr. Peter Zvirinský

Data mining in social network analysis

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the doctoral thesis:	doc. RNDr. Iveta Mrázová CSc.
Study programme:	Theoretical Computer Science and Artificial Intelligence

Prague 2023

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources.

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

In ..... date ..... signature of the author

Acknowledgments

First of all, I would like to thank my supervisor, doc. RNDr. Iveta Mrázová, CSc. for her kind, patient, and tireless guidance and support throughout my PhD studies. I am also thankful for all the feedback and corrections I received for this dissertation to make it my best work yet. For all the remaining errors, I alone am responsible.

Throughout my doctoral study, my research work was partially supported by the Grant Agency of Charles University under Grant No. 120616, and I would like to thank Charles University for its support.

Finally, I am grateful to my whole family, especially my wife Romana and son Dorian, for supporting me during my seemingly never-ending studies. They were always patient with me and provided me with time and space to finish this thesis.

Title: Data mining in social network analysis

Author: Mgr. Peter Zvirinský

Department: Department of Theoretical Computer Science and Mathematical Logic

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

Abstract. In the past several years, the global economy has experienced a significant increase in overall debt, reaching 238% of the world GDP in 2022, as reported by the International Monetary Fund. This growing indebtedness raises concerns about the stability of the financial system and the welfare of individuals and institutions. It also underscores the need for effective strategies to understand the intricate relationships between debtors and creditors and to mitigate associated risks. In response, this thesis proposes a novel approach based on data mining methods for the comprehensive analysis of debt formation patterns among individuals and companies, focusing on the largely untapped data from the Insolvency Register (IR) of the Czech Republic.

We aim to leverage social network analysis (SNA) methods to model and analyze the interactions among subjects participating in insolvencies, namely debtors, creditors, and insolvency administrators. Additionally, we focus our research on dynamic social networks that capture structural changes in the data over time. Our approach enables an in-depth exploration of interactions, offering insights into debtors' and creditors' behavior and facilitating the prediction of future developments related to bankruptcies.

The methodology proposed in this thesis contributes to a better understanding of economic systems, identification of financial distress patterns, and facilitates decision-making in insolvency-related matters. Furthermore, the generic nature of the approach allows for its application to data from other insolvency registers and publicly available datasets, assuming similar preprocessing steps are undertaken.

Keywords: data mining, knowledge discovery, social network analysis, insolvency register, structured data, unstructured data

Contents

Introduction

In recent years, the global economy has experienced a surge in debt levels, reaching unprecedented heights. According to the International Monetary Fund (IMF), the worldwide total debt-to-GDP (public, household, and corporate) ratio soared to 238% in 2022[1]. Of the total debt-to-GDP ratio, 38.7% corresponds to public debt, 23.1% to household debt, and 38.2% to corporate debt. This surge in indebtedness raises critical concerns about the stability of financial systems and the well-being of individuals and institutions.

The same IMF report reveals comparable debt-to-GDP ratios of 254.4% for the EU and 273.9% for the US, while Japan maintains the highest ratio at 447.7%. However, the countries' abilities to manage and repay their debt vary greatly. For instance, despite its large debt-to-GDP for more than 30 years, Japan benefits from the majority of its debt being in its local currency. The Bank of Japan's low to negative interest rate policy ensures manageable debt-servicing costs alongside predictable economic growth and political stability. In contrast, Italy, despite having a relatively small total debt-to-GDP ratio of just above 260%, faces significantly higher risks. Unlike Japan, Italy's debt is primarily denominated in euro, making its debt-servicing costs directly susceptible to concerns about its economic prospects. Political instability and struggle to implement essential structural reforms further threaten Italy's ability to meet its debt obligations.

As the volume of debt generally continues to rise, the processes of accruing and managing debt are becoming increasingly complex. This poses a challenge in comprehending the intricate relationships between debtors and their creditors. At the same time, it is crucial to discover effective ways to minimize risks linked to the attitudes of consumers and corporations toward debt to prevent potential bankruptcies. A prominent case illustrating the consequences of failing to handle these risks is the Lehman Brothers' bankruptcy in 2008, recognized as one of the largest corporate bankruptcies in history. This bankruptcy had far-reaching implications and is considered to have triggered the 2008 financial crisis, resulting in more than two trillion USD in losses to the global economy.

The Lehman Brothers' bankruptcy was caused by an interplay of multiple factors, including excessive risk-taking, overexposure to subprime mortgages[2], and lack of adequate liquidity. This bankruptcy led to a credit market freeze as financial institutions became hesitant to lend to one another, having cascading effects on businesses and individuals, often resulting in secondary defaults. Consequently, numerous countries witnessed negative economic growth, escalating unemployment rates, and substantial consumer and business spending declines. In 2009, the United States witnessed its unemployment rate[3] soar to 10% and

[1]The debt-to-GDP ratio was obtained from the latest IMF report (page 1 and 6): `https://www.imf.org/-/media/Files/Conferences/2023/2023-09-2023-global-debt-monitor.ashx`. Accessed 21 November 2023.

[2]Subprime mortgages are loans with higher interest rates than traditional bank loans offered to borrowers with lower-quality credit histories.

[3]Source: US Bureau of Labor Statistics (`https://www.bls.gov/`)

also experienced a decline in GDP[4] by -2.9%. Industries, particularly manufacturing, were severely affected, leading to major bankruptcies such as those of General Motors[5] and the Chrysler Corporation[6]. In light of these challenges, there is a growing need to delve into the details of debt formation processes more comprehensively than ever before.

A vast amount of bankruptcy-related data is readily available through publicly accessible insolvency registers, particularly in the EU[7]. As of today, this data represents an untapped opportunity for a comprehensive analysis of debtors' behavior during the bankrutpcy process, but also the events leading up to it. In this study, we propose a novel approach for the large-scale analysis of debt formation patterns among individuals and companies. The main source of data used in this thesis will be the Insolvency Register (IR) of the Czech Republic, which contains over 20 million documents related to more than 370,000 insolvencies spanning over 14 years. Considering the substantial volume of data in the IR, our approach will rely on big data technologies to first extract information from structured data, such as debtors' demographic information and the list of creditors. More importantly, however, we will also use big data technologies to extract domain-specific information from a much larger volume of unstructured data in the IR, encompassing the 20 million documents related to individual insolvencies.

The use of big data technologies within the scope of this thesis will begin with the effective scraping and storing of data from the IR. In the subsequent step, we employ the latest Optical Character Recognition (OCR) software to extract data from millions of documents that we will also download from the IR. We will employ parallelization technologies available in both the Java and Python ecosystems to facilitate this process. Ultimately, this approach should lead to a comprehensive dataset related to insolvencies, offering us the advantage for subsequent analyses. This dataset is intended to cover the insolvency process details, including the studied debt's amount, characteristics, and associated timeline.

The insolvency process involves various participants, including debtors, creditors, insolvency courts, and, notably, insolvency administrators. Insolvency administrators play a crucial role in facilitating communication between debtors and creditors. We will refer to these entities collectively as *insolvency actors*. The interactions among the insolvency actors are complex, typically occurring before and during insolvency. Conventional mathematical methods often struggle to capture the dynamic and complex nature of these interactions.

In this thesis, we plan to adopt the data mining methodology, which focuses on extracting and discovering patterns in large datasets, employing techniques at the intersection of machine learning, statistics, and database systems. Social network analysis (SNA) is a subfield of data mining that utilizes networks and graph theory to examine interactions among entities (nodes) and their relationships (edges).

[4]Source: The World Bank (`https://data.worldbank.org/`

[5]General Motors bankruptcy announcement: `https://www.theguardian.com/business/2009/jun/01/general-motors-bankruptcy-chapter-11`. Accessed 21 November 2023

[6]Chrysler Corp. bankruptcy announcement: `https://www.theguardian.com/business/2009/apr/30/chrysler-verge-bankruptcy-talks-collapse`. Accessed at 21 November 2023

[7]List of all active insolvency registers in the EU are available at: `https://e-justice.europa.eu/110/EN/bankruptcy_and_insolvency_registers`. Accessed at 21 November 2023

Since insolvency actors naturally form a social network of interacting entities, SNA is well-suited for addressing this problem and will be the primary focus of this thesis. Moreover, we emphasize the use of dynamic social networks, i.e., networks that undergo structural changes over time. This approach enables a comprehensive study of interactions over extended periods, allowing us to discern trends in debtors' and creditors' behavior and forecast future developments.

Despite the wealth of information within the IR, this data source has so far remained largely unexplored and underutilized, not only in the realm of data mining but also more broadly in the fields of social sciences, economics, and humanities. This thesis seeks to bridge that gap by harnessing data mining and social network analysis, aiming to unlock the latent potential embedded within this data source. The knowledge extracted using our approach can have important implications for better comprehension of economic systems, identifying patterns of financial distress, and improving decision-making regarding insolvencies and their prevention. Furthermore, our approach is generic enough to apply to data from other insolvency registers and publicly available datasets, assuming the data was preprocessed similarly.

The text is organized as follows. In Chapter 1, we will formulate the main objectives of this thesis. Chapter 2 provides an overview of social network analysis, specifically focusing on methods relevant to this work. Chapter 3 outlines the insolvency process in the Czech Republic, with a particular focus on the IR.

Chapter 4 details the essential steps taken to prepare the *Czech insolvency dataset* (CID), which will serve as the basis for all our experiments. First, we use a relational database to establish the dataset's schema and storage. Subsequently, we outline the process of extracting and merging data from multiple interfaces provided by the IR. The rest of this chapter centers on designing and implementing our custom large-scale document extraction system, named *IRES* (Insolvency Register Extraction System). IRES utilizes Open Source OCR technology to extract data from insolvency documents. In Chapter 5, we introduce the model of a dynamic social network and propose a novel network processing software system, *GraphSlices*, for analyzing such networks. We will use GraphSlices to perform the most critical experiments in the following chapter.

Chapter 6 is dedicated to detailing the experiments conducted in this thesis. Experiment 1 focuses on constructing a static insolvency social network (which is not assumed to change over time) and defining the relationships within such a network. In Experiment 2, we further enhance the insolvency network by considering the timestamps of the relationships (or edges) in the network, enabling us to model insolvencies as a dynamic social network. Subsequently, Experiments 3 and 4 focus on enriching the insolvency network with additional domain-specific data from the insolvency dataset, such as the reasons for the debt and its value. This chapter culminates in Experiment 5, wherein we present our methodology for forecasting the future evolution of the insolvency network.

1. Goals of this thesis

The main goal of this thesis is to use social network analysis (SNA) to understand the indebtedness structure present in the Czech society today. We aim to introduce methods and tools enabling the application of SNA to insolvency-related data, with a particular emphasis on data from the Insolvency Register of the Czech Republic (IR). SNA will allow us to model interactions among insolvency actors before and during the insolvency process, letting us draw new insights into the insolvency dynamics and the root causes leading to bankruptcy. This thesis will focus on insolvency cases initiated in the Czech Republic between January 1, 2008, and December 31, 2022.

The complexity of analyzing data, such as from the IR, arises from the need to use a multifaceted approach. Traditional mathematical models typically rely on tabular data, describing specific observations and subsequently applying statistical models for analysis and inference. However, the tabular form of data fails to capture the dynamically changing interactions among the insolvency actors.

The first crucial aspect involves considering the structural relationships among the insolvency actors. These actors engage with each other before and during the insolvency process, forming a complex network where actors correspond to nodes and relationships among the actors are represented as edges. The structure of such a network can reveal essential patterns, such as a small number of nodes with strong influence over the network or communities of nodes that are more densely interconnected among themselves than with the rest of the network.

The second aspect we need to consider when studying the insolvency data is time. Insolvency outcomes result from historical interactions between debtors and creditors, often spanning years before the actual insolvency event. Understanding the chronological sequence of these interactions is vital for comprehending the factors leading up to the insolvency, its outcome, and future resolution.

Finally, the third and final aspect involves the domain-specific characteristics (or metadata) related to the individual insolvency actors (or nodes) and their relationships (or edges). This includes additional structured metadata, such as demographic information (e.g., age, gender, education) and categorical information about the entity type (e.g., natural person, self-employed, or company). Additionally, there are supplementary data providing insights into the relationships as well. This may encompass structured data, such as the amount of debt owed by the debtor to the creditor, and unstructured data, such as textual descriptions detailing the nature of the debt.

Thus, the goals of this thesis are:

Goal 1: Model the insolvency process utilizing a social network approach.

Goal 2: Model the insolvency process in time utilizing a dynamic social network approach.

Goal 3: Enrich the insolvency network by metadata extracted from the IR.

Goal 4: Predict the future development of the insolvency network.

Goal 1: Model the insolvency process utilizing a social network approach. We aim to propose a strategy for modeling insolvency actors and their relationships using a static social network. The primary challenge lies in the many possibilities of constructing such a social network, and our objective is to determine the fundamental relationships that need to be incorporated into the insolvency network. We seek to explore if even a simple static social network of insolvencies can capture certain aspects of the indebtedness structure in the Czech Republic and provide novel insights into the insolvency process. We plan to utilize the structure of the constructed network to identify the most important nodes and evaluate whether nodes within the network form natural communities, i.e., groups of mutually densely interconnected nodes.

Goal 2: Model the insolvency process in time utilizing a dynamic social network approach. Our primary objective is to understand the main structural changes occurring within the insolvency network over time. To facilitate this analysis, we intend to enhance the static social network from the previous goal by incorporating time information indicating when individual relationships were formed and analyzing distinct snapshots of the insolvency network. We aim to quantify the structural changes in the insolvency network by measuring the network's density over time and examining how the importance of nodes evolves.

Goal 3: Enrich the insolvency network by metadata extracted from the IR. We seek to propose an approach for extending the insolvency network further by incorporating domain-specific information about individual nodes and their relationships, such as the size and nature of the debt. The objective is then to propose methods that can effectively utilize the enhanced structure of the insolvency network to gain deeper insights into the insolvency process. Specifically, the proposed methods should be able to classify debtors based on the nature of their debt, unveiling common patterns and circumstances leading to indebtedness. These methods should also be adept at utilizing the debt size to identify key creditors acting as sources of specific types of debts within the insolvency network, such as mortgages, credit card debts, or utility bills.

Goal 4: Predict the future development of the insolvency network. Our primary objective is to introduce a methodology that integrates findings from the preceding three goals to construct a predictive model capable of anticipating future development in the insolvency network. The proposed model should effectively utilize the insolvency network structure, edge timestamps, and additional domain-specific metadata to predict the future importance of nodes within the network. Furthermore, the proposed methodology also needs to include the interpretation of the constructed model, enabling the identification of key drivers influencing the network's future evolution.

2. Social network analysis

Social network analysis (SNA) is a field of study that investigates relationships and interactions among individuals or entities within a network. Within SNA, networks are defined as graphs where the nodes and edges possess attributes. By studying patterns present in the networks, SNA can reveal the structure and dynamics of complex systems. Utilizing concepts such as nodes, edges, and centrality, it offers insights into information flow, influence, and community structures. Social network analysis is widely applied across many disciplines, including biology, chemistry, sociology, and economics.

The growing interest in social network analysis is driven by the widespread use of internet-enabled devices and digitization, transforming industries globally. The popularity of online platforms like Twitter, Facebook, and Instagram contributes to a wealth of network-centric data, offering insights into diverse scenarios and situations. In this case, social networks can be defined as users communicating with each other or sharing content on platforms like Facebook or Instagram.

Digitization is reshaping not just businesses but also governments. Many governments today actively engage in the growing Open Data[1] initiative and contribute to a rapidly expanding pool of publicly available data. This initiative revolutionizes how governments and institutions communicate with citizens, leading to the transformation of legacy government systems into internet-era counterparts through electronic systems and public registers designed for efficient information exchange.

For instance, electronic justice systems that allow users to access a specific judicial file are quickly becoming widespread in the EU[2]. Nearly every EU country also provides a public company register with information about business entities, owners, and relationships. Similarly, almost every EU country maintains an insolvency register[3], documenting bankruptcies and financial obligations among entities. In these cases, entities and their connections can also be viewed as social networks, as visualized in Figure 2.1. Applying SNA to these types of governmental data can bring a lot of new insights into how society works and evolves.

2.1 Research areas

One of the main aspects of online social networks is that they provide access to not only a large amount of structural data in the form of relationships between entities but also rich data about the entities themselves. This new entity-related data opens new opportunities from data mining and knowledge extraction perspectives. Generally, the type of data analyzed in the context of social networks can be assigned into the following two categories.

[1]EU Open Data Portal: `https://data.europa.eu`

[2]Overview of electronic judicial systems in the EU: `https://e-justice.europa.eu/content_judicial_systems-14-en.do`. Accessed 21 November 2023

[3]The full list of insolvency registers available within EU: `https://e-justice.europa.eu/110/EN/bankruptcy_and_insolvency_registers`. Accessed 21 November 2023

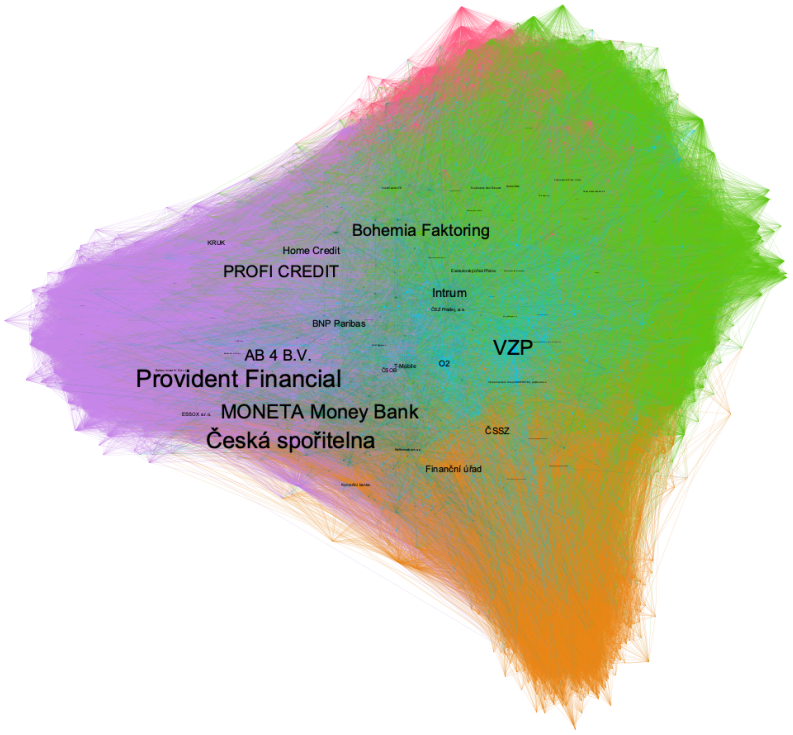


Figure 2.1: Visualization of debtor-creditor interactions as a social network constructed using data from the IR of the Czech Republic. The colors represent different communities within the network obtained by applying a network clustering algorithm (Section 2.4).

Structural analysis focuses solely on the linkage patterns in the network in order to determine important nodes, communities, and evolving regions in the network. Such analysis provides an excellent overall view of the behavior in the studied network. Since it only relies on structural information, it transfers well from one type of network to another.

Content-based analysis builds on top of the structural analysis by incorporating additional content available in online social networks. Social networks such as message networks (Messenger, Whatsapp, Viber), Youtube, Facebook, or Instagram contain a large amount of textual, image, and video data shared by their users (or nodes in the network). Typically, content-based analysis combined with structural analysis provides more effective results in many applications. For instance, community detection incorporating textual content is much more precise in providing information about the topics relevant to individual communities.

Another important distinction in the field of social network analysis is between *static analysis* and *dynamic analysis*. In static analysis, it is assumed that the social network either does not change over time or only changes slowly. Such analysis is usually performed on the whole network or a small number of snapshots separately. An example of a very slowly evolving network is the bibliographic network. Other networks, such as messaging networks, are much more dynamic as they evolve almost continuously. The analysis of such networks often leads to analysis using a large number of snapshots or even network streams [1].

The most well-known structural problem in social network analysis is *community detection*, which focuses on identifying structurally related groups in the network. These groups are referred to as communities. The community detection problem is relevant for analyzing both static and dynamic social networks. Some of the popular community detection algorithms are the Kernighal-Lin algorithm [2], the Girvan and Newman's algorithm [3], and the MCL algorithm [4].

From the communication perspective, social networks can also be looked at as structures that enable the spread of information through mutual interactions. For instance, interactions between entities propagate essential news through the network. The study of social interactions and how they affect the dissemination of information is an important problem in social network analysis. A closely related problem is one of influence analysis in social networks. This research focuses on identifying the most influential members in the network and how their influence is propagated through the network. The most influential members in the network can be determined by using either flow models [5] or by using page rank style methods [6], which identify the most well-connected nodes in the social network.

The last structural problem, which is especially important in the context of this study, is the inference of links that are not known yet in the social network. This problem can also be referred to as *link prediction* [7]. The problem of link prediction is instrumental in the context of dynamic analysis for predicting future linkages in the studied social networks. The future linkages can provide us with an idea of future relationships and, thus also, future development in the network.

All of the applications above can be significantly enhanced by incorporating content information. For instance, it has been shown that content associated with nodes can significantly improve the quality of detected communities in the social network [8]. This improvement stems from the fact that content information and the network structure are often closely related. It has also been shown that using content information can improve results for link prediction [9]. Incorporating domain-specific content in the context of dynamic social network analysis will also be one of the main aspects of this study.

2.2 Statistical properties of social networks

Properties of large social networks were first studied in the static context, which led to the discovery of many interesting properties of real-world social networks. One such property is the *small world phenomenon* [10], which suggests that any two individuals in the world are, on average, connected by a surprisingly short chain of acquaintances, typically around six degrees of separation. Another important property is the power-law degree distribution of nodes [11, 12]. Even though time-evolving or dynamic graphs have attracted attention only recently, their analysis already led to the discovery of many additional properties. For example, the shrinking diameters and the so-called densification power law described in [13]. Further studies have focused on the properties of networks with multiple edges between the same nodes (multigraphs) or edge weights.

In this section, we will describe structural patterns that are common for social networks. We will first formally define a social network as a graph. Then we will focus on the most important statistical properties of static graphs. Afterward, we will move our attention to dynamic graphs. We will also treat social networks represented as weighted and unweighted graphs separately.

2.2.1 Definitions

Graphs. A social network can be represented by a *graph*. In the rest of this chapter, we will use the terms *network* and *graph* interchangeably.

A static, unweighted graph G consists of a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E} : G = (\mathcal{V}, \mathcal{E})$. We will denote the sizes of $\mathcal{V}$ and $\mathcal{E}$ as N and E, respectively. A graph can be either *directed* or *undirected*.

Graphs may also be *weighted*, and there may be multiple edges occurring between two nodes (e.g., multiple e-mails sent), or a specific weight may be associated with an edge (e.g., monetary value for a specific transaction). In a weighted graph $\mathcal{G}$, let $e_{i,j}$ denote the edge between node i and node j. These two nodes will be referred to simply as *neighboring nodes* or *incident nodes* of edge $e_{i,j}$. Let $w_{i,j}$ denote the weight associated with edge $e_{i,j}$. The total weight w_i of node i is defined as the sum of weights of all its incident edges, i.e., $w_i = \sum_{k=1}^{d_i} w_{i,k}$ where d_i denotes the node's degree. All the notations used in this chapter are summarized in Table 2.1.

Lastly, graphs may be *unipartite* or *multipartite*. Most real-world social networks are unipartite, i.e., all nodes belong to a single class, and an edge can be drawn between any two nodes in the graph. In multipartite graphs, there are multiple classes of nodes, and an edge exists only between nodes from different classes. The simplest example is a bipartite graph which consists of two disjoint sets of nodes $\mathcal{V}_1$ and $\mathcal{V}_2$, and an edge exists only between nodes from these two different classes. An example of a bipartite graph can be a movie-actor graph generated from the IMDB database[4] consisting of two separate classes of nodes: actors and movies.

There are multiple representations commonly used for graphs, including a visual representation, list of edges, or an *adjacency matrix* $\boldsymbol{A}$. In an adjacency matrix, rows and columns correspond to nodes in the graph, and entries in the matrix indicate the existence of edges. For unweighted graphs, all entries are either a 0 or a 1, and for weighted graphs, the adjacency matrix contains number values representing edge weights.

Components. A *connected component* in a graph is a set of nodes and edges where a path exists between any two nodes in the set. For directed graphs, we refer to these types of connected components as *weakly connected components*, while a *strongly connected component* requires a directed path between any two nodes in the set. The forming of a single *giant connected component* is common for real-world social networks [13].

[4]IMDB is an online database of information related to films (`https://www.imdb.com/`).

Notation	Description
$\mathcal{G}$	Graph representation of a dataset
$\mathcal{V}$	Set of nodes for graph $\mathcal{G}$
$\mathcal{E}$	Set of edges for graph $\mathcal{G}$
N	Number of nodes, corresponds to $\lvert\mathcal{V}\rvert$
E	Number of edges, corresponds to $\lvert\mathcal{E}\rvert$
$e_{i,j}$	Edge between node i and node j
$w_{i,j}$	Weight of edge $e_{i,j}$
$deg(i)$	Degree of node in an undirected graph i
$deg^{out}(i)$	Out-degree of node i in a directed graph, i.e. the number of edges pointing out from i
$deg^{in}(i)$	In-degree of node i in a directed graph, i.e. the number of edges pointing to i
$dist(i,j)$	Distance between node i and j measured as the number of edges along the shortest path.
w_i	Weight of node i (sum of weights of all incident edges)
$\boldsymbol{A}$	The adjacency matrix of a graph. The entry $\boldsymbol{A}_{ij}$ is nonnegative and corresponds to edge weight w_{ij}. For unweighted graphs $\boldsymbol{A}_{ij}$ is 1 for all edges.
λ_1	Principal eigenvalue of an unweighted graph
$\lambda_{1,w}$	Principal eigenvalue of a weighted graph

Table 2.1: Notation used to describe social networks.

Diameter. For a given graph, the *diameter* is defined as the maximum *distance* between any two nodes, where the distance is the minimum number of edges that must be traversed on the path between nodes (ignoring directionality). Measuring a graph's diameter, especially over time, can be used to understand how large components evolve, for instance, whether they are growing or shrinking.

Heavy-tailed distributions. Even though the Gaussian distribution is common in nature, there are many domains where the probability of events far from the mean is significantly higher than the Gaussian would allow. As we will see in the following sections, social networks are precisely one of the areas where this occurs. The so-called *heavy-tailed* distributions model such events, and while traditional exponential distributions have a bounded variance, i.e., large deviations from the mean are almost impossible. For heavy-tailed distributions, the probability of an event $p(x)$ decays polynomially rather than exponentially as $x \to \infty$. As a result, these types of distribution have a "fat tail" for extreme values in their corresponding PDF (probability density function) plots.

Let us define a generic continuous distribution in the form:

$$p(x) = Cf(x) \tag{2.1}$$

where $f(x)$ is the basic functional form of a specific distribution and C is the appropriate normalization constant such that $\int_{x_{min}}^{\infty} Cf(x)\,dx = 1$.

Using this generic form, we will now define the power law, a heavy-tailed distribution typical for social networks:

$$\begin{aligned} f(x) &= x^{-\alpha} \\ C &= (\alpha - 1)x_{min}^{\alpha-1} \\ \alpha &> 1, x \geq x_{min} \end{aligned} \tag{2.2}$$

2.2.2 Static properties

Next, we will review the *static* properties common in most real-world social networks. Some of the networks we will mention in this section evolve over time. However, the properties are always measured on a single graph snapshot. Among the most common patterns occurring in static networks are: degree distributions, minimal path length in terms of the number of edges between pairs of nodes, and communities. Next, we will describe all these patterns in more detail.

Heavy-tailed degree distribution. The degree distribution in most real-world social networks is governed by a power law in the form $f(d) \propto d^{-\alpha}$, with $\alpha > 1$, where $f(d)$ corresponds to the fraction of nodes with degree d. These types of power law relations have been reported in many studies, including [12, 14, 15]. Intuitively, a power-like distribution in a graph indicates that the vast majority of nodes have small degrees, and just a small fraction of the remaining nodes have large degrees.

Small diameter. One of the most interesting patterns common for real-world networks is that they often have a small diameter, which is often referred to as the small-world phenomenon. The graph diameter intuitively tells us how quickly we can get from one end of the network to the other. Many real-world graphs that have been studied extensively have very small diameters. For instance, only 19 was measured for the Web [16] and only 6 for social networks [17]. Since the diameter is defined as the maximum-length shortest path between all possible pairs, it is often easily highjacked by long chains in the graph. Therefore, reporting the effective diameter is more common. This more robust metric reports a specific percentile of the pairwise distances among all reachable pairs of nodes.

Community structure. Lastly, real-world graphs often exhibit a modular structure in which nodes form groups and possibly groups form larger groups [18, 19]. Consequently, this leads to the formation of communities where groups of nodes in the same community are more tightly connected among each other than with nodes outside of the community. The quantitative measurement for such structure is called *modularity*, and it was first introduced in [3].

2.2.3 Dynamic properties

Next, we present the most important dynamic properties observed in real-world evolving graphs. These are typically studied by examining a series of

static snapshots and comparing measurements across these snapshots. The most notable patterns described in this section are shrinking diameter, the densification law, and the largest eigenvalue law.

Shrinking diameter. It was shown by Leskovec et al.[13] that not only is the diameter of real-world graphs small, but it also tends to shrink over time until it reaches an equilibrium. This pattern occurs due to the so-called *gelling point* and *densification* typical for real-world graphs (described in more detail below). The gelling point indicates a moment when many small components in the graph merge and form one giant component. During the "coalescence" of the graph, the diameter first spikes, and then as new edges are added to the graph, it tends to keep shrinking until it reaches an equilibrium.

Densification power law. Let $E(t)$ and $N(t)$ denote the number of edges and nodes for a specific graph at time t. Time-evolving graphs are governed by the so-called "Densification power law" that can be expressed as $E(t) \propto N(t)^{\beta}$ for all time ticks t [13], where β is the densification exponent. The densification exponent was measured to be between 1.03 and 1.7 on several real-world graphs. Intuitively, this law tells us that graphs are getting denser over time, explaining the shrinking diameter described earlier.

2.3 Measuring node importance

One of the central tasks in the structural analysis of social networks is the identification of important nodes. Intuitively, important nodes in the network are nodes that are linked with other nodes extensively. For instance, in an organization, a person with extensive contacts who communicates with many other people is generally considered more important than someone with significantly fewer contacts. Following this simple intuition, several importance measures have been proposed, collectively denoted as *centrality measures*. The calculation of centrality measures uses simple properties such as the number of edges connected to the node (i.e., the degree) and/or counting the number of shortest paths in the graph crossing the considered node.

As we will show, centrality-based measures have several shortcomings, one of which is that when the importance calculation of a single node does not consider the importance of nodes connected to the measured node. Such an approach requires a recursive definition of importance and leads to more sophisticated measures relying on random walks in graphs. Furthermore, this approach can be used to frame the measuring of importance more generally as a problem of ranking entities in the graph. The ranking can be used to define graph theoretic measures of similarity that have applications in many areas, including link prediction, graph search techniques, and collaborative filtering for recommender networks. Random walks provide a simple framework for unifying information from ensembles of paths between two nodes, leading to more robust similarity measures.

Here is the organization of this section: First, we will start by describing simple centrality-based measures for node importance and show their shortcomings. Then, we will provide the mathematical background for random walks on graphs. Finally, we will define several important measures using the random walks theory, including PageRank, HITS, and Katz measure.

2.3.1 Centrality

A *central node* is a node that is connected to many other nodes in the graph. There are different ways to quantify the connectedness of a node to the rest of the network. Therefore, several types of centralities can be defined for both undirected and directed graphs. In this section, we will cover the most relevant types of centralities.

Degree. The *degree centrality* is the simplest one and relies mainly on the degree of the node in the graph. For an undirected graph, we can define the degree centrality of node i, denoted by $C_D(i)$, as follows:

$$C_D(i) = \frac{deg(i)}{N-1} \tag{2.3}$$

The value of $C_D(i)$ is between 0 and 1 since $N-1$ is the maximum possible value of $deg(i)$. For directed graphs, the *degree* centrality is usually defined based on the out-degree of nodes only:

$$C'_D(i) = \frac{deg^{out}(i)}{N-1} \tag{2.4}$$

Closeness centrality. The other way to view centrality is based on how close a node is to all the other nodes in the graph. Intuitively, a central node is relatively close to all (or at least the majority) nodes. To define the closeness, we will use the distance measured as the number of edges along the shortest path between two nodes i and j. The *closeness centrality* $C_C(i)$ for node i in an undirected graph is defined as:

$$C_C(i) = \frac{N-1}{\sum_{j=1}^{N} dist(i,j)} \tag{2.5}$$

The value of this measure is also between 0 and 1, as $N-1$ is also the minimum value of the sum of shortest distances from i to all the other nodes. Note that this equation only makes sense for connected graphs.

For directed graphs, the *closeness centrality* is defined by the same equation, it is only necessary to consider the direction of links in the distance computation, and the graph must be strongly connected.

Betweenness centrality. If two non-adjacent nodes j and k want to interact through their edges, and node i is on the shortest path between j and k, then i might have some control over their interactions. The *betweenness centrality*

quantifies the control of node i over other pairs of nodes in the graph. Intuitively, if node i is on the shortest paths of many such interactions then it should have a high betweenness centrality.

Let $p_{jk}(i)$ be the number of shortest paths between nodes j and k that pass node i. The notation p_{jk} without specifying the target node is the total number of shortest paths between j and k. The betweenness centrality C_B is then defined as:

$$C_B(i) = \sum_{j<k} \frac{p_{jk}(i)}{p_{jk}} \tag{2.6}$$

There might be multiple shortest paths between nodes j and k, while some may pass i and others may not. $C_B(i)$ has a minimum value of 0 when i is not a part of any shortest paths. Its maximum value is $(N-1)(N-1)/2$, the number of pairs of nodes not including i. We can use this maximum value as a normalization factor to make sure the final betweenness centrality value is between 0 and 1:

$$C'_B(i) = \frac{2\sum_{j<k} \frac{p_{jk}(i)}{p_{jk}}}{(N-1)(N-2)} \tag{2.7}$$

We need to make two additional adjustments to make Equation 2.7 work for directed graphs. First, it must be multiplicated by 2 because there are now $(N-1)(N-2)$ pairs to consider since a path from j to k and k to j are now different. Secondly, p_{jk} must consider the direction of edges when calculating the shortest paths.

2.3.2 Random walk based measures

Centralities, such as those defined above, provide simple measures to identify important nodes in the network. As was shown in the case of betweenness and closeness, it is useful to consider a broader neighborhood of a node in the graph when estimating its importance. Centralities solved this by employing shortest-path statistics, which come with several shortcomings. First, all centrality measures assumed that each node was equally important during calculation. However, intuitively, we expect connections to other important nodes to be weighted more than connections to unimportant nodes. For instance, let us imagine a personal website of someone who creates an arbitrary number of new websites that point to it. A website such as this should naturally have a low importance score. Secondly, we want to include more than just the shortest paths in the calculation but all possible paths with different probabilities. We need a more robust mathematical apparatus called *random walk on graphs* to fix these shortcomings.

In the first part of this section, we will provide the mathematical background of random walks. Then, we will define some essential random walk-based measures such as PageRank, HITS, and others, some of which we will use in this work extensively.

Random walks on graphs: Background

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined[5] as a set of vertices $\mathcal{V}$ and edges $\mathcal{E}$. The ij-th entry of the adjacency matrix $\boldsymbol{A}$ is non-negative and corresponds to the weight $w_{i,j}$ assigned to edge $e_{i,j}$. For unweighted graphs, the weight of any edge is 1. Let $\boldsymbol{D}$ be an $N \times N$ diagonal matrix, where $\boldsymbol{D_{ii}} = \sum_j \boldsymbol{A}_{ij}$. We will denote $\boldsymbol{D}_{ii}$ also as the degree $deg(i)$ of node i. The Laplacian $\boldsymbol{L}$ of graph $\mathcal{G}$ is defined as $\boldsymbol{D} - \boldsymbol{A}$, which is also commonly used in machine learning algorithms as a regularizer [20].

Next, let $\boldsymbol{P} = p_{ij},\ i, j \in V$ denote the transition probability matrix, so that $\boldsymbol{P}_{ij} = \frac{\boldsymbol{A}_{ij}}{\boldsymbol{D}_{ii}}$ if $(i, j) \in \mathcal{E}$ and 0 otherwise. A *random walk* on graph $\mathcal{G}$ is a Markov chain [21] with transition probabilities specified by matrix $\boldsymbol{P}$. For undirected graphs, $\boldsymbol{A}$ is symmetric, and $\boldsymbol{L}$ is symmetric positive semi-definite. We will denote the set of neighbors of a node i by $\mathcal{N}(i)$. For unweighted graphs, $deg(i)$ is the size of this neighborhood. For directed graphs, $\mathcal{I}(i)$ will denote the set of nodes with a directed edge pointing to node i. Similarly, $\mathcal{O}(i)$ denotes the out-neighbors, i.e., the set of nodes pointed to from node i. Both in and out-degrees are assumed to be weighted for weighted graphs.

Let us now assume to start a random walk and let node v_0 be chosen from a distribution x_0 (over all nodes). In the following steps, we choose from distributions $x_1, x_2, \ldots$, which generally differ from each other. However, once $x_t = x_{t+1}$, then we say that x_t is the stationary distribution for graph $\mathcal{G}$. According to the Perron-Frobenius theorem, a unique stationary distribution exists if $\boldsymbol{P}$ is irreducible and aperiodic.

Importance measures

Pagerank [22] is probably the most well-known method, especially for ranking web pages. Suppose the world-wide-web is considered a graph, with web pages representing nodes and URLs representing edges pointing from one web page to another. In that case, PageRank is defined as the distribution that satisfies the following linear equation:

$$\boldsymbol{v} = (1 - \alpha)\boldsymbol{P}^T\boldsymbol{v} + \frac{\alpha}{n}\mathbf{1} \tag{2.8}$$

Parameter α is the probability of restarting the random walk at a given step, also called the *damping factor*. Note that the damping factor α plays a vital role. The Perron-Frobeniuns theorem indicates that a stochastic matrix $\boldsymbol{P}$ has a unique principal eigenvector if and only if it is irreducible and aperiodic. The random restart assures that: 1) all nodes are reachable from all other nodes during the random walk and 2) the underlying Markov chain is aperiodic. The damping factor also makes the second-largest eigenvalue upper bounded by α [23].

One of the first alternatives to PageRank introduced by Kleinberg [24] is the Hubs and Authorities (HITS) algorithm. The fundamental intuition behind HITS is that generally, there are two types of important web pages on the world-wide-web: authorities and hubs. The authorities are pages that represent good sources

[5] In this section, we will continue to use the notation defined in Table 2.1.

of information and therefore are linked to from many other pages, i.e., tend to have a larger in-degree. On the other hand, a hub links to many other websites (often authorities) and has a large out-degree. During the computation of HITS, the algorithm assigns two numbers to each node: an authority score and a hub score. This leads to an iterative algorithm that uses two vectors $\boldsymbol{h}$ and $\boldsymbol{a}$, with updates defined as follows:

$$\boldsymbol{a}(i) = \sum_{j \in \mathcal{I}(i)} \boldsymbol{h}(j) \tag{2.9}$$

$$\boldsymbol{h}(i) = \sum_{j \in \mathcal{O}(i)} \boldsymbol{a}(j) \tag{2.10}$$

Both vectors from the previous equation are normalized to have unit length. Assuming $\boldsymbol{A}$ is the unweighted adjacency matrix of the considered graph, the last two equations can also be rewritten as follows:

$$\boldsymbol{a} = \boldsymbol{A}^T \boldsymbol{h} \tag{2.11}$$

$$\boldsymbol{h} = \boldsymbol{A}\boldsymbol{a} \tag{2.12}$$

The previous equations mean that $\boldsymbol{h}$ converges to the principal eigenvector of AA^T, whereas $\boldsymbol{a}$ converges to the principal eigenvector of A^TA.

Similarity measures

In this section, we will generalize the ideas we used to define importance measures and adopt the same principles to define similarity (or proximity) measures between nodes in the graph. These similarity measures will also be helpful in the context of link prediction, which will be discussed in Section 2.6.

Simrank is a similarity measure introduced in [25], and it is based on the intuition that two nodes are similar if they share many neighbors. Formally, Simrank between nodes a and b is defined as:

$$s(a,b) = \frac{\gamma}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{i \in \mathcal{I}(a), j \in \mathcal{I}(b)} s(i,j) \tag{2.13}$$

This equation corresponds to two backward random walks that start simultaneously from nodes a and b. Therefore the value of $s(a,b)$ can be interpreted as the expectation of γ^l, where l equals the time when those two walks meet.

Katz Score [26] is a similarity measure based on the ensemble of paths between nodes and is formally defined as:

$$Katz(i,j) = \sum_{l=1}^{\infty} \beta^l \boldsymbol{A}^l_{i,j} \tag{2.14}$$

In this equation, $\boldsymbol{A}^l_{i,j}$ is the number of paths of length l between nodes i and j. β is the so-called *attenuation* factor, which works as a penalization factor for distant neighbors. To compute the *Katz score*, solving a system of linear equations over the adjacency matrix is necessary. A very small β yields Katz scores that mostly return nodes with many common neighbors, whereas larger values of β emphasize

longer paths. There is also a weighted version of the Katz score, which considers the weights assigned to the edges [7].

The *Adamic/Adar* score introduced in [27] is similar to the common neighbors measure except that it weighs high degree neighbors less. The Adamic/Adar score is defined as:

$$Adamic/Adar(i,j) = \sum_{k \in \mathcal{N}(i) \cap \mathcal{N}(j)} \frac{1}{log(|\mathcal{N}(k)|)} \tag{2.15}$$

The *Jaccard coefficient* computes the probability that two nodes will have a common neighbor and is defined as:

$$J(i,j) = \frac{|\mathcal{N}(i) \cap \mathcal{N}(j)|}{|\mathcal{N}(i) \cup \mathcal{N}(j)|} \tag{2.16}$$

It was shown that the matrix of Jaccard coefficients is positive definite [28].

2.4 Community detection and clustering

Many real-world problems can be effectively modeled as social networks, where nodes represent entities of interest and edges correspond to relationships between them. It is a very generic framework which is also why we find social network analysis applied in many different domains, for instance – biology [29, 30], ecology [31, 32], engineering [33, 34], linguistics [35], social science [36], and many other.

It was quickly observed that even though the networks in these empirical studies arise in entirely different contexts, they still share many essential concepts and properties. One particular interest that arose from these studies was the uncovering and understanding of community structures present in these different networks. The communities can often be identified across multiple topological and temporal scales.

Community detection discovery is challenging for several reasons. First, the topological properties of most networks are very complex and often coupled with uncertainties arising from the underlying data [37]. Second, networks' natural dynamic aspects must be incorporated into the solutions. Here, by *dynamic*, we refer to any network that changes over time. These dynamic networks include time-evolving networks and networks that change due to external factors (e.g., intelligence networks that are affected by credibility issues). Finally, the solutions have to be able to scale to massive graphs that can involve millions of nodes and billions of edges [38].

This section will be organized as follows. In Section 2.4.1, we will provide a more formal definition of a community and quality functions associated with evaluating communities. In the remaining sections, we will discuss the core methods for community discovery proposed in the literature to date. Every section will focus on a particular group of algorithms.

2.4.1 Quality functions

Informally, a community in a network is a group of nodes that has a larger number of ties internally than it has to the rest of the network. This intuitive definition was formalized in several ways, usually through a *quality function*, which quantifies the "goodness" of a given network division into communities. We will describe some of the quality functions used in the literature to date.

The *normalized cut* [39] of a set of vertices $S \subset \mathcal{V}$ is defined as:

$$Ncut(S) = \frac{\sum_{i \in S, j \in \overline{S}} \boldsymbol{A}(i,j)}{\sum_{i \in S} deg(i)} + \frac{\sum_{i \in S, j \in \overline{S}} \boldsymbol{A}(i,j)}{\sum_{i \in \overline{S}} deg(i)} \tag{2.17}$$

where $\overline{S} = \mathcal{V} - S$. In other words, the normalized cut of a set of nodes S equals the sum of weights of the edges that connect S to the rest of the graph, normalized by the total edge weight of S and $\overline{S}$. Subsets of nodes with lower normalized cut values represent good communities since they are well connected within the subset S but are only sparsely connected to the rest of the graph.

The *conductance* [40] of a set of vertices $S \subset \mathcal{V}$ is similar to the normalized cut and is defined as:

$$Conductance(S) = \frac{\sum_{i \in S, j \in \overline{S}} \boldsymbol{A}(i,j)}{min(\sum_{i \in S} deg(i), \sum_{i \in \overline{S}} deg(i))} \tag{2.18}$$

Let $V_1, \ldots, V_k$ be the division of the graph into k subsets (or clusters), then the normalized cut (or conductance) is the sum of the normalized cuts (or conductances) of each subset $V_i,\ i = 1, \ldots, k$.

The *Kernighan-Lin (KL) objective* [2] minimizes the edge cut, i.e., the sum of the inter-cluster edge weights, under the constraint that all clusters have the same size (assuming the size of the network is a multiple of the number of clusters). The KL objective is defined as:

$$KLObj(V_1, \ldots, V_k) = \sum_{i \neq j} \boldsymbol{A}(V_i, V_j), \text{ subject to } |V_1| = |V_2| = \ldots = |V_k| \tag{2.19}$$

where $\boldsymbol{A}(V_i, V_j) = \sum_{u \in V_i, v \in V_j} \boldsymbol{A}(u,v)$.

The most popular measure today is *modularity* [3]. One of its advantages is that it is independent of the number of clusters the graph is divided into. Modularity is based on the intuition that the farther the subgraph of each community is from a random subgraph (or the null model), the better the discovered community structure is. For a division of the graph into k clusters $V_1, \ldots, V_k$ modularity Q is given by:

$$Q = \sum_{c=1}^{k} \left[\frac{A(V_i, V_i)}{E} - \left(\frac{deg(V_i)}{2E} \right)^2 \right] \tag{2.20}$$

where E is the number of edges in the graph and $deg(V_i)$ is the total degree of the cluster V_i. For each cluster, we compute the difference between the fraction of edges within the cluster and the fraction of edges we would expect in a random cluster with the same total degree.

Optimizing any of these objective functions is NP-hard as was shown in [41, 42].

2.4.2 The Kernighal-Lin (KL) algorithm

The KL algorithm [2] is one of the oldest partitioning algorithms proposed in the 1970s. It aims to optimize the KL objective function as defined in Equation 2.19, i.e., minimize the edge cut while keeping the cluster sizes balanced. The basic algorithm is iterative, and to explain how it works, let us first assume we only want to partition the nodes into two equally large sets A and B. The algorithm starts with an arbitrary assignment of all nodes to A or B. The only requirement is $|A| = |B|$. Next, the algorithm maintains and improves the partitioning in each pass using a simple greedy procedure. This greedy procedure pairs up vertices from A with vertices from B so that switching the partitions of these two nodes would reduce the edge cut. Once all the vertices are paired up, a subset of them is chosen to accomplish the best overall reduction of the edge cut. Each iteration of the original KL algorithm has a $O(|E|log(|E|)$ complexity. The above algorithm can be generalized to handle more than two partitions.

2.4.3 The Agglomerative/Divisive algorithms

Agglomerative algorithms begin with each node in the graph assigned to its own community, and each iteration of the algorithm merges sufficiently similar communities. This process is continued until the desired number of communities is reached, or the remaining communities are too dissimilar to merge. Divisive algorithms operate precisely in the opposite way. They begin with the entire graph assigned to a single community, and each iteration of the algorithm selects the most suitable community to be divided into two. Both algorithms generate a hierarchical clustering and output a *dendrogram* as a binary tree. Each leave in the dendrogram corresponds to a node in the network, and each internal node is a community. In the case of divisive algorithms, a parent-child relationship indicates that the community represented by the parent node was divided to obtain the communities in the child nodes. In the case of agglomerative algorithms, the parent-child relationship indicates that the communities represented by the child nodes were merged to obtain the community in the parent node.

Girvan and Newman's algorithm. The algorithm proposed by Newman and Girvan [3] is a divisive algorithm for community detection which uses *edge betweenness*. Edge betweenness is another centrality measure similar to node betweenness defined in Section 2.3.1. The edge betweenness for each edge is defined as the number of the shortest paths that go through that edge. The shortest paths are calculated between each pair of nodes in the network. Intuitively, edges with high betweenness scores are more likely to represent edges that connect different communities. More specifically, inter-community edges should have higher edge betweenness scores than intra-community edges. Hence, one can split the social network into its communities by iteratively identifying edges with high betweenness and removing them from the graph.

The original Girvan and Newman's algorithm follows these four steps:

1. Calculate the edge betweenness score for all edges in the network.

2. Find the edge with the highest score and remove it from the network.

3. Recalculate betweenness for all remaining edges.

4. Repeat from step 2.

The procedure above is continued until a sufficient number of communities is obtained. The hierarchical nesting of the communities (dendrogram) is obtained as the by-product of this process. The main disadvantage of this method is the relatively high computational cost. Computing the betweenness for all edges takes $O(|V||E|)$ time, and the algorithm requires $O(|V|^3)$ time overall.

Newman's greedy optimization for modularity. In [43], Newman proposed a greedy agglomerative clustering based on modularity optimization. The basic idea behind this algorithm is that at each stage, groups of vertices are successively merged to form larger and larger communities. The modularity of the division of the network increases after each merge. At the start, each node in the network belongs to its own community, and at each step, the algorithm chooses the two communities whose merging will lead to the largest increase in modularity. By using efficient data structures, this algorithm can achieve a time complexity of $O(|E|log(|V|))$ [44].

2.4.4 Spectral algorithms

In general, spectral methods refer to algorithms that assign nodes to communities based on the eigenvectors of the graph-related matrices, e.g., the adjacency matrix. The top k eigenvectors represent an embedding of the nodes as points in a k-dimensional space. On this k-dimensional space, one can use classical data clustering techniques such as K-means to derive the final assignment of nodes to clusters [45]. The key idea behind these techniques is that the low-dimensional representation obtained by the top eigenvectors exposes the community structure in the graph.

The Laplacian matrix $\boldsymbol{L}$ is the most common matrix used by spectral clustering algorithms. Let $\boldsymbol{A}$ be the adjacency matrix of the network, and let $\boldsymbol{D}$ be the diagonal matrix with degrees of the nodes on the diagonal. The unnormalized Laplacian matrix is given by $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$. The normalized Laplacian denoted by $\boldsymbol{\mathcal{L}}$ is given by $\boldsymbol{\mathcal{L}} = \boldsymbol{D}^{-\frac{1}{2}}(\boldsymbol{D} - \boldsymbol{A})\boldsymbol{D}^{-\frac{1}{2}} = \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$, where $\boldsymbol{I}$ is the identity matrix. It can be shown that both $\boldsymbol{L}$ and $\boldsymbol{\mathcal{L}}$ are symmetric and positive definite and therefore have real and positive eigenvalues [45]. The number of 0 values among the eigenvalues of the Laplacian matrix is equal to the number of connected components in the graph. Additionally, the eigenvector corresponding to the smallest non-zero eigenvalue of $\boldsymbol{\mathcal{L}}$ is known as the *Fiedler vector* [46], which usually forms the basis for bi-partitioning graphs.

The main disadvantage of spectral algorithms lies in their computational complexity. Most existing implementations of spectral algorithms rely on the *Lanczos algorithm* for computing the eigenvector approximations. The Lanczos algorithm is an iterative algorithm that performs a series of matrix-vector multiplications

in each step to obtain successive approximations of the eigenvectors. The complexity for computing the top eigenvector is $O(lM(m))$, where l is the number of matrix-vector multiplications and $M(m)$ is the complexity of each multiplication, which depends primarily on the number of non-zeros m in the matrix. The value of l depends on the specific properties of graphs, such as the *spectral gap*, i.e., the difference between the current eigenvalue and the next. The smaller the spectral gap is, the larger number of matrix-vector multiplications are necessary for convergence. Regarding applications, spectral clustering is hard to scale up to networks with more than tens of thousands of vertices. That is, without employing some parallelization techniques.

2.4.5 Markov clustering

The Markov clustering algorithm (MCL) was proposed by Stijn Van Dongen [4] and is based on the manipulation of the transition probability matrix $\boldsymbol{M}$ (also denoted as the stochastic matrix) of the underlying graph. For the rest of this section, let us refer to the transition probability between nodes as the *stochastic flow*. The MCL algorithm consists of two alternating operations on the stochastic matrix, namely $Expand$ and $Inflate$. $Expand(\boldsymbol{M})$ simply performs $\boldsymbol{M} \times \boldsymbol{M}$ and $Inflate(\boldsymbol{M}, r)$ raises each entry of the matrix $\boldsymbol{M}$ to the inflation parameter r, $r > 1$. The inflation step is followed by the re-normalization of matrix $\boldsymbol{M}$ so that it again represents a valid stochastic matrix, i.e., the columns of $\boldsymbol{M}$ are made to sum to 1. The algorithm starts with the initial stochastic matrix, and then each iteration applies the expansion and inflation steps to the matrix. These steps are repeated until the process converges.

The Expand step spreads the stochastic flow out of a vertex to new vertices. Vertices that are reachable by multiple paths are naturally enhanced more during this process. Intuitively, within-cluster stochastic flows are magnified since there should be more paths between nodes in the same cluster than between nodes in different clusters. The inflation step introduces non-linearity into the process to reinforce the intra-cluster flow even further and reduce the inter-cluster flow. As a result, all the nodes within a cluster stochastically flow into one *attractor* node. We can also identify individual clusters in the graph by identifying the attractor nodes.

MCL is especially popular in bioinformatics as it was shown to be very effective at clustering biological networks [47]. However, MCL comes with two significant downsides [48]. First, MCL is relatively slow and does not scale to large graphs, primarily because of the Expand step, which involves a matrix-matrix multiplication that is very time-consuming. Furthermore, many entries in the stochastic flow matrix have not been pruned out in the first few iterations, so optimization is limited. Second, MCL often produces imbalanced clusters, usually one very large cluster with many small ones.

To address these issues, several new variants of MCL have been proposed recently [48, 49]. First, regularization was introduced to MCL to ensure that the stochastic flows of neighboring nodes are considered when updating each node's flows. In this case, the expand step is replaced by the *Regularize* step, which is

$M = M \times M_G$, where M_G is the stochastic matrix of the original graph. The inflation step remains the same. These improvements lead to further enhancing within-cluster flows. Multi-level regularized MCL (or MLR-MCL) employs regularized MCL in a multi-level framework, as described in the previous section. MLR-MCL solves the scalability issues of the original MCL algorithm since the initial iterations of the algorithm can be performed on a smaller graph. When the matrices become sparse enough, the underlying graph can be uncoarsened.

2.5 Evolution in social networks

All the methods described so far were working under the assumption that the underlying graph does not change or that they are being applied to a single snapshot of an evolving graph. One of the most recent areas in social network analysis that has gained a lot of traction in the past several years is the study of evolving networks. The interest in this area is driven mainly by the abundance of data that can be viewed as an evolving social network. These data sources include interaction data (likes or follows) from platforms like Facebook and Twitter and messaging networks such as Whatsapp and Messenger.

Informally, evolution refers to a change that manifests itself across the time dimension. The field of *Data Mining* defines a clear distinction between mining static data and mining data streams. The stream paradigm usually works under the assumption that instances of data arrive in sequential order, and each instance is seen only once [50]. Therefore, stream mining algorithms in social network analysis often work by adapting or monitoring the underlying structures, such as communities or node importance.

This section is organized as follows. First, we will introduce the stream paradigm computation, which we will use as the basis for the rest of this section. Then, we will focus mainly on the problem of community detection in evolving social networks. The communities can be perceived as clusters built at each point in time. In such cases, the analysis of community evolution involves tracing the same community/cluster at consecutive time points and identifying changes (Section 2.5.2). Communities can also be perceived as smoothly evolving structures when community monitoring involves learning models that adapt smoothly from one time point to the other (Section 2.5.3).

Other tasks, such as importance modeling across time, are more closely related to the problem of link prediction, which will be discussed in Section 2.6.

2.5.1 Framework

The activities of entities captured by a social network can be viewed as a stream (e.g., a stream of edges). Streaming represents a natural model of computation that can be used to describe evolving social networks across time and will also be adopted in this section.

Modeling social networks across the time axis

In this section, we study a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ across the time axis, and we assume a series of discrete time points $t_1, \ldots, t_{n-1}, t_n, \ldots$. At each time point t_i we observe a graph instance $\mathcal{G}(\mathcal{V}_i, \mathcal{E}_i)$ which we will also denote as $\mathcal{G}_i$. The most typical change that can occur in the graph between two time points t_{i-1} and t_i is the addition of edges or nodes, i.e., $\mathcal{E}_{i-1} \subset \mathcal{E}_i$ and $\mathcal{V}_{i-1} \subset \mathcal{V}_i$.

One way to model time evolution under such a definition is to assume that each time point t_i corresponds to a moment at which a new node or edge is recorded. This approach corresponds to the streaming model of computation formalized in [50]. A stream is a sequence of records $x_1, \ldots, x_i, \ldots, x_n, \ldots$ arriving in an increasing order, where x_i is either a new node or a new edge. Alternatively, the time axis can be discretized into intervals of equal lengths, such as years, days, and hours, or buckets of records of equal size.

The objective of a streaming algorithm is to maintain an up-to-date model or representation of the incoming data, which means the model must be continuously adapted to the most recently added records. In the simplest case, a model can store all records seen so far, or in other applications, some of the older records might be forgotten over time and removed from the model. For instance, for identifying important nodes or understanding how communities grow or shrink, it would be reasonable to forget some graph elements as they age. The process of forgetting is often modeled by a *sliding window* where only records within a specific window are considered by the model at any given time. The records within the window can also be assigned different weights based on their recency [51].

Modeling the changes in a dynamic social network represented by a stream of ordered events makes it possible to abstract the joint task of community discovery and community evolution monitoring as follows. One model M_i is built at each time point t_i and then adapted into model M_{i+1} using the data from the following time window. Methods leveraging this abstraction will be discussed in the remaining part of this chapter.

2.5.2 Incremental tracing of communities

In incremental clustering and stream clustering, a cluster built at some point in time evolves as new records arrive and old records are forgotten. One could also observe a community as a cluster in community stream mining. However, it is required to incorporate some additional insights about the evolution of each specific community/cluster from one time point to the other. One research branch focuses on tracing the same community at different time points, which we will discuss in this section. The other branch learns communities across time while it also ensures temporal smoothness. The latter will be discussed in Section 2.5.3.

One of the earliest works in this area was done by Aggarwal and Yu [1], which adheres to the stream paradigm we defined here but also incorporates an offline exploratory component. This method makes a distinction between online and offline components. The online component is responsible for summarizing the information in the stream and building micro-clusters. In contrast, the offline

component can be invoked for different time frames, delivering the final clusters built from the microclusters. This method assumes three possible community transitions: expansion, contraction, and no change, based on the interaction level among the members of the communities. For community clustering, the arriving interactions are weighted based on their recency during the summarization step.

To characterize the transitions that are taking place in the network, Aggarwal and Yu point out that some edges in the graph indicate intensified interactions (denoted as positive edges), and other edges are negative, indicating less interaction than before. The algorithm for the offline component starts with a set of centroids, to which further nodes are assigned, which eventually leads to the partitioning of the graph.

A distinction between positive and negative edges in community evolution was also made by the authors of DENGRAPH [52], which is an incremental variation of DBSCAN adjusted for graph applications. For each snapshot of the graph, DENGRAPH computes the proximity between nodes, thereby forgetting old interactions and taking new ones into account. The former indicates negative changes, which eventually lead to community splits or shrinking, while the latter are positive changes that lead to community growth and fusions with other communities.

The approach adopted by Falkowski et al. also consists of a static and a dynamic component [53, 54]. The static component uses the hierarchical divisive clustering of Girvan and Newman introduced in Section 2.4.3 to cluster the graph within the current time window. Subsequently, the dynamic component matches communities detected at different time points. For this task, Falkowski et al. employ the MONIC framework [55], specifically designed to compare clusters across different time points. Within the MONIC framework, two communities are considered the same if the number of their overlapping members exceeds a certain threshold. The output from MONIC is a temporal graph where communities found at different time points are represented by nodes, and two communities (nodes) are connected by an edge if the overlap of their members exceeds the predefined threshold. The clustering algorithm of Girvan and Newman is applied again to this resulting graph which allows the study of the evolution of volatile communities. Measures of stability, density, and cohesion of communities across time points are also proposed in [54].

The authors of [56] also study the evolution of communities using clustering over bipartite graphs. The graph clustering is performed at different time points, and the algorithm (named *TimeFall*) compares the clusters at different time points. The community must be expressed as a set of words, e.g., words describing a user profile. This requirement limits the approach to networks where word vectors can describe nodes. The upside of this method is that it employs the robust Minimum Description Length criterion for community matching. Matched communities are linked into a temporal graph that resembles a *time waterfall* which inspired the name of this algorithm.

2.5.3 Tracing smoothly evolving communities

The methods described in Section 2.5.2 work by matching communities detected at different time points and using this information to infer how they evolve (e.g., shrink or grow). A completely different approach to community evolution incorporates assumptions on how communities evolve into the model. To be precise, let us assume that communities are a smoothly evolving collection of interconnected entities. Then, community detection overtime translates either into finding a sequence of models [57] or into dynamically learning and adapting a probabilistic model [58], such that the model evolves smoothly from one time point to the next.

Temporal smoothness for clusters

To explicitly capture the continuity between the new and prior models, the authors of [57] introduce the notion of *temporal smoothness*. This notion is included in the objective function to be optimized during learning. The model quality at each time point is captured by a *cost function*, which has two components. The first is the *snapshot cost*, which measures the clustering quality at the given time point (or snapshot). The second is the *temporal cost*, which measures the similarity of the clustering learned at two consecutive time points. Under such formulation, community detection becomes an optimization problem of finding a sequence of models that minimize the overall cost. The model cost denoted by ξ_t is learned from the entity similarity matrix denoted by $\boldsymbol{M}_t$ valid at time point t:

$$Cost(\xi_t, \boldsymbol{M}_t) = snapshotCost(\xi_t, \boldsymbol{M}_t) + \beta \times temporalCost(\xi_{t-1}, \xi_t) \quad (2.21)$$

The matrix $\boldsymbol{M}_t$ is computed by considering: (a) the similarity of entities from the current snapshot of the underlying graph and (b) the temporal similarity, which reflects the similarity between entities from earlier moments. The change parameter β weighs the importance of the *temporal smoothness* relative to the quality of the clustering model. β is typically chosen from the range $[0, 1]$.

The authors of [59] extended this method first by adjusting the cost function by introducing an α parameter to control the impact of the snapshot cost explicitly:

$$Cost(\xi_t, \boldsymbol{M}_t) = \alpha \times snapshotCost(\xi_t, \boldsymbol{M}_t) + \beta \times temporalCost(\xi_{t-1}, \xi_t) \quad (2.22)$$

Then, the authors also provided two ways of modeling temporal smoothness: (a) preservation of cluster membership at time point t by measuring the overlap between the clusters at t vs. the previous time point $t-1$, (b) preservation of cluster quality at t by measuring the degradation of the quality of clusters found in $t-1$. Further, they did not only consider K-means clustering as in [57] but also spectral clustering. For K-means, the snapshot cost of model ξ_t is expressed as the sum of square errors $SSE(\xi_t, t)$. Preservation of cluster quality from $t-1$ to t (PCM) is expressed as the SSE of the clusters in ξ_t towards the centers they had at $t-1$ and are normalized by the cluster cardinalities:

$$PCM(\xi_{t-1}, \xi_t) = -\sum_{X \in \xi_{t-1}} \sum_{Y \in \xi_t} \frac{|X \cap Y|}{|X| \times |Y|} \quad (2.23)$$

Dynamic probabilistic models

Probabilistic models are well suited for learning network dynamics under the assumption of smooth evolution. A similar field where these methods have been successfully applied for years is text stream mining (also called topic modeling), e.g., [60, 61, 62]. In the methods discussed so far, a community was defined as a cluster of proximal entities, where the proximity was modeled as a similarity in behavior or properties of nodes. Probabilistic models, on the other hand, assume that the formation of communities results from a generating process that a number of latent variables can describe. If, for instance, the latent variables are called communities, as in [63], then a community determines each activity observed at each time point with some probability.

One of the earlier studies that applied dynamic probabilistic models to community evolution in networks was done by Sarkar and Moore [58]. They study a social network of interacting entities, i.e., a single stream of interactions, and look at the evolution of relationships among entities under two assumptions. First, entities can move in the latent space between time steps, but large moves are improbable. Second, they make the standard Markov assumption that latent locations at time $t+1$ are conditionally independent of all previous locations given latent locations at time t. Thanks to the first assumption, the contribution of a latent variable to the given entity's interaction behavior can change from one time point to the next, but not drastically.

The model of Sarkar and Moore (named Dynamic Social Network in Latent Space model or DSNL) consists of an *observation model* and a *transition model* [58]. The observation model includes a likelihood score function that measures how well the model explains pairs of connected entities in the graph. Entities vary their *sociability*, and a kernel function weights linking probabilities. The core idea of this method is that the sociability of each entity is a radius in the latent space, and each entity will connect with all entities within its radius with a high probability. They also model less likely connections to entities beyond the radius with a constant probability p corresponding to noise. The transition model penalizes drastic changes of the current model towards the previous one.

There is a correspondence between the snapshot and temporal costs proposed for clusters introduced in Section 2.5.3. The method of Sarkar and Moore targets the quality of the probability model at each time point (snapshot quality) and minimal perturbation between time points (temporal smoothness). They also make sure to avoid local minima, and as a result, this method performs very well for both synthetic and real-world networks.

Dynamic topic modeling for community monitoring is used in FaceNet [63], which builds upon the evolutionary clustering introduced in [57]. The cost of deviating from temporal smoothness is modeled using Kubler-Leibler divergence. A community is a latent variable, and each node is described by all communities (with different probabilities). Community evolution is captured by a so-called *Evolution Net* where the nodes correspond to communities at distinct time points. In Evolution Net, an edge between community c at time t and a community c' at time $t' > t$ is the probability of reaching c' from c.

2.6 Link prediction

Link prediction is an essential task in social network analysis and has applications in numerous areas, including information retrieval, e-commerce, and bioinformatics. There are various approaches to link prediction in the literature, ranging from feature-based classification [7, 64] and kernel methods to matrix factorization [65] and probabilistic graphical models [66, 67]. These methods differ from each other in terms of complexity, prediction performance, and scalability. In this section, however, we will only cover the feature-based classification methods as they are most relevant to this study.

2.6.1 Background

Formally, the link prediction problem can be formulated as follows. Given a social network defined as graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, in which an edge $e = (u, v) \in \mathcal{E}$ represents some form of interaction between two nodes at a given time $t(e)$. Multiple interactions between two nodes can be represented by parallel edges or by using complex time stamps for edges (e.g., a list of time stamps). For time $t \leq t'$, we assume that $\mathcal{G}[t, t']$ denotes the sub-graph of $\mathcal{G}$ restricted to the edges with time stamps between t and t'. In the link prediction task, we choose a *training interval* $[t_0, t'_0]$ and a *test interval* $[t_1, t'_1]$ where $t'_0 < t_1$. Now, the link prediction task is to output a list of edges not present in $\mathcal{G}[t_0, t'_0]$, but are predicted to appear in the network $\mathcal{G}[t_1, t'_1]$. This formulation is based on one of the earliest definitions of the link prediction problem proposed by Kleinberg et al. [7].

Link prediction has a wide variety of applications in many different domains. In the context of the internet, it can be used for automatic web hyperlink creation [68] and website hyperlink prediction [69]. In e-commerce, one of the most common usages of link predictions is to build recommendation systems [70]. In bioinformatics, it is used to predict protein-protein interactions (PPI) [71]. In security-related applications, it is used to identify hidden groups of terrorists and criminals. Thanks to its generality, link prediction can be applied to graphs that represent online social networks (e.g., Facebook) and other types of networks such as information networks, biological networks, and many others.

2.6.2 Feature-based methods

The link prediction problem can be modeled as a *supervised classification* task, where each data point corresponds to a pair of vertices in the social network. The model is first trained on the link information from the training interval $[t_0, t'_0]$ and then used to make predictions of future links between nodes in the test interval $[t_1, t'_1]$. More formally, let us assume that $u, v \in \mathcal{V}$ are two vertices in the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and the label of the data point $\langle u, v \rangle$ is $y^{\langle u,v \rangle}$. In this case, we will assume that the interactions between u and v are symmetric, so the pair $\langle u, v \rangle$ and $\langle v, u \rangle$ represent the same data point. Now, the label y is defined as:

$$y^{\langle u,v \rangle} = \begin{cases} +1 & \text{if } \langle u, v \rangle \in \mathcal{E} \\ -1 & \text{if } \langle u, v \rangle \notin \mathcal{E} \end{cases} \tag{2.24}$$

By using the above labeling on a set of training data points, it is possible to build a classification model to predict unknown labels for pairs of vertices $\langle u, v \rangle \notin \mathcal{E}$ in the graph $\mathcal{G}[t_1, t_1']$. Effectively, we have reduced link prediction into a typical classification task that supervised classification algorithms, such as naive Bayes, support vector machines, or neural networks, can solve. The key to successful classification-based link prediction is the feature set construction which we will discuss in the following subsection.

Feature set construction

Selecting an appropriate feature set is one of the most critical parts of any machine learning algorithm. In link prediction, each data point represents a pair of vertices with a label denoting their link status. Intuitively, the selected features for link prediction should represent some form of proximity between pairs of nodes. In existing research, most features are extracted from the graph topology (or structure). However, one of the significant advantages of feature-based link prediction is the ability to incorporate features based on any additional node-relevant data. The use of domain-specific node data for link prediction is valuable in areas such as online social networks where both the users and the resources they upload represent rich data that can be converted to useful features for prediction, e.g., users' topic interests.

The work of [7, 64] on link prediction uses feature sets consisting of only topological features. They compute a similarity score based on the node neighborhood or path-based statistics between pairs of nodes. The most common similarity scores that are also used for link prediction have been discussed in Section 2.3, and they include: the *Adamic/Adar score*, the *Katz score*, and the *Jaccard coefficient*. All of these similarity measures can be directly used as features in the classification task of predicting future links. The main advantage of these features is that they are generic and applicable to graphs from any domain. However, for large social networks, constructing some of these features may become computationally expensive.

Numerous studies, including [72, 73], showed that features derived from vertex attributes can significantly increase the performance of link prediction tasks. For instance, Hasan et al. [73] showed that attributes such as the degree of overlap among the research keywords used by two authors were one of the most predictive features in their dataset for link prediction in a co-authorship social network. In this case, the vertex attribute was the research keyword set, and the assumption was that a pair of authors were close (in the context of a social network) to each other if their research work used a larger set of common keywords. The advantage of such feature sets is that they are generally cheap to compute. On the other hand, the main disadvantage is that these features are tightly tied to the domain, which the user of this technique must be familiar with.

Classification models

Many classification models are used for supervised learning, including naive Bayes, decision trees, random forests, SVMs, neural networks, and others. There

are also regression models, such as the logistic regression, which can be used for link prediction as well [74]. Even though the performance of these methods is often comparable, it was also shown that some methods might have advantages in specific datasets or domains. For instance, in [73], the authors showed that bagging and SVMs have a marginal competitive advantage over other methods in a co-authorship social network. Learning a classification model in the context of link prediction has some specific challenges that can make certain models more attractive.

The first challenge in supervised link prediction is the underlying data's extreme *class skeweness*. The number of possible edges in a social network is quadratic in the number of its vertices. The number of links to be added to the graph is often only a small fraction of this number, which naturally leads to a very large class skewness, making training and inference difficult. For example, Hasan et al. [73] reported a good performance on the link prediction task on the DBLP dataset. However, they ignored the class distribution and reported cross-validation performance from a dataset where the population was balanced. If the original class distribution were used, the performance would drop significantly.

To demonstrate the extent of this issue, let us look at the DBLP dataset more closely. In 2000, the ratio of the actual and possible edges was as low as 2×10^{-5}. So, in a uniformly sampled dataset with one million training instances, we can expect only approximately 20 positive examples. Even worse, it was observed that the ratio between the number of positive edges and the number of possible edges decreases over time because the number of negative links grows quadratically. In contrast, positive links only grow linearly with each new node. It was reported in [75] that the number of authors in DBLP increased from 22 thousand to 286 thousand between 1995 and 2005. Thus the possible collaborations increased by a factor of 169, while the actual collaborations only increased by a factor of 21.

There are several different approaches to cope with class skewness. Some methods work by altering the training samples by either up-sampling or down-sampling [76]. Then, other methods alter the learning method itself by turning it into an active [77] or const-sensitive [78] process and by treating the classifier scores with different thresholds [79]. For kernel-based classification methods such as SVMs, there are specific approaches that can reduce the imbalance problem [80].

The second big challenge of supervised link prediction is model calibration which is often even more important than the model selection in this context. Model calibration is the process of finding a function that transforms the output score value of the model into a label. This calibration function controls the ratio of the false positive error and the false negative error. In many applications of link prediction, such as, for instance, detecting links in a terrorist network, the cost of missing a true link can be catastrophic. On the other hand, in online social networks recommending a wrong link can be considered more severe than missing a true link. Hence, it is essential to incorporate these assumptions into the model by a calibration process. Popular calibration techniques used in machine learning include *Platt scaling* [81] and *Isotonic regression* [82].

The last common problem in supervised link prediction is the training cost in terms of time complexity. Most real-world social networks are very large, and

due to the class imbalances, a model's training dataset needs to consist of many samples so that rare cases are represented [83]. In such a scenario, classification cost may also be considered when choosing the model. For instance, running an SVM with millions of training samples could be costly regarding resource needs, while a naive Bayes classification would be comparably cheaper.

2.7 Pattern mining in graphs

Pattern mining is a fundamental area of data mining that focuses on applying algorithms to detect meaningful patterns within existing datasets. A pattern is considered meaningful if it provides novel insights that contribute to understanding historical trends in the data or predicting future occurrences. This approach has been employed across diverse data types, including transactional data [84], time series [85], spatial data [86], and graphs [87]. The challenges within pattern mining stem from the need to consider a vast number of potential patterns before identifying the desired ones. As a result, algorithms used for pattern mining rely on efficient data structures and implementing strategies for pruning the search space of all possible patterns.

Graphs, as a foundational structure in various data domains, have attracted significant attention in the context of pattern mining. Graph pattern mining pertains to analyzing patterns embedded within graph-structured data, encompassing different graph types such as weighted, directed, attributed, and *dynamic graphs*. These methods are especially compelling in the context of dynamic networks since they can generate rules that describe the evolution of a network on a microscopic level, i.e., individual nodes or edges. Furthermore, these rules are easily understandable by humans and can be directly used to extract knowledge from networks.

Graph mining is a very active field of research. In this section, we will cover key concepts and approaches that underpin graph mining algorithms. By first exploring fundamental techniques for pattern mining in static graphs, we will establish a solid foundation to subsequently delve into algorithms designed for discovering patterns in dynamic graphs.

2.7.1 Static graphs

Most algorithms designed for static graphs work with labeled graphs, as labels represent additional data attached to the nodes/edges that the graph mining algorithms can leverage. Therefore, we first introduce static labeled graphs in Definition 2.7.1 below.

Definition 2.7.1 (Static labelled graph)**.** *A static labeled graph is a tuple* $G = (V, E, L_V, L_E, \phi_V, \phi_E)$*, where* V *is a set of vertices,* $E \subseteq V \times V$ *is a set of edges,* L_V *is a set of vertex labels,* L_E *is a set of edge labels,* $\phi_V : V \to L_V$ *is a function mapping vertices to labels, and* $\phi_E : E \to L_E$ *is a function mapping edges to labels. The underlying graph can be directed or undirected, and this definition can also be generalized to multigraphs.*

One of the most popular graph mining tasks is *frequent subgraph mining* which aims to find all subgraphs that appear frequently in a database of connected graphs. Given a parameter called minimum support threshold or *minsup*, a graph is frequent if it appears at least *minsup* times in the database. In the context of this study, subgraph mining could help find associations between creditors that appear together in different insolvencies. To formally define frequent subgraph mining, we will first define the concept of a subgraph using *graph isomorphism* in Definition 2.7.2 and then *support* of a subgraph in Definition 2.7.3.

Definition 2.7.2 (Graph isomorphism)**.** *Let $GD = \{G_1, G_2, G_3, \ldots, G_n\}$ denote a graph database which consists of n labeled static graphs. Consider two labeled graphs, $G_x = (V_x, E_x, L_{Vx}, L_{Ex}, \phi_{Vx}, \phi_{Ex})$ and $G_y = (V_y, E_y, L_{Vy}, L_{Ey}, \phi_{Vy}, \phi_{Ey})$. The graph G_x is isomorphic to graph G_y, if and only if there exists a bijective function $f : V_x \to V_y$ such that the following requirements hold:*

1. $\forall v \in V_x, L_{Vx}(v) = L_{Vy}(f(v))$
2. $\forall \{u, v\} \in E_x, \{f(u), f(v)\} \in E_y$
3. $L_{E_x}(u, v) = L_{Ey}(f(u), f(v))$

A graph G_x is a subgraph isomorphism (appears in) a graph G_z, denoted as $G_x \sqsubseteq G_z$, if there exists a subgraph $G_y \subseteq G_z$ such that G_x is isomorphic to G_y.

Definition 2.7.3 (Support)**.** *The support of a graph G_x in a graph database GD is defined as $sup(G_x) = |\{g | g \in GD \wedge G_x \sqsubseteq g\}|$*

A subgraph isomorphism is also called an *embedding*. Given a graph database GD and a threshold *minsup*, the task of frequent subgraph mining is to enumerate all frequent subgraphs (or embeddings) that have support no less than *minsup*. Frequent subgraph mining is difficult because many subgraphs must be considered, and their support must be calculated before identifying the frequent subgraphs. Several efficient algorithms were proposed for frequent subgraph mining on static graphs in the literature, and the most notable ones will be covered in the rest of this section.

Given a set of undirected graphs and a *minsup* threshold, the *gSpan* [87] algorithm identifies all frequent subgraphs in the given set. gSpan maps subgraphs to unique minimum depth-first search (DFS) codes and uses this code to sort the subgraphs in lexicographic order. This lexicographic order allows gSpan to use a DFS (depth-first search) strategy to mine subgraphs efficiently. More specifically, gSpan traverses a DFS Code Tree, where the code of a node corresponds to the parent's code extended by one edge, and the siblings are ordered according to the lexicographic order. The algorithm starts from the smallest subgraphs and backtracks if the corresponding subgraph is not frequent.

Unlike gSpan, the *Subdue* [88] algorithm searches for subgraphs that can best compress the input graph or the set of input graphs. The compressibility is evaluated by the Minimum Description Length (MDL) principle. The best substructure is the one that minimizes $DL(S) + DL(G|S)$, where $DL(S)$ is the description

length of the substructure and $DL(G|S)$ denotes the description length of the input graph G after the compression of S. Once the best substructure is found, the input graph is compressed by replacing the occurrences of the substructure with pointers. Then, the whole process repeats. This method results in a hierarchical description of the input graph regarding the discovered substructures.

Another algorithm based on depth-first search is *Sleuth* [89], designed for mining frequent subtrees from a set of rooted trees. An improved version of Sleuth can also mine *ordered* and *unordered* trees and *induced* and *embedded* trees. In *ordered* trees, the order of sibling nodes matters (unlike in *unordered* trees). On the other hand, *induced* trees are tree embeddings that preserve the parent-child relationship, i.e., if two nodes are in a parent-child relationship in the induced tree, then the same relationship also exists in the input trees. *Embedded* trees only require the ancestor-descendant relationship to be preserved. Specifically, if two nodes are in a parent-child relationship in the embedded tree, then the path connecting these two nodes must exist in the input graphs, but it can also contain other nodes.

2.7.2 Dynamic graphs

In this section, we will focus on pattern mining in dynamic graphs. First, we introduce a notion of a *dynamic graph* in the context of pattern mining. Then, we focus on mining algorithms for subgraph mining and rule mining.

Definition 2.7.4 (Dynamic labeled graph)**.** *A dynamic graph is a sequence $DG = (G_1, G_2, \ldots, G_n)$, where G_i is a static labeled graph extended by timestmap functions $t_{G_I,V}$, $t_{G_i,E}$ for $i = 1, \ldots, n$. The timestamp functions are defined as $T_{G,V} : V_G \to T$ and $T_{G,E} : E_G \to T$ and map vertices and edges to a point in time, respectively. We assume a discretized time represented by a set of integers, i.e., $T = \mathbb{Z}$. The graph G_i is called the snapshot of DG at time i.*

An example of a dynamic graph in the context of this study is the insolvency graph capturing debtor-creditor relationships where timestamps on the edges represent the creation of the creditor's claim against the debtor. Using the timestamp functions, we can add more information about the processes in the graphs and look for relative differences between timestamps. For instance, we can discover changes in a graph that happen at similar times. Vertex and edge timestamps can represent the creation of the vertices and edges or the change in their labels.

It is essential to point out that there are also other definitions of dynamic graphs in the literature, including other definitions of patterns in dynamic graphs. In the following two (sub)sections, we will focus specifically on methods for mining *dynamic subgraph patterns* and *evolution rules*. Other methods designed for mining sequence patterns, discriminative patterns, or anomaly patterns are beyond the scope of this work.

Dynamic subgraph mining

The algorithms described in this section are similar to those used for static graphs. The mining process typically also involves solving a subgraph isomorphism problem with the additional requirement to consider constraints on the node/edge timestamps.

The first algorithm, *Dynamic GREW*, proposed in [90], assumes the input graph has a fixed set of nodes, and edges are inserted and deleted over time. The presence of edges is expressed by a sequence of 0s and 1s called *existence strings*, where 1 represents presence of an edge, and 0 represents absence. A dynamic subgraph of length k of a dynamic graph is a subgraph from both the topological and dynamic views. More specifically, the existence strings of the dynamic subgraph have length k, and they are substrings of existence strings of the original graph starting from the same position. A frequent dynamic subgraph is one with at least t occurrences for a given value of t.

The dynamic subgraph enumeration algorithm proposed in [91] was designed to solve the Dynamic *Subgraph Enumeration Problem* (DSE), which can be stated as follows. Given a subgraph H and a sequence $\mathcal{G}$ of graphs $G_1, G_2, \ldots$, where G_{t+1} is obtained by modifying a single edge in G_t. The goal of DSE is to maintain a dynamic data structure for each G_t so that the number of subgraphs of G_{t+1} isomorphic to H can be estimated efficiently without recomputing them in G_{t+1} from scratch.

Subgraph mining from *interaction temporal graphs* was considered in [92]. An interaction temporal graph was defined as a static directed graph, where edges represented interactions between entities (nodes) and edges were marked with the start time and duration of the interaction. The algorithm proposed in [92] mines so-called *time-respecting subgraphs*, i.e., connected subgraphs in which the interaction of an edge starts soon after the interactions of the adjacent edges.

Rule mining in dynamic graphs

In this part, we will cover methods for rule mining in dynamic graphs. Let B and H be two graphs, then $B \rightarrow H$ is a graph rule in which B is the *body* (precondition), and H is the *head* (postcondition) of the rule. Graphs B and H are static graphs extended by timestamp functions on their edges or nodes. Their embeddings are typically subgrahs of snapshots G_i and G_j of a dynamic graph $DG = (G_1, G_2, \ldots, G_n)$, where $1 \leq i, j \leq n$. Generally, the body and the head can be arbitrary graphs, and their interpretation may differ for different methods. For instance, if both the body and the head come from the same snapshot, then the rule is denoted as an *association rule*, and such a rule can be used to examine the co-occurrence of subgraphs. However, if the body precedes the head's snapshot, the rule is denoted as *predictive* or an *evolution rule*. These evolution rules can help understand the evolution processes in dynamic graphs.

Similarly to frequent subgraph mining, a *support* measure is also defined for these rules. The literature has several different support definitions, and each is typically adapted to a specific scenario. For instance, rules in [93] only assume

edge additions. Thus, the body is always a subgraph of the head, and support is calculated as the number of occurrences of the head. In order to allow the algorithms to mine these rules efficiently, it is necessary for the support definitions to be anti-monotonic, i.e., the support of a pattern is at least as large as the support of its super patterns. Using the support measures, defining a *confidence* measure for rules is also possible. Confidence is typically expressed as the support ratio of the rule's head to its body, and different definitions of support lead to different definitions of confidence.

One of the earliest approaches for mining rules in dynamic graphs was proposed in [93], where the authors introduced the so-called *Graph Evolution Rules* (GER). A GER is a rule in which the same subgraph is used for the body and the head, but the body contains all edges except those with maximum timestamps. The method for extracting these rules is called the *Graph Evolution Rule Miner* (GERM), and it was designed for undirected graphs in which nodes and edges are only added and never deleted. This approach, however, can also be extended to cases with edge and node deletions. Furthermore, this method assumes that the node and edge labels do not change over time and timestamps are assigned only to edges but not nodes.

Similarly, LFR-Miner proposed in [94] was also designed to mine rules for predicting new edges between pairs of vertices in a directed graph. The body of a rule is made up of a subgraph, but there is a designated pair of vertices called the *start node* and *end node*. The head of a rule only contains a directed edge from the start node to the end node. Timestamps of all edges in the body have to be smaller than the timestamp of the edge in the head. Additionally, all the other nodes in the body have to be connected to both the start node and the end node. This algorithm also assumes that edges are only added and not removed.

3. Insolvencies in the Czech Republic

On January 1, 2008, the Parliament of the Czech Republic adopted a new Act No. 182/2006 Coll. on Insolvency and Methods of its Resolution, also known as the Insolvency Act. The purpose of the new law was to modernize the existing insolvency process and to increase the transparency and effectiveness of the insolvency proceedings in the Czech Republic. The new Insolvency Act was also complemented by the launch of a new publicly accessible information system called the Insolvency Register of the Czech Republic (IR), which the Ministry of Justice of the Czech Republic operates. The new Act mandates that every insolvency proceeding starting January 1, 2008, is published in the register.

As of 2022, IR contains detailed information regarding approx. 375,000 insolvencies. It contains demographic (e.g., age, gender, address) and socioeconomic data (e.g., income, assets, debt information) of more than 270,000 debtors[1] in the Czech Republic, including both companies and individuals.

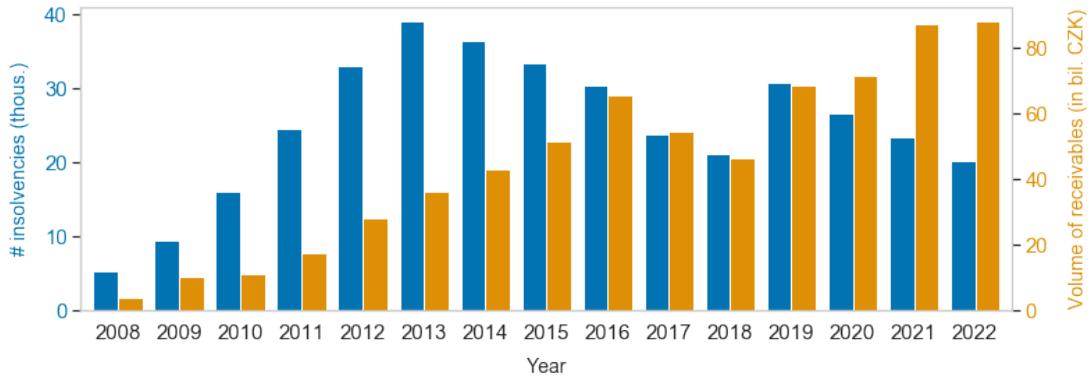


Figure 3.1: Number of insolvencies commenced each year since the launch of the IR in 2008 in blue and the corresponding volume of receivables claimed[2] by the creditors in the same year in orange.

There is significant interest in analyzing this large and growing database of insolvencies in the Czech Republic from multiple perspectives. From a purely economic perspective, this data can help understand how company bankruptcies occur and the network effects of a single bankruptcy on the whole company ecosystem. Furthermore, the outcomes of this analysis can help financial institutions to build more robust risk models. From the socioeconomic perspective, this data can shed light on how individuals get into debt, what role the demography plays in this process, and what systemic changes could prevent people from getting into excessive debt. Lastly, this analysis can provide great feedback to policymakers and lawyers to understand the effects of individual amendments to the existing law and how upcoming amendments could be designed.

[1] A single debtor can be the subject of multiple insolvencies, but not at the same time.

[2] In the insolvency proceeding the receivables are first claimed by the creditor by submitting the application of receivables and then the insolvency court either admits or denies the claim.

Modern data mining methods represent an appealing approach to analyzing these vast amounts of publicly available data. It would be especially interesting to understand the ongoing proceedings in the context of mutual relationships of individual entities participating in the insolvency proceedings (e.g., debtors, creditors, and insolvency administrators). Social network analysis methods provide a natural framework to study the indebtedness structure present in the Czech society today and how it evolves.

The historical evolution of the IR in terms of the yearly number of commenced insolvencies and claimed receivables is depicted in Figure 3.1 and Figure 3.2 respectively. In the first six years after adopting the Insolvency Act in 2008, the number of commenced insolvencies increased yearly until the peak in 2013, when the trend reversed. Another trend reversal can be seen in 2019, caused by the new amendment (see Section 3.1.6) that came into effect in June 2019. This amendment significantly relaxed the requirements for insolvencies of individuals, thus making it available to a broader set of debtors.

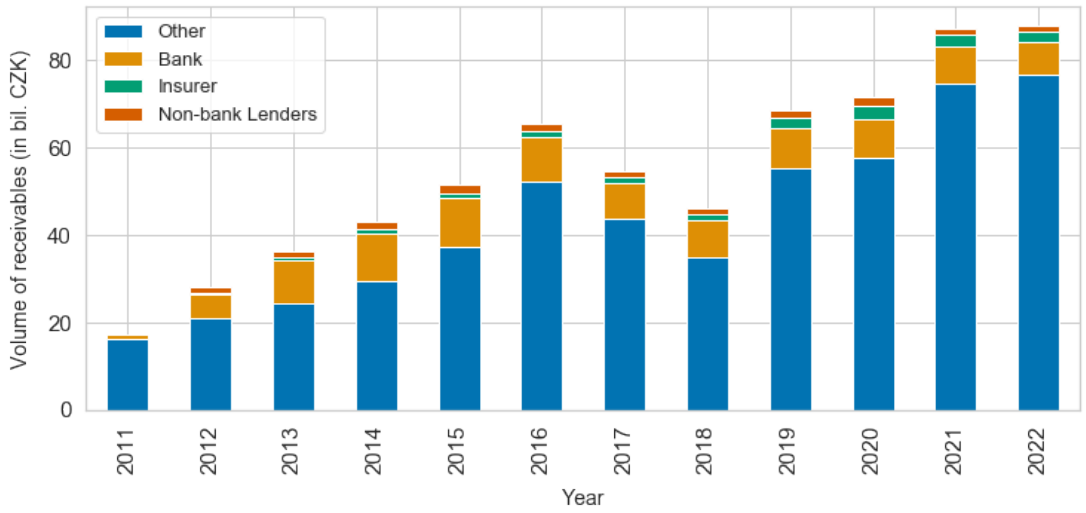


Figure 3.2: Yearly volume of receivables per creditor category.

The rest of this chapter is organized as follows. First, we describe the Insolvency Act and define all the terms used in this study. Next, we will describe the individual entities involved in the insolvency proceedings and the debt resolution methods defined by the Insolvency Act. Lastly, we will describe the Insolvency Register and the states of the insolvency proceedings and provide real-life examples of insolvencies to demonstrate the insolvency process.

3.1 Insolvency Act

This section will describe the Insolvency Act to the extent required for fully comprehending this thesis. We will also provide the definitions of all essential terms. Our description is based primarily on the English translation of the Insolvency Act by Wolters Kluwer ČR, a.s.[95], and the terminology established in this publication will be adopted for this study.

Since the Wolters Kluwer translation was published in 2011, it only contains the original Insolvency Act from 2008 and all its amendments added until August 2011. Nevertheless, it still represents a valid source of more detailed information. The most significant amendments in the context of this study will be reviewed at the end of this section. The original up to date version of the Insolvency Act is available in the Collection of Laws of the Czech Republic[3].

The Insolvency Act denotes the Act No. 182/2006 Coll. on Insolvency and Methods of its Resolution including all its later amendments[4] and was created to replace the former Act No. 328/1991 Coll.

For the purposes of this study:

- An *insolvency proceeding* (IP) is a court proceeding, the subject of which is the debtor's insolvency or imminent bankruptcy and the method of its resolution. A detail page in the IR corresponding to the insolvency proceeding of a company called *Pilsen Steel s.r.o.* is shown in Figure 3.3).
- An *insolvency court* is the court before which the insolvency proceeding is held. The insolvency courts are a part of the regional courts (listed in Table 3.1) in the Czech Republic. The corresponding regional court can be identified by the first four letters in the reference number. For instance, *KSPL* in Figure 3.3 refers to the Regional Court of Plzeň.
- An insolvency proceeding is commenced by filing an *insolvency petition* to the insolvency court. With equal rights, the petition can be filed by the creditor(s) or the debtor himself to one of the regional courts. The insolvency proceeding must start within three days of submitting the petition.
- An *application of receivable* is a procedural act by which a creditor applies the satisfaction of its rights (typically unpaid claims) in the insolvency proceeding. In simple terms, this means that the creditor has to submit an application of receivable to claim his unpaid debt by the debtor. The typical due date for the submission is 30 days from the commencement of the insolvency proceeding.

3.1.1 Insolvency

Debtors are insolvent from the perspective of the Insolvency Act if they have[5]:

1. several creditors and
2. outstanding financial liabilities for more than 30 days overdue, and
3. they are not able to fulfill such liabilities.

[3]The Collection of Laws of the Czech Republic (only in Czech) is available at: `https://aplikace.mvcr.cz/sbirka-zakonu/`. Accessed at 21 November 2023

[4]The current list of all amendments (only in Czech) is available at `http://insolvencni-zakon.justice.cz`. Accessed at 21 November 2023

[5]Defined in Part 1, Chapter1, Section 3 of the Insolvency Act.

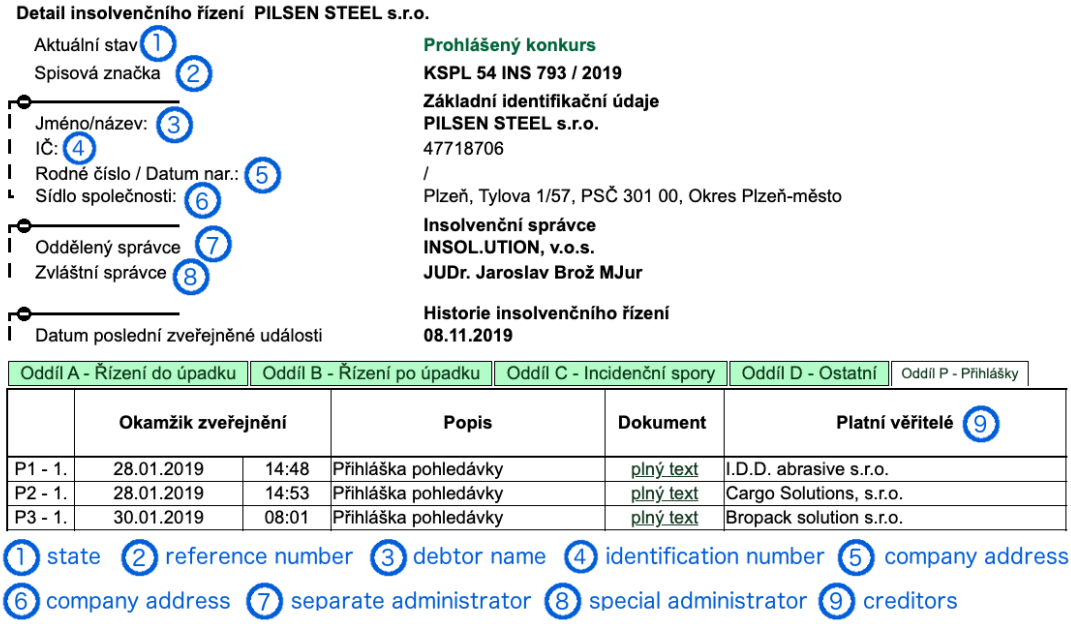


Figure 3.3: Insolvency Register detail page related to the insolvency proceeding of the company **Pilsen Steel s.r.o** with the English translation of the fields in blue.

It is believed that the debtors are not able to fulfill their financial liabilities if:

1. they stopped the payments for the substantial part of their financial liabilities or
2. they have defaulted for more than three months overdue or
3. the enforcement of an execution might not satisfy any outstanding financial receivables against the debtor.

3.1.2 Participants in the insolvency proceedings

The subjects of every insolvency proceeding are the debtor and the creditors who exercise their rights against the debtor.

The creditor is a party (legal entity or an individual) that has delivered a product, service, or a loan to the debtor and is owed money based on unpaid claims. Conversely, the debtor is an entity (legal or an individual) that owes money to the creditor. Creditors claim their receivables by submitting an application of receivables and are satisfied based on the method of resolution chosen by the insolvency court. A specific debtor can only be the subject of at most one insolvency proceeding at a time. However, a debtor can become insolvent on multiple occasions throughout his lifetime.

The role of the insolvency court in the insolvency proceeding is to issue decisions per the Insolvency Act and to continuously supervise the process and activities of all the other procedural bodies (administrators, creditors, and debtors).

The entity that stands between the debtor and the creditors is the insolvency administrator, whose purpose is the handling of assets of the debtor during the insolvency proceeding to achieve the highest possible satisfaction of the creditors. The insolvency administrator is appointed by the insolvency court from the official list[6] of administrators managed by the Ministry of Justice. The appointment of administrators is governed by a special Act No. 312/2006 Coll on Insolvency administrators.

3.1.3 Exceptions from the effects of the Insolvency Act

The Insolvency Act is not applicable if it is in regards to a debtor who is one of the following:

1. the State
2. the local government unit
3. the Czech National Bank
4. the General Health Insurance Company of the Czech Republic
5. the Deposit Insurance Fund
6. the Guarantee Fund of the Securities Traders
7. a public non-profit institutional health facility
8. a public college.

3.1.4 Methods of insolvency resolution

The methods of resolution, as defined in the Insolvency Act, define different ways to satisfy the receivables claimed by the creditors. The three primary methods of insolvency resolution are: (1) bankruptcy order, (2) restructuring, and (3) discharge.

Bankruptcy order is a method of insolvency resolution based on the fact that the determined receivables of the creditors will be essentially satisfied from the proceeds of the liquidation of assets. However, the non-satisfied receivables or any part thereof do not cease to exist unless the law stipulates otherwise. The bankruptcy order applies to both legal entities and individuals.

Restructuring corresponds to the gradual satisfaction of creditors' receivables while preserving the operation of the debtor's company, secured by measures taken for the company's economic recovery under the restructuring plan approved by the insolvency court. The restructuring only applies to companies if their total turnover for the last accounting period preceding the insolvency proceeding was at least 100,000,000 CZK or if they have more than 100 employees.

[6]The up-to-date list of all licensed insolvency administrators is available at `https://isir.justice.cz/InsSpravci/public/seznamFiltr.do`.Accessed at 21 November 2023

Discharge is applicable only in cases when the debtor is not an entrepreneur. It may be performed by the liquidation of the debtor's assets or through the execution of a payment calendar. A discharge by liquidating the assets proceeds similarly to liquidating assets in case of a bankruptcy order. The assets in this case include immovable property (e.g., real estate), movable property (e.g., vehicles), and financial assets (e.g., savings). In case of a discharge through the execution of the payment calendar, the debtor must pay monthly installments to the creditors from his income for at most five years. This resolution prefers social purpose over economic, allowing the debtor a "fresh start" and motivating him to repay his debts [96].

3.1.5 Insolvency states

The description of the Insolvency Act described in the previous sections suggests that the insolvency proceeding goes through a series of states that describe its process. This section lists all the possible states of an insolvency proceeding. The complete insolvency process, including all the states and possible transitions, is depicted in Figure 3.4.

Unresolved is the first and initial state of every insolvency proceeding. In this state, the insolvency court has admitted the insolvency proceeding, but the court has yet to make any decisions about the matter.

Incorrect Entry state, as the name suggests, represents an incorrect entry submitted to the IR with no legal consequences. It is usually employed for cases when an error was made by the court officials working with the register.

Unresolved — Advanced occurs when the insolvency petition was submitted to the incorrect court. The most common example is that the insolvency petition was submitted to the incorrect regional court. Again, this state has no legal consequences, and no decisions or changes are made to the insolvency proceeding while in this state. In this case, the court officials simply transfer the insolvency petition to the proper regional court.

Bankruptcy denotes a state when the insolvency court has decided that the debtor is insolvent based on the criteria defined in Section 3.1.1. While in this state, the insolvency court determines the resolution method most suited for the given insolvency proceeding. It may be one of the three methods defined in Section 3.1.4: discharge, bankruptcy order, and restructuring. The insolvency court can often determine the insolvency and the resolution method in a single decision. In such a case, this state is skipped, and the insolvency proceeding moves directly to the state corresponding to the resolution method.

Bankruptcy Order, Restructuring, and *Discharge* states correspond to the resolution methods described in Section 3.1.4.

Finished indicates that the insolvency court has decided to end the insolvency proceeding, but the decision has yet to reach legal effect.

Effective follows the state Finished, indicating that the decision about the insolvency proceeding's end has attained legal effect. After the insolvency enters this state, it is no longer possible to appeal against the court's decision.

Checked Off indicates that the insolvency proceeding was removed from the list of active cases of the corresponding court.

Revived is a special state used in the appeal procedure when the court declares the previous court decision invalid, which causes the insolvency proceeding to be "revived" and start over.

Canceled by Supreme Court is another special state that is used for cases when the regional court's decision gets overruled by the Supreme Court.

Moratorium is reserved for cases when the debtor can resolve his debts with its creditors before the insolvency proceeding starts. The debtor may file a petition to declare a moratorium to the insolvency court within seven days of the submission of the insolvency petition. Once the Moratorium period has ended, the insolvency court has to decide whether it was successful, and it can either end the proceeding or decide that the debtor is still insolvent.

Bankruptcy Order After Cancellation is a special state reserved for cases when the insolvency court has ended the ongoing Bankruptcy Order but the creditors or the administrator make an appeal to restore it.

3.1.6 Amendments

Since 2008, more than 30 amendments[7] have been added to the original Insolvency Act. The Wolter Kluwer[95] translation used for reference in this chapter contains amendments up to the end of 2013. Covering all amendments is beyond the scope of this study. However, this section will review the two essential amendments most relevant to this work.

Act No. 64/2017 Coll. on improved transparency and abuse prevention

The first amendment, Act No. 64/2017 Coll., took effect on July 1, 2017. This act aimed to improve the transparency of the insolvency proceedings, emphasize digitization of relevant processes, provide better oversight over the work of administrators, and provide more protection against the so-called "bullying" insolvency proposals. Bullying proposals are submitted to knowingly and purposefully harm the reputation of the target entity, which is usually an entrepreneur, to create an unfair competitive advantage.

This amendment significantly changed how the court judges the bankruptcy of entrepreneurs. It provides debtors legal tools that leverage their accounting books, which they can use to better defend themselves against dubious insolvency proposals. Additionally, it gives the court the right to perform a preliminary examination before the insolvency is published in the Insolvency Register to prevent reputation damage to the target entity.

[7] The current list of all amendments is available at `https://insolvencni-zakon.justice.cz`. Accessed at 21 November 2023

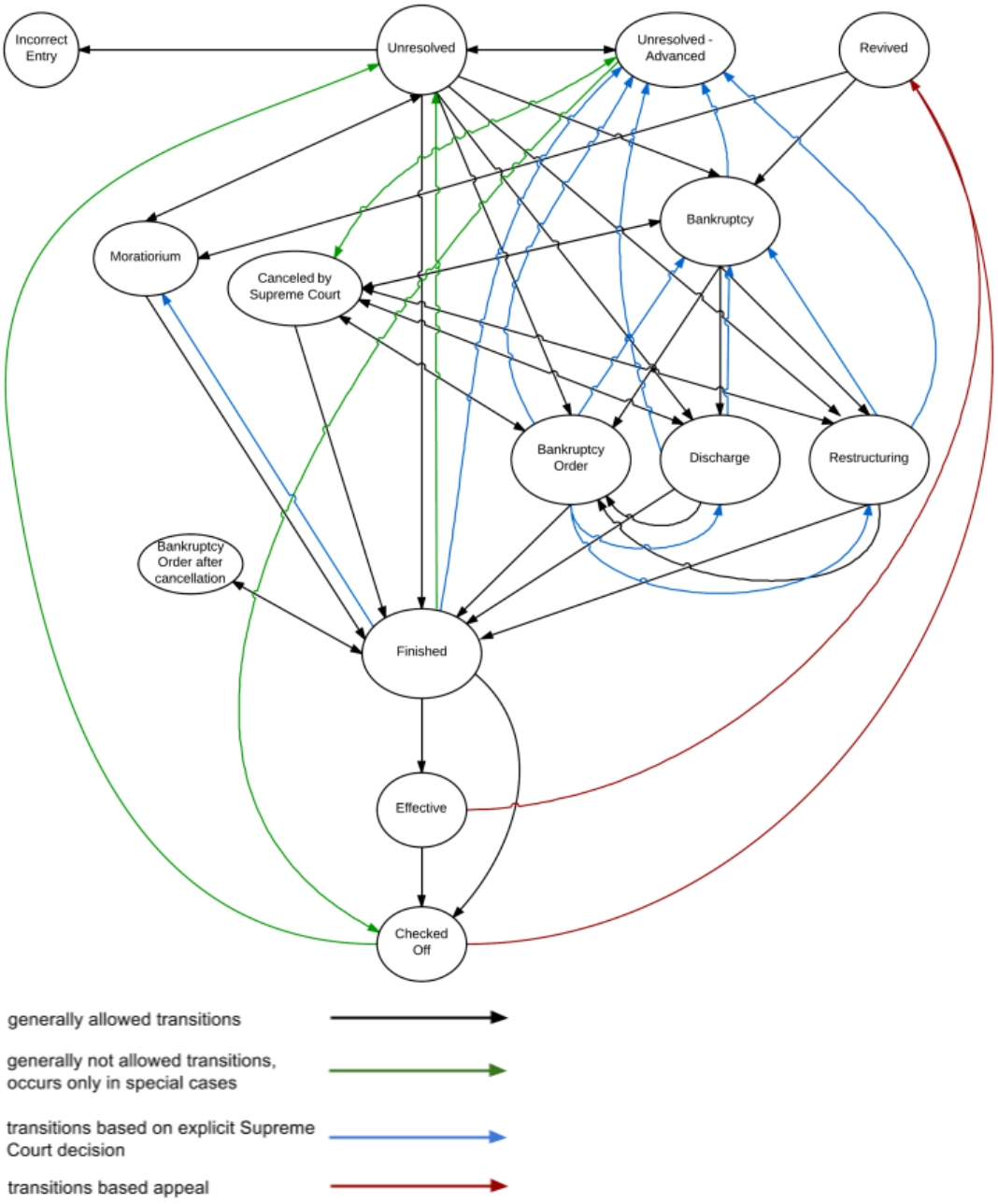


Figure 3.4: Insolvency states as defined by the Insolvency Act.

Act 31/2019 Coll. on Discharge and its application

The second amendment, Act 31/2019 Coll., was enacted on June 1, 2019, and focuses mainly on discharges. Its purpose was to ease the requirements for entering discharge and thus make it accessible to a broader range of potential debtors. Until June 2019, the debtor needed to show that he could repay at least 30% of his debts to the creditors within five years. This requirement has now been lifted, and it is up to the court to decide whether the discharge was successful, regardless of the repaid amount. However, the amendment does not specify any exact evaluation criteria which should be used for such decisions. Understandably, this change gives more decision power to the court and creates much uncertainty for creditors regarding their returns.

The main goal of the amendment was to "unlock" the possibility of discharge to low-income debtors who face multiple executions. These debtors were not admissible for discharge under prior rules. There are between 600,000 and 850,000 such debtors in the Czech Republic[8], representing 5.6% to 8% of the country's population. The effect of the amendment was noticeable immediately after it took effect in June 2019, and the average number of new monthly insolvencies rose by approximately 130% in the second half of 2019 (see Figure 3.5).

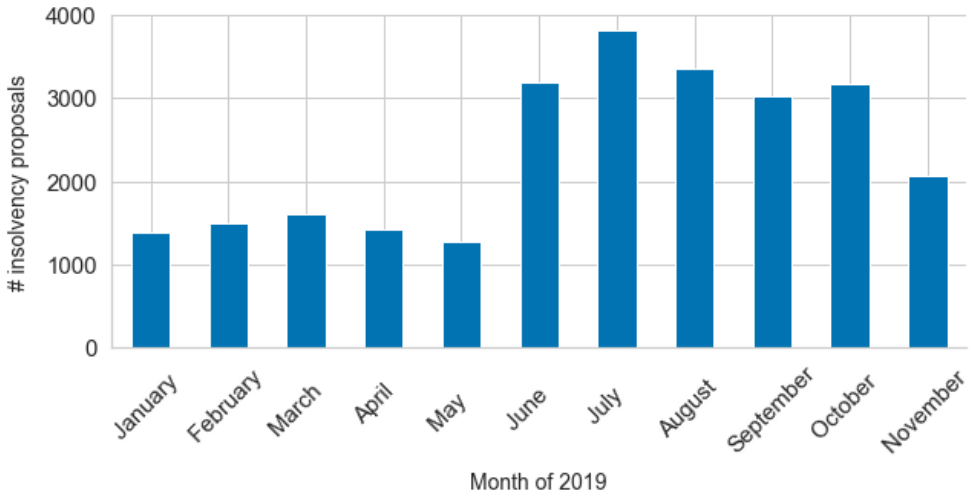


Figure 3.5: Number of newly commenced insolvency proceedings (individuals only) per month in 2019.

3.2 Insolvency Register

The Insolvency Act also introduced a new information system called the Insolvency Register. The IR was created to provide up-to-date information about all ongoing insolvency proceedings. The Insolvency Register is administered by the Ministry of Justice of the Czech Republic, and it is fully accessible to the general public without any restrictions. Based on the Insolvency Act, everyone has the right to inspect the register and make copies and extracts of it.

The Insolvency Register is located at `https://isir.justice.cz` along with its official documentation[9]. Unfortunately, the description of the IR in this chapter is the only existing documentation in English.

3.2.1 Insolvency data

This section lists all the data recorded and published in the Insolvency Register about every commenced insolvency proceeding.

[8]Source: The Chamber of Executors of the Czech Republic (`https://www.ekcr.cz/`)

[9]The official documentation of the Insolvency Register can be found at `https://isir.justice.cz/isir/common/stat.do?kodStranky=NASTENKA`. Accessed at 21 November 2023

Identifier	Court	Region
KSBR	Regional Court of Brno	Jihomoravský
KSCB	Regional Court of České Budějovice	Jihočeský
KSUL	Regional Court of Ústí nad Labem	Ústecký
KSLB	Branch Court Liberec	Ústecký
KSOL	Regional Court of Olomouc	Olomoucký
KSOS	Regional Court of Ostrava	Moravskoslezský
KSHK	Regional Court of Hradec Králové	Královéhradecký
KSPA	Branch Court of Pardubice	Královéhradecký
KSPH	Regional Court of Prague	Prague
MSPH	City Court of Prague	Prague
KSPL	Regional Court of Plzeň	Plzeňský

Table 3.1: List of all insolvency courts in the Czech Republic

If the debtor is an individual, his (or her) basic identification information is recorded: name, surname, domicile, and birth certificate number (if they do not possess a birth certificate number, their date of birth is included). If the debtor is also an entrepreneur, his place of business and identification number are recorded. If the debtor is a legal entity, the company name, headquarters, and identification number are recorded, too.

Every insolvency proceeding has a unique reference number in the format such as **KSPH 37 INS 4970/2010**, where the first four letters **KSPH** identify the insolvency court (in this case, the Regional Court of Prague) and the region where the insolvency proceeding is being held. Table 3.1 lists all possible identifiers and courts. Next in the example reference number comes the number **37**, which identifies the Senate that decides in the given insolvency matter. The Senate is a concrete judicial body within the insolvency court structure. The senate number is followed by letters **INS**, which denotes the case matter (insolvency). Last comes the code **4970/2010**, where the number before the slash is the case number of the proceeding, and the number after the slash is the year of the proceeding's commencement.

The IR also contains detailed process-related information about the insolvency proceeding, including the current state and the insolvency administrator assigned to individual cases. Furthermore, the insolvency court is compelled to publish the following information in the IR in chronological order:

1. all the court decisions related to individual insolvency proceedings
2. any other submissions related to the debtor's case which are recorded in the official judicial file kept by the insolvency court

3.2.2 Documents

The IR further contains various documents related to the insolvency proceedings. These include court decisions and submissions by other parties participating in the proceedings (e.g., creditors and administrators). Even though the new Insolvency Act mandates that all this information must be published, it does not regulate its publishing form. Unfortunately, this has led to limited digitization of the insolvency process, and as a result, many of the documents are uploaded in the form of PDF scans. Figure 3.6 shows an example of a scanned document. As the IR matured over the last decade, there have been some improvements in terms of digitization. Some documents (such as the applications of receivables) are now mostly electronically generated. However, many other documents are still being uploaded in the form of scans.

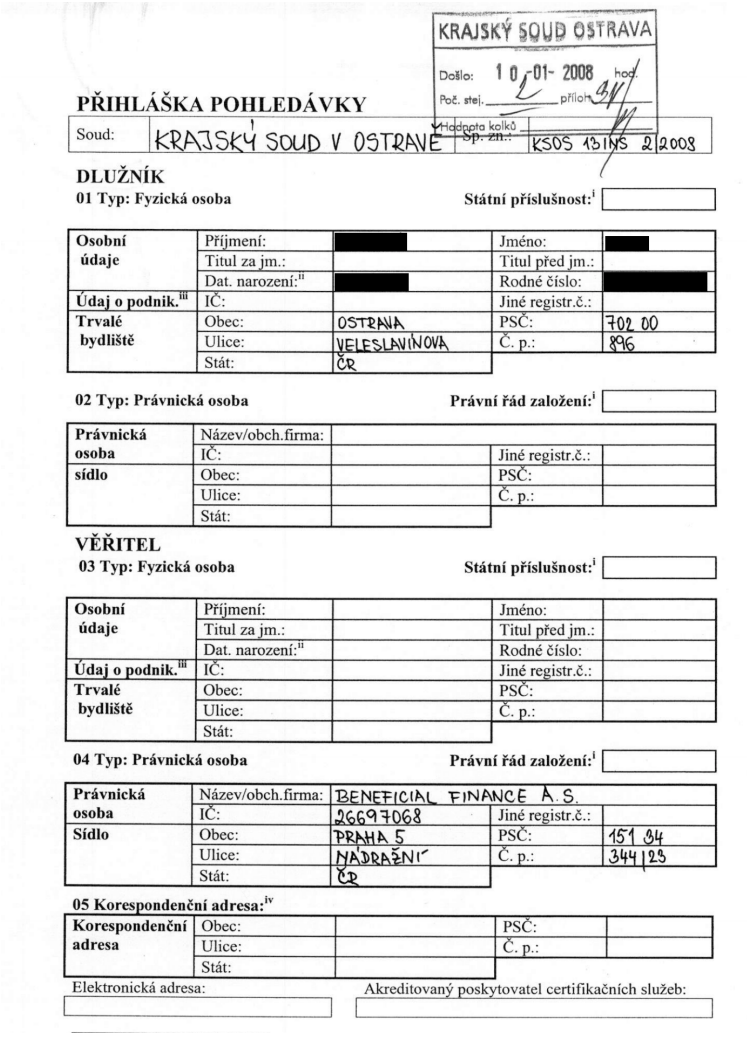
At the end of 2022, approximately 20M of documents related to more than 370,000 insolvency proceedings have been submitted to the IR. Additionally, there are almost 900 different types of documents being used today for tracking the insolvency process and communication between individual stakeholders. The content of these documents is crucial to fully understand the details of the insolvency proceedings, such as how the debtor got into the debt in the first place, what creditors the debtor owes money to, and the size of the debt. Naturally, these details are also crucial for this study since our aim is to analyze and model the dynamics of the insolvency process.

The documents related to a specific insolvency proceeding are divided into five sections: (A) proceeding before declaring bankruptcy, (B) proceeding after declaring bankruptcy, (C) incidental disputes, (D) others, and (P) applications of receivables. As the name suggests, sections A and B, respectively, contain all documents related to the insolvency proceeding before and after declaring bankruptcy. They contain mostly the decisions of the insolvency court, such as the declaration of bankruptcy, the selected method of resolution, or the appointment of the insolvency administrator.

Section C is dedicated to disputes that might occur during the insolvency proceeding. Section D contains documents that do not belong to any other section, such as various requests from creditors and debtors. In Section P, the user can find all the applications of receivables and all their amendments submitted by the creditors.

Applications of receivables

The applications of receivables represent the most essential documents for this work since they determine which creditors take part in the insolvency proceeding and what is the claimed debt. The applications of receivables are also the most frequently submitted documents to the IR. By the end of 2022, more than 3.5 million applications of receivables have been submitted by approximately 130,000 different creditors. The total sum of receivables claimed through these applications adds up to approximately 590 billion CZK ($\sim$ 26.6 billion USD), and the yearly volume of submitted receivables is depicted in Figure 3.1.



KRAJSKÝ SOUD OSTRAVA
Došlo: 10-01- 2008 hod
Poč. stej. 2 příloh 34
Hodnota kolků
Sp. zn.: KSOS 13 INS 2/2008

PŘIHLÁŠKA POHLEDÁVKY

Soud: KRAJSKÝ SOUD V OSTRAVE

DLUŽNÍK

01 Typ: Fyzická osoba — **Státní příslušnost:**[i]

Osobní údaje	Příjmení:	[redacted]	Jméno:	[redacted]
	Titul za jm.:		Titul před jm.:	
	Dat. narození:[ii]	[redacted]	Rodné číslo:	[redacted]
Údaj o podnik.[iii]	IČ:		Jiné registr.č.:	
Trvalé bydliště	Obec:	OSTRAVA	PSČ:	702 00
	Ulice:	VELESLAVÍNOVA	Č. p.:	896
	Stát:	ČR		

02 Typ: Právnická osoba — **Právní řád založení:**[i]

Právnická osoba	Název/obch.firma:			
	IČ:		Jiné registr.č.:	
sídlo	Obec:		PSČ:	
	Ulice:		Č. p.:	
	Stát:			

VĚŘITEL

03 Typ: Fyzická osoba — **Státní příslušnost:**[i]

Osobní údaje	Příjmení:		Jméno:	
	Titul za jm.:		Titul před jm.:	
	Dat. narození:[ii]		Rodné číslo:	
Údaj o podnik.[iii]	IČ:		Jiné registr.č.:	
Trvalé bydliště	Obec:		PSČ:	
	Ulice:		Č. p.:	
	Stát:			

04 Typ: Právnická osoba — **Právní řád založení:**[i]

Právnická osoba	Název/obch.firma:	BENEFICIAL FINANCE A.S.		
	IČ:	26697068	Jiné registr.č.:	
Sídlo	Obec:	PRAHA 5	PSČ:	151 34
	Ulice:	NÁDRAŽNÍ	Č. p.:	344/23
	Stát:	ČR		

05 Korespondenční adresa:[iv]

Korespondenční adresa	Obec:		PSČ:	
	Ulice:		Č. p.:	
	Stát:			

Elektronická adresa: | Akreditovaný poskytovatel certifikačních služeb:

Figure 3.6: One of the first applications of receivables ever published in the IR in January 2008. The sensitive information about the debtor was blacked out for privacy concerns.

Every application of receivables contains the following information:

- unique identification of the creditor (birth certificate number for natural persons or identification number for entrepreneurs)
- list of individual claims (one creditor can have more than 1 claim against the same debtor, e.g., different loans)
- origin of the debt (e.g., loan, unpaid credit cards, or unpaid utility bills)
- outstanding debt for individual claims and the total sum claimed

A significant part of this study focuses (see Section 4.3) on the automatic extraction of this information from the application of receivables.

3.2.3 Exceptions

Upon the request of an individual who made the relevant submission, the insolvency court may decide that some of the personal data in the submission shall not be publicly accessible in the IR. However, the insolvency court always publishes at least the name and surname of such individuals.

The insolvency court can also decide that some submissions are subject to confidentiality under special regulations and can exclude them from publication completely. Nevertheless, all these submissions, together with the data on the nature of the submission, must be indicated in the IR. In most cases, these submissions only represent supporting documents for court decisions, never the decisions themselves.

3.2.4 Data expiration

Five years after the decision by which the insolvency proceeding was completed, the insolvency court deletes the debtor from the Insolvency Register and renders all the information about the case inaccessible. The Insolvency Act also allows the debtor to request the removal of his insolvency proceeding from the Insolvency Register before the five-year expiration period has elapsed. These requests are related to specific decisions made by the insolvency court regarding the particular insolvency petition and include:

1. the rejection of the insolvency petition due to errors in the the submission
2. the termination of the proceeding due to the lack of the conditions of the proceeding, which cannot be eliminated or which could not be eliminated, or due to the withdrawal of the insolvency petition
3. the dismissal of the insolvency petition

3.2.5 Using the Insolvency Register

The IR comprises two main components: the web application and the web service. Even though both of these components were designed for different purposes, they only provide different ways of accessing the same data stored in the IR.

Web Application

The web application was designed mainly for the general public. It allows the users to search through ongoing insolvency proceedings and provides detailed, up-to-date information about them.

The landing page (shown in Figure 3.7) consists of a search form that allows one to search and filter ongoing proceedings by different attributes and preferences. The most important ones comprise (1) the name of the debtor, (2) identification number for debtors who are also entrepreneurs, (3) birth certificate number or date of birth, (4) domicile, (5) reference number, and (6) commencement date.

Once the search form is submitted, the application returns a list of all insolvency proceedings matching the criteria and links to their respective detail pages (Figure 3.3). The detail page contains all the insolvency data described in Section 3.2.1 and the documents described in Section 3.2.2.

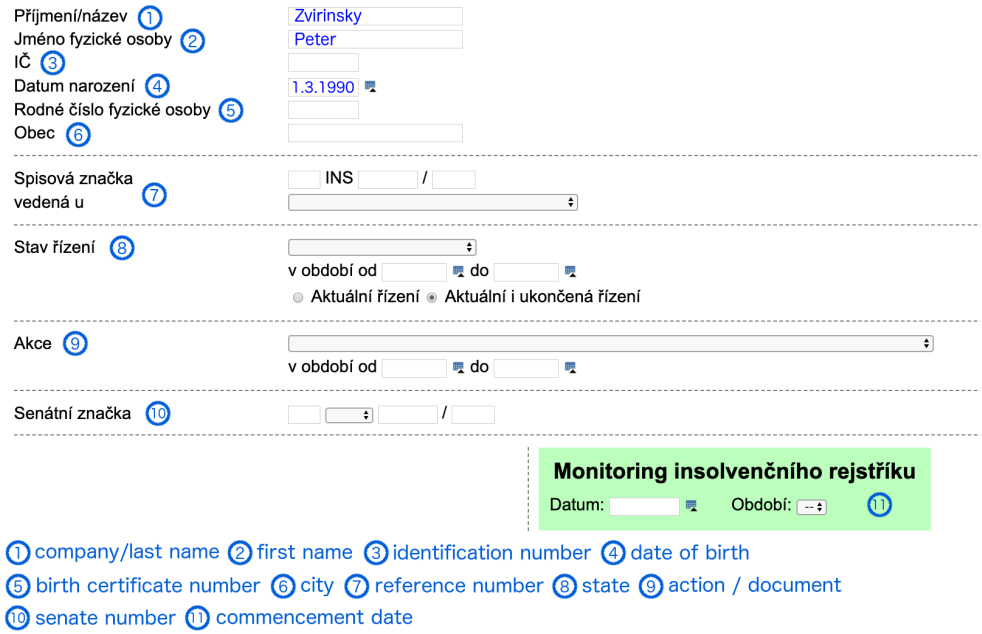


Figure 3.7: Insolvency Register's landing page with the search form.

Web Service

Unlike the web application, the web service of the IR was designed to provide a machine-readable interface for automated data processing. This interface[10] was implemented as a SOAP[97] web service on top of the classical HTTP protocol. To guarantee high availability and throughput, the web service is hosted on a different physical server than the web application, even though they are both accessible from the same domain[11]. For the Ministry of Justice, the availability of the web application has a higher priority than the web service because a much broader set of people uses the web application. Just like the web application, the access to the web service is not restricted in any way and can, therefore, be utilized at all times.

Despite having no restrictions for the web service, its usage is regulated to prevent the system from overloading, and it is recommended to send up to 1 request every 30 seconds. Any user (identified by his/her IP address) who overuses the web service is blocked by the maintainers of the web service.

Another difference compared to the web application is that the web service does not provide search and filtering capabilities. Instead, the interface adopted a publish-subscription [98] design where the web service consumer only receives

[10]The WSDL definition of the interface is available at: `https://isir.justice.cz:8443/isir_public_ws/IsirWsPublicService?wsdl`. Accessed at 21 November 2023

[11]The Insolvency Register's web service is hosted at: `https://isir.justice.cz:8443`

incremental changes to the database. The user is responsible for creating his internal database from these increments and structuring it to fit his/her needs.

Because of the publish-subscribe design, all the insolvency data from the web service are provided as so-called events. For every change in an insolvency proceeding, an event is generated and immediately published by the web service. The very first event shared by all insolvency proceedings indicates its creation. All the following events can result from submitting documents or changing insolvency proceedings states. An example of an event in its original form is shown in Listing 3.1. Finally, it is essential to note that in the publish-subscription model, once an event is published, the event does not change. The effects of the event can only be changed by a subsequent event.

```
<soapenv:Body>
    <ns1:getIsirPub0012Responsexmlns:ns1=urn:IsirPub001/types>
        <result>
            <cas>2008-11-01T00:00:00.000Z</cas>
            <id>522</id>
            <idDokument></idDokument>
            <poznamka><?xml version="1.0" encoding="UTF-8"?>
                <tns:udalost
                xmlns:tns="http://www.cca.cz/isir/poznamka"
                xsi:schemaLocation="http://www.cca.cz/isir/poznamka
                https://isir.justice.cz:8443/isir_ws/xsd/poznamka.xsd">
                <idOsobyPuvodce>KSJIMBM</idOsobyPuvodce><vec>
                <druhStavRizeni>NEVYRIZENA</druhStavRizeni></vec>
                </tns:udalost>
            </poznamka >
            <spisZnacka>INS 86/2008</spisZnacka>
            <typ>3</typ>
            <typText>Insolvencni navrh</typText>
            <oddil>A</oddil>
            <poradiVOddilu>1</poradiVOddilu>
        </result>
    </ns1:getIsirPub0012Response>
</soapenv:Body>
```

Listing 3.1: An event describing the submission of an **insolvency petition** (Insolvencni navrh) for a new insolvency proceeding with a file number **INS 86/2008** which enters the state **Unresolved** (NEVYRIZENA).

The XML data structure of the event is divided into two parts. The first contains common XML elements shared by all events such as **id**, and the second called **"poznamka"** (∼ **"note"**), which is a nested XML snippet which includes data specific to individual events. The content of the element "**note**" has its own XML structure defined in an XSD[99] format which can be found on the IR's website[12]. As a result, the definition and structure of "**note**" can change without directly affecting the web service interface. The elements shared by each event are listed in Table 3.2.

[12]The up-to-date XSD definition is available at: `https://isir.justice.cz/isir/help/poznamka_1_9.xsd`. Accessed at 21 November 2023

Element	Element in English	Description
Cas	DateTime	Date and time when the event occurred.
Id	Id	Unique serial id of the event in the IR.
idDokument	DocumentId	URL of the document complementing the event.
Note	Note	Variable XML note containing event specific data.
spisZnacka	FileNumber	Reference number of the affected insolvency proceeding.
Typ	Type	Event type id from the list of all existing event types[13].
typText	TextType	Event type name.
Oddil	Section	One of the sections A,B,C,D or P (as defined in Section 3.2.2) to which the complementing document belongs.
Poradi v oddilu	Order within section	Serial number of the event within the respective section.

Table 3.2: List of elements common for all events.

The structure of **"note"** is more complicated than the structure of the web service interface. It has also undergone significant changes, making it even more challenging to grasp. Therefore, the structure of the **"note"** will not be described here. Instead, the reader will be referred to the latest documentation available on the IR's web page[14].

[13]The list of all existing event types is available at: `https://isir.justice.cz/isir/help/Cis_udalosti.xls`. Accessed at 21 November 2023

[14]The full description of the note structure is available at: `https://isir.justice.cz/isir/help/Popis_WS_1_v2_0.pdf`. Accessed at 21 November 2023

4. Czech insolvency dataset

To perform the experiments in this thesis, we have to prepare a new dataset that will capture the development of insolvencies in the Czech Republic over more than a decade. The construction of this dataset is possible due to new legislation adopted in the Czech Republic (see Section 3.1). This legislation mandates that all insolvencies must be publicly available in the so-called Insolvency Register of the Czech Republic.

Insolvencies in the Czech Republic represent a relatively novel legal framework that was only introduced in 2008. For that reason, a similar dataset is yet to be created. The preparation of this dataset requires a combination of big data approaches that rely on the ability to process large quantities of data and machine learning-based document extraction methods. The result will be a dataset that provides an unprecedented amount of information about every insolvency commenced between 2008 and 2022. In the remainder of this study, we will refer to this dataset simply as the *Czech insolvency dataset* (CID).

In this section, we will describe the preparation of the CID in detail since it will serve as the basis for all the experiments in this study. Every experiment in this work will use the CID to derive its sub-dataset and use it to perform specific experiments. We adopted this progressive approach because we developed the CID over almost eight years and extended it multiple times. As a result, we always executed individual experiments against a specific snapshot of this dataset. For simplicity's sake, we will describe the dataset construction as if someone were to create it from scratch.

4.1 Dataset schema and storage

The data from the IR that we wanted to extract would not fit into a simple tabular format because of its complexity. We decided to store all the data in a relational database. Given the relatively small number of data entities in our schema and the straightforward relationships between them, we found this storage ideal. We also preferred a relational database due to its strong support for enforcing data consistency and its ability to perform all the necessary data transformations we required for the experiments in this study. We show the relational model we used to represent the insolvency data in Figure 4.1. The complete description of each entity, including all their data fields, is provided in Appendix A. We will refer to this database for the remaining part of this study as the *CID DB*.

We summarize the disk space required to store the data for each entity in Table 4.1. These amounts are well in the range of what today's relational databases can handle effectively. Therefore, at no point did we have to consider alternative data stores, such as *NoSQL* databases, designed for more complicated data structures and much larger amounts of data. Finally, we used the *PostgreSQL*[1] relational database for this study.

[1]PostgreSQL homepage: `https://www.postgresql.org/`

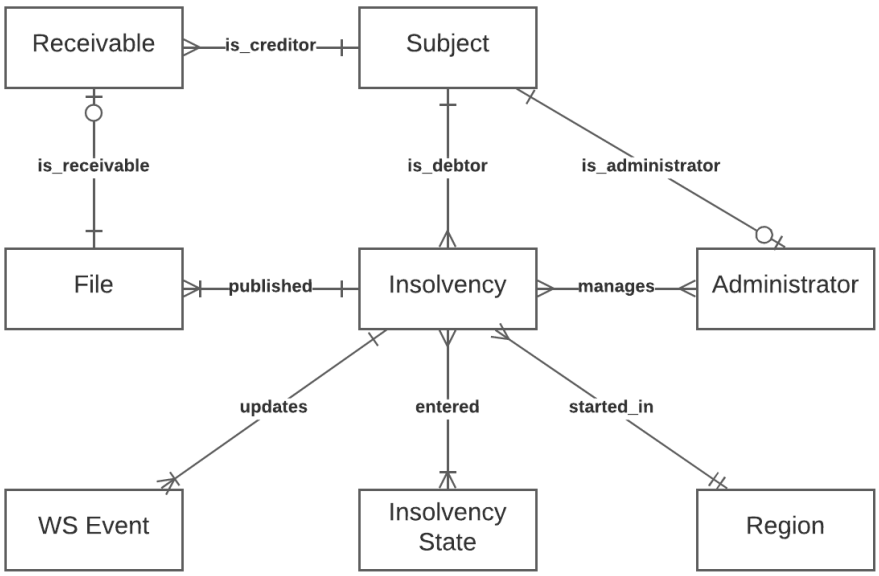


Figure 4.1: Entity-relationship (ER) model of the data extracted from the Insolvency Register.

Table 4.1: The estimations for the data size in terms of both the number of entities and storage space required to store them in PostgreSQL.

Entity	**# entities**	**Storage space**
Insolvency	375,000	160 MB
Subject	180,000	23 MB
Administrator	2,000	$<$ 500 kB
Creditor	140,000	$<$ 500 kB
Insolvency State	1,200,000	1,100 MB
Document	20,000,000	49 GB
Receivable	3,500,000	600 MB

4.2 Insolvency Register data extraction

The IR provides two ways to access the stored data. It is either through the web application or the web service (API) interface (see Section 3.2.5). Both interfaces provide slightly different views of the same data. For instance, the web application's purpose is to provide the general public with a simple current view of every insolvency. On the other hand, the web service was created for advanced users that require a complete view of each insolvency, including its entire history.

To create the CID, we will use both interfaces. First, we scrape the web application to obtain the list of every IP and the current data about individual IPs from their respective detail pages. For example, from the insolvency detail, we scrape data such as the debtor's information (name, address, age), the list of creditors, and the list of documents. Then, we use the web service data to enrich this basic data structure for additional information, such as the history of administrators assigned to the particular IP and its state development.

4.2.1 Web application scraper

To extract data from the IR's web application, we implemented a web scraper that mimics the activity of a hypothetical user that browses through all IPs commenced within a given range of dates. The scraper visits every IP's detail page, scrapes the relevant data from the HTML page using XPath[100], and stores the result in CID DB. The scraping process is described in Algorithm 1.

Algorithm 1 The IR web scraping algorithm

```
1: Input
2:     dateFrom  start date for the scraper (e.g., January 1st, 2019)
3:     dateTo    end date, or until which date the scraper should run

4: procedure scapeIsir(dateFrom, dateTo)
5:     currentDate ← dateFrom
6:     while currentDate ≤ dateTo do
7:         listingHtml ← fetch(
               https://isir.justice.cz/isir/ueu/vysledek_lustrace.do
               ?spis_znacky_datum=currentDate
           )
8:         detailUrls ← XPathQuery(
               listingHtml,
               //table[@class='vysledekLustrace']//tr[td//text()
               [contains(.,'currentDate')]]//a/@href
           )
9:         for detailUrl in detailUrls do
10:            scrapeInsolvencyDetail(detailUrl)
11:        end for
12:    end while
13: end procedure

14: procedure scrapeInsolvencyDetail(detailUrl)
15:    Extract IP data from the detailUrl using a series of XPath queries.
16:    Store the extracted data in CID DB
17: end procedure
```

The *scrapeInsolvencyDetail* procedure extracts all the data available on the IP detail page except for the birth certificate number and the date of birth. This data is protected under the Personal Protection Data Act[2] and should not be stored by third parties. For this reason, we only extract the birth year and information derived from the birth certification number, such as gender. We store the output of the web scraper as the following entities in the CID DB: *Insolvency*, *Region*, and *File*. For the complete list of extracted attributes refer to Appendix A.

In the Czech Republic, gender and birth date are unambiguously embedded into the birth certificate numbers[3] of the holders. The process works as follows. First, the birth certificate number is a nine or ten-digit number, such as

[2]Act No. 101/2000 Coll on personal data protection.

[3]The method for generating birth certificate number is defined by Act No. 133/2000 Coll.

1272127890. The first six digits represent the date of birth in the format YYMMDD[4]. The remaining 3 or 4 digits are randomly selected to distinguish people born on the same day. Lastly, for every female, a value of 50 or 70 is added to their birth month — that is why the example number contains the number 72 as the month of birth (which corresponds to month 2, i.e., February).

The maintainers of the IR did not set any access restrictions for accessing and using the register, so it would be possible to scrape data without any limitations. However, preventing the host server from overloading and limiting the number of requests submitted per a given time frame is appropriate. Therefore, the web scraper we implemented uses an adaptive policy that waits for $2 * t$ seconds between each download, where t represents the time it took to download the last page. This policy represents a compromise between scraping speed and excessive usage of the system. With this policy in place, it was possible to scrape all the insolvency proceedings available in the register in approx. seven days.

4.2.2 Web service scraper

The web service interface provides two methods for querying the data in a machine-readable format. Both methods are called $getIsirPub0012$, but have a different set of input parameters. The first method takes an integer parameter representing the event's ID and returns all events with an ID larger than the input ID. The second method takes a date parameter corresponding to the event's occurrence date and returns all events that occurred after the given date. Both methods only return up to 1000 events in a single call.

To extract data from the IR's web service, we implemented a web service scraper which works in two phases. First, we store all the newly published events from the web service in CID DB (*WS Event* entity). Then in the second phase, the scraper runs through every new event and extracts additional data about the IPs. Specifically, we extract the information about the change of state of the IPs (described in Section 3.1.5) and the information about administrators being assigned/unassigned to specific IPs. Finally, we store the output of the web service scraper as the *Administrator* and *Insolvency State* entities.

By January 2023, approximately 51,000,000 events have been published in the IR's web service. Therefore, to scrape all events, roughly 51,000 requests are necessary (the web service only returns up to 1,000 events per 1 request). Considering the 30-second restriction stated in Section 3.2.5, it would take approximately 18 days to scrape all published events. Thus, scraping the entire IR, including the web application (7 days) and the web service, would take approximately 25 days.

4.3 Extracting data from documents

The data obtained from the IR come in two forms: structured and unstructured. The structured data describe basic information about every IP, such as its current participants, i.e., the debtor, administrator, and the list of creditors.

[4]Y — year, M — month, D — day

Additionally, structured data also includes the current state of each IP and a list of documents that provide detail for every proceeding.

The semi-structured or unstructured information from the insolvency (PDF) documents that often come in the form of scans is even more profound. It can be leveraged to gain further insights into the IP and its participants. For instance, we can determine the value of each creditor's receivable, whether it is secured or unsecured, and its origin. More than 20 million documents have been published in the IR to date (see Section 3.2.2). Thus the amount of unstructured data is naturally much larger than structured data.

The data contained within the documents is crucial for this study since it will allow us to build more accurate models of debtors' behavior. Therefore, we have developed an automated data extraction system called *IRES* (Insolvency Register Extraction System) explicitly designed for extracting data from PDF documents (scanned or electronically generated) as part of this thesis. IRES consists of a document processing pipeline that includes the following three high-level steps: (1) custom document preprocessing, (2) Optical Character Recognition (OCR), and (3) structured data extraction. We will now describe the individual components of the document processing pipeline and explain how they work together to maximize extraction accuracy.

The complete data-gathering process is visualized in Figure 4.2. The process starts with gathering structured data and storing them in our database using web scrapers. Then, the documents are processed separately by our document processing pipeline. Finally, the extracted data flows back into CID DB in the form of additional entities, such as *Receivable* in Figure 4.1.

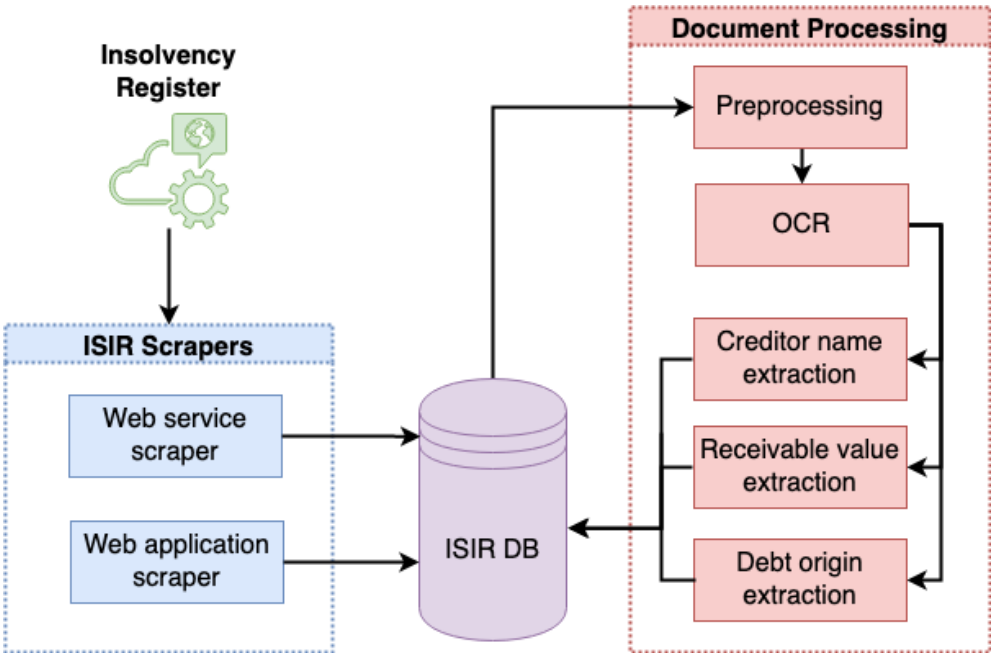


Figure 4.2: The complete process of extracting data from the Insolvency Register.

4.3.1 Optical Character Recognition

The most challenging step of the proposed data extraction system is Optical Character Recognition (OCR), which converts PDF documents back to electronic text. The accuracy of the adopted OCR solution will be the most significant determinant of the overall accuracy of our extraction framework. Given the large volume of documents we have to process, this step will be computationally the most expensive one.

The general OCR process typically consists of the following three steps: (1) document preprocessing, (2) image segmentation, and (3) character recognition. Most OCR software (especially commercial ones) comes with all three components. We selected Tesseract [101] as our base OCR framework for this study. As an open source project, Tesseract performs very well compared to existing commercial tools such as ABBYY FineReader[5], Transym OCR[6], and AWS Textract[7]. A detailed technical comparison between Tesseract and Transym OCR is available in [102].

To validate our decision to use Tesseract as our main OCR framework, we performed a simple test that compared Tesseract's performance on ARs with two commercial tools mentioned above, namely AWS Textract and ABBYY. For this test, we first created an evaluation dataset of 500 ARs. Next, we manually extracted the full creditor name and the receivable value (in CZK) of each AR. These two values must be present in every AR, and they are also essential for the experiments in this study. Lastly, we used this dataset to evaluate how accurate the selected OCR solutions are for extracting long, mainly alphabetical strings (creditor names) and purely numerical strings (receivable values).

The test we applied to all 3 OCR solutions consisted of the following 4 steps:

1. perform OCR on every AR and save the obtained texts
2. perform a text search for the corresponding creditor name in the obtained text
3. perform a text search for the corresponding receivable value in the obtained text
4. count the number of matches for both searches

This test gave us a good approximation of the extraction accuracy that we can expect from each solution when applied to ARs. We anticipate the OCR software to correctly recognize both the creditor name and the receivable value in the AR. Thus, both the creditor name and the receivable value should be present in the obtained text, and finding them using a simple text search should be straightforward. On the other hand, the only reason we would not find these two texts in the OCR result would be that the given OCR solution failed to process a specific document.

[5] ABBYY FineReader homepage: `https://pdf.abbyy.com`
[6] Transym OCR homepage: `https://transym.com`
[7] AWS Textract homepage: `https://aws.amazon.com/textract`

The results for AWS Textract, Abby Fine Reader, and Tesseract are shown in Table 4.2. The results show that out-of-the-box Tesseract performs very poorly on the test dataset without any preprocessing (denoted as *Tesseract original*). For example, it correctly recognizes the creditor name in only 69.3% of cases and the receivable value in only 63.1% of cases. The following section will describe how we used advanced document preprocessing techniques to improve Tesseract's performance significantly.

4.3.2 Document Preprocessing

Tesseract's advantage is its very sophisticated character recognition engine built using neural networks [103]. That is why it typically performs on par with commercial solutions [104]. On the other hand, Tesseract's limitation are its image segmentation capabilities, that cannot handle complex documents containing forms and tables. Tesseract also comes with limited image preprocessing capabilities which degrades its performance on low-quality scan images.

We investigated the OCR failures on our test dataset and realized we could attribute most of them to the two limitations described above. The first and most prevalent issue that accounted for more than 80% of errors was related to the structured nature of ARs, which includes document forms and tables. These structures proved to be beyond Tesseract's basic segmentation capabilities. As a result, Tesseract missed many texts in forms and tables, which explained the poor results shown in Table 4.2. The second issue, which covered most of the remaining errors, stemmed from documents scanned with low resolution and poor quality, affecting Tesseract's core character recognition capabilities.

The document (or image) preprocessing step is crucial for every OCR task, and can lead to significant improvements in terms of overall accuracy [105][106]. Therefore, we designed a custom document preprocessing pipeline for insolvency documents. The first goal of this pipeline was to simplify the document structure by removing all nontextual artifacts from the pages, which included mainly form and table outlines (borders). The simplified page structure would significantly improve the odds of Tesseract's built-in segmentation algorithm capturing all the text on the page. The second goal of the preprocessing pipeline was to improve Tesseract's performance on poorly scanned documents by denoising and straightening (deskewing) the images before the OCR step.

We used existing image manipulation tools such as ImageMagick [107] and OpenCV [108] to build the preprocessing pipeline. First, we used ImageMagick to binarize the image using Otsu's thresholding method [109], which effectively removed background noise from scanned documents. Next, we used ImageMagick's ability to deskew document pages, and finally, we used OpenCV to detect and remove lines utilizing Probability Hough Transform [110].

By revisiting Table 4.2, we can see that these three preprocessing steps significantly improved Tesseract's OCR accuracy on ARs. For example, the creditor name and receivable value extraction accuracy improved to 92.1% and 93.4%, respectively. Finally, we demonstrate the impact of the preprocessing steps on Tesseract's OCR accuracy on a sample document in Figure 4.3.

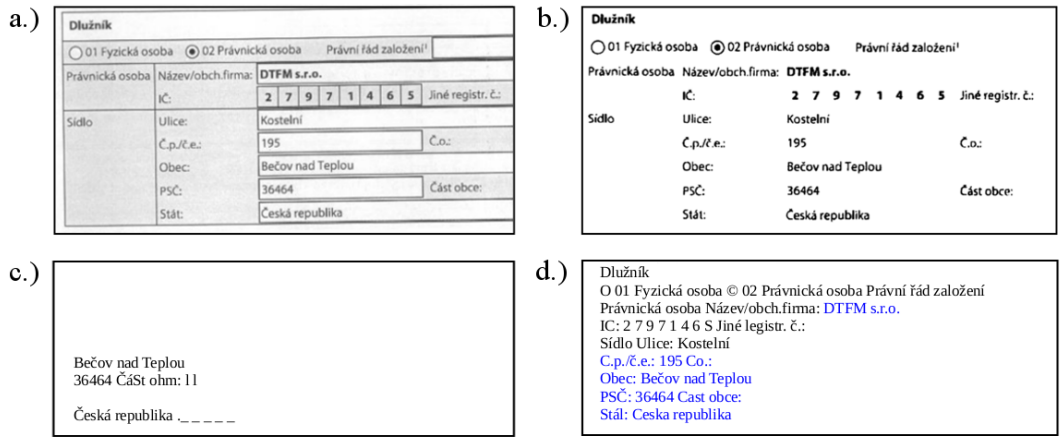


Figure 4.3: OCR process: (a) original (slightly skewed) scanned document, (b) pre-processed document, (c) text extracted from the original document (a), and (d) text extracted from the pre-processed document (b).

Table 4.2: OCR performance comparison

	Tesseract original	**Tesseract prepro-cessed**	**AWS Textract**	**Abby Fine Reader**
Creditor name match	69.3%	92.1%	**97.2%**	91.4%
Receivable value match	63.1%	**93.4%**	85.5%	87.3%
Creditnor & rec. value match	59.4%	**90.6%**	83.5%	80.2%
Mean runtime per page (sec.)	11.2	21.4	N/A	**9.5**

We can see that our custom OCR processing framework outperforms both commercial solutions by a relatively large margin (7.1% improvement over AWS Textract and 10.4% improvement over Abby Fine Reader in creditor name & receivable value match task), which shows how important domain specific image preprocessing is for the OCR task at hand. These conclusions are in line with findings from similar studies such as [105][106]. This test was performed on a 2017 MacBook pro with a 2.2 GHz Quad-Core Intel Core i7 processor and 16 GB 1600 MHz DDR3 RAM.

For total transparency, we must state that AWS Textract was at a slight disadvantage in this test because, officially, it still only supported English-written documents at the time of writing. However, despite this fact, AWS Textract outperformed all solutions, including ours, by a large margin on the creditor name match task. On the other hand, Textract performed significantly worse on the receivable value extraction task (only 85.5% accuracy), which was surprising since there is no language barrier related to the extraction of numerical values.

Abby Fine Reader performed more consistently on both creditor name (91.4%) and receivable value (87.3%) tasks in terms of accuracy. However, these results were still significantly worse than Tesseract with preprocessing and even worse in overall accuracy (80.2%) than AWS Textract (83.5%).

Table 4.3: AWS Textract pricing for the Frankfurt Data Center[a]

Monthly	**Price per page**	**Effective Price per 1,000 Pages**
First 1 Million pages	$0.001875	$1.875
Over 1 Million pages	$0.00075	$0.75

[a]Cost were obtained from `https://aws.amazon.com/textract/pricing/` on September 22nd, 2020

4.3.3 Scaling up to millions of documents

As of January 2023, the IR contained more than 20 million documents related to more than 370,000 IPs, and this number continued to grow year from year. These documents belonged to approximately 900 different types and included various forms, submissions, and statements related to the individual IPs. For instance, every IP is commenced by submitting an *insolvency petition* by the debtor or one of the creditors. The most common document type in IR is the *application of receivables* (AR), which we described in Section 3.2.2.

Given this vast amount of documents, we limited our focus to only a smaller subset of these documents. We reasoned that ARs are the most useful for this study since they will provide us with information about all creditors in the IR. In addition, they will tell us the debt size and the reason behind the debt. However, even focusing on ARs alone meant processing more than 3.5 million documents, on average six pages[8] long. Restricting ourselves to just a subset of pages in every AR was not an option because the information we looked for could occur on any page in the document, and we could not determine the relevant pages in advance. Therefore, the task ahead was to OCR and extract information from approximately $3.5M \times 6 = 21M$ pages in a reasonable amount of time on a minimal budget.

In the rest of this section, we will consider the economic feasibility of the individual solutions we have proposed so far. Then, we will determine which one is the most cost-effective for processing all 3.5 million ARs.

Estimating the total costs for AWS Textract is straightforward since the usage of the service is billed on a per page basis, and there is a volume discount after processing more than 1M pages in a single month. The complete pricing table is shown in Table 4.3. We assumed that we would process all ARs within a single month in our calculation, resulting in a total price of $(1,000,000 \times \$0.001875) + (20,000,000 \times \$0.00075) = \$16,875$ for processing all ARs.

The pricing structure for Abby Fine Reader is much more opaque, and we could not accurately estimate the licensing cost of the on-premise deployment. That is why we considered Abby's cloud service instead. The pricing structure is also on a per-page basis, and it is similar to AWS Textract. The user first pays a fixed

[8]One page has a standard size of A4.

Table 4.4: Abby Cloud OCR SDK plans and pricing[a]

Price per month	$29.99	$99.99	$199.99	$299.99	$839.99
# pages per month for free	500	2000	5000	10000	30000
Price per additional page	$0.06	$0.05	$0.04	$0.03	$0.028

[a]Plans and pricing were obtained from `https://www.ocrsdk.com/plans-and-pricing/` on September 22nd, 2020

monthly fee to be able to process a limited number of pages for free. After that, every additional page beyond this limit is billed separately (see Table 4.4). The total cost of this solution, assuming we would process all documents within one month, is approximately $\$839.99 + (21,000,000 - 30,000) \times \$0.028 = \$587,999.99$.

To estimate the cost-effectiveness of our custom Tesseract OCR solution, we assumed the computation resources would be provided by Amazon AWS. More specifically, the AWS EC2[9] service, which rents virtual servers billed on an hourly basis. We also assumed that the computation requirements would be the same ones[10] as estimated in Table 4.2, i.e., the processing of a single page takes approximately 21.4 seconds.

Since our goal was to minimize the costs for OCR processing, we selected the cheapest family of EC2 Instances called Spot Instances. Spot Instances are generally 80% cheaper than regular on-demand EC2 instances. However, they come with a downside: they can be terminated at any given moment, and their price is determined dynamically by the current demand in real-time. Consequently, their price increases with the demand, and if someone is willing to pay more for a given Spot Instance, then the instance is terminated and provided to the higher bidder.

The low price of Spot Instances is compensated by increased infrastructure complexity since the user must anticipate that the instance can be stopped at any moment. However, despite this obvious downside, Spot Instances are well suited for batch processing, when the intermediate results can be stored on external storage. The OCR processing of a vast amount of documents falls into this category of computation. A terminated OCR computation can be quickly resumed, and it can pick up where it left off.

The price breakdown for different types of AWS Spot Instances is provided in Table 4.5. For this study, we selected the compute-optimized instances (starting with the letter *c*). We can see that even though the cost of the instances increases with the number of virtual CPUs (vCPUs), the cost per a single vCPU remains roughly the same. As a result, we can calculate the prices assuming single core computation. The computation speed-up is then just a matter of introducing more vCPUs by allocating more Spot Instances for the computation, which will not significantly affect the resulting price for processing all documents.

[9]Amazon EC2: `https://aws.amazon.com/ec2`

[10]The OCR runtime statistics on AWS instances were comparable with those obtained on our custom hardware.

Table 4.5: AWS Spot Instance pricing for the Frankfurt Data Center[a]

Instance Type	**# vCPUs**	**Cost per hour**	**Cost per core per hour**
c5.large	2	$0.0333	$0.0166
c5.xlarge	4	$0.0681	$0.0171
c5.2xlarge	8	$0.1386	$0.0173
c5.4xlarge	16	$0.2718	$0.0169
c5.9xlarge	36	$0.6453	$0.0179

[a]Cost were obtained from `https://aws.amazon.com/ec2/spot/pricing/` on September 22nd, 2020

Table 4.6: Total OCR costs for processing all 3.5M ARs by individual solutions

OCR Method	**Cost per a single AR page**	**Total cost for 2.8M receivables**
Tesseract on AWS Spot Instances	**$0.000100**	**$2,100.00**
AWS Textract	$0.000817	$16,875.00
Abby Cloud OCR SDK	$0.028000	$587,999.99

We can now calculate the final cost of our custom Tesseract OCR solution running on AWS Spot Instances using all the available pricing information. Let us assume that we can process approximately $60 \times 60/21.4 = 168.2$ pages on a single vCPU in an hour and that the cost of one vCPU-hour is, on average, $0.01696. Then, the total cost for processing a single page equals $\$0.01696/168.2 = \$0.0001$, and the total cost for processing all 21M pages of ARs is approx. $21,000,000 \times \$0.0001 = \$2,100.00$.

The total costs for all three solutions are summarized in Table 4.6, and as we can see, the cost differences are significant. The most expensive solution is Abby Cloud OCR, with a total cost of $587,999.99 for processing all ARs. This price is entirely unrealistic for us, given our limited budget. Therefore, we can conclude that Abby Cloud OCR is a viable solution only for much smaller batches of documents than we need. AWS Textract, on the other hand, is significantly more cost-effective, totaling in $16,875.0 for processing all ARs. This cost is much more reasonable, especially considering the relatively high extraction accuracy we obtained in our tests. Finally, our custom solution built on Tesseract and deployed on AWS Spot Instances is the most economically viable solution, with a total cost of only $2,100 for processing all ARs. Given that this solution also has the highest extraction accuracy, we can conclude that it is the best for the task at hand.

4.3.4 Extracting missing creditor names

The IR provides the name of the creditor who submitted the given AR, and it can be obtained through both the web application and the web service. However,

this is true only for IPs commenced after October 2011. Therefore, this information is not available in the approximately 50,000 IPs commenced before October 2011 (affecting more than 270,000 ARs). This information is missing in the early years of the IR because the necessary functionality was not implemented yet. As the IR evolves over time, so does the scope of information it provides.

The missing creditor names affected 50,000 out of the total 375,000 IPs (13%), and approximately 270,000 ARs of the total 3.5 M of all ARs submitted to the IR (8%). This is not necessarily a large number, but given that it only affects IPs from a particular time period, it would mean that we could miss essential information from the first three years of the IR's existence. Since one of the main goals of this study is to model the development of the entities in the IR, these early years are crucial.

The only way to identify the creditors in the early 270,000 ARs is to look into the document and look up the creditor by the name or the company ID. To avoid the manual processing of approximately 270,000 documents, we decided to leverage the texts extracted from all ARs by our OCR pipeline. We used the texts of newer ARs for which the creditor name was already available and used them to train a text classification model that could automatically assign missing creditor names to older ARs.

Table 4.7: The 19 most frequent creditors in the Insolvency Register in 2012

Creditor	**# ARs**
GE Money Bank a.s.	12,517
Česká spořitelna a.s.	6,588
CETELEM CR a.s.	6,261
Home Credit a.s.	5,850
PROFI CREDIT Czech a. s.	4,721
SMART Capital a.s.	4,564
ESSOX s.r.o.	4,088
T-Moble CR a.s.	3,783
Provident Financial s.r.o.	3,666
Komerční banka a.s.	2,929
Všeobecná zdravotná poistovňa, a.s. (VZP)	2,911
Cofidis s.r.o.	2,904
ČEZ Prodej a.s.	2,653
O2 CR a.s.	2,646
ČSOB a.s.	2,473
Bohemia Faktoring a.s.	2,324
Česká podnikatelská pojišťovna a.s.	2,039
Citibank Europe plc	1,919
Raiffeisenbank a.s.	1,744

The distribution of creditors in the IR is skewed, and a small number of large creditors submit the majority of receivables. For instance, in 2012, which was the

first year all creditor names were available in the IR, 182,045 ARs were submitted by 26,184 different creditors. Out of those, 76,580 ARs were submitted by the 19 most frequent creditors, representing 42.1% of all ARs submitted that year. Given the relatively short period of missing information (2008 to October 2011), we assumed that the distribution of creditors would be similar to those in 2012. Therefore focusing only on the top 19 creditors should yield missing creditor labels on a large portion of the early 270,000 ARs. We list the selected creditors in Table 4.7.

For this text classification task, we selected the ARs submitted by the 19 most frequent creditors in 2012, and we used their names to generate a labeled dataset as follows. First, for each of the 19 creditors, we randomly sampled 1000 ARs. Then, we created an additional 20th class to represent all the 'other' creditors and sampled an additional 2,000 ARs submitted by any of the remaining creditors. Thus, the final training dataset consisted of 21,000 samples (ARs), each labeled by one of the 20 classes.

Before training, we converted the document texts to lowercase, and we removed all Czech stop-words[11]. Next, we removed all low-frequency words that did not occur in at least 100 documents. Similarly, we removed too frequent words occurring in more than 50% of all documents. Finally, we transformed the obtained texts into a bag of n-grams [111] representation (word level n-grams and n ranged from 1 to 3). The final dictionary contained 34,740 different n-grams.

We used this dataset to train several different classifiers, including the Naive Bayesian classifier, logistic regression, linear kernel SVMs [112], and Extreme Learning Machines [113] (ELMs) with 500 and 5000 hidden neurons. We evaluated the individual classifiers using 10-fold cross-validation and summarized the results in Table 4.8.

Table 4.8: 10-fold cross-validation results for the creditor classification task. We report Recall, Precision, F-measure and Training time for each of the considered algorithms along with the corresponding 95% confidence interval.

Classifier	**Recall**	**Precision**	**F-measure**	**Training time (sec.)**
Naive Bayes (multinomial)	0.854 ±0.005	0.828 ±0.005	0.824 ±0.005	1.143 ±0.027
Logistic Regression	0.966 ±0.002	0.965 ±0.002	0.965 ±0.002	235.004 ±1.053
SVM (linear kernel)	0.960 ±0.002	0.959 ±0.002	0.959 ±0.002	59.176 ±1.729
ELM (500 hidden neurons)	0.813 ±0.006	0.828 ±0.006	0.817 ±0.006	43.942 ±0.117
ELM (5000 hidden neurons)	0.943 ±0.003	0.942 ±0.003	0.941 ±0.003	3 736.228 ±2.885

The best-performing classification algorithms were logistic regression and SVM (linear kernel), with reported F-measures equal to 0.965 and 0.959, respectively. The ELM model with 5,000 hidden neurons also performed very well (F-measure equal to 0.941). However, given its very long training time (3,736.228 seconds on average), it was unsuitable for the given task. The multinomial Naive Bayesian classifier and ELM with only 500 hidden neurons performed significantly worse, reporting an F-measure of less than 0.95.

[11]The stopwords were obtained from `https://github.com/stopwords-iso/stopwords-cs` on October 10th, 2020

To evaluate the importance of the individual *n-grams* for classification, we also performed correlation analysis, and we show the correlation matrix for selected creditors and the corresponding *n-grams* in Figure 4.4. Correlation analysis shows that parts of creditors' names, for instance, *'cetelem'* or *'home credit'*, are among the most discriminative features. Other essential features include parts of creditors' addresses, e.g., *'karla'* from Cetelem's company headquarters address (Karla Engliše 5/3208) and similarly *'michle'* which is a neighborhood of Prague where GE Money Bank's headquarters resided in 2012. In Figure 4.4, we can also notice company identification numbers, such as *'25085689'*, and data box addresses, such as *'i48ae3q'*. The last category of essential *n-grams* for classification were the names of lawyers representing the creditors in the IP (e.g., *'sona'*, which was the first name of the lawyer commonly representing Home Credit in 2012).

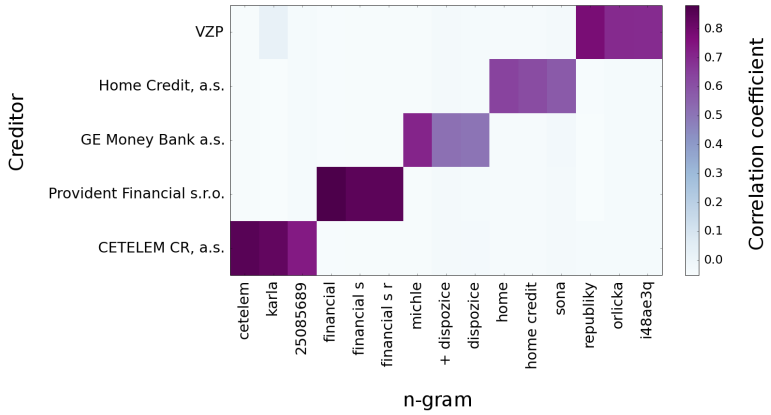


Figure 4.4: Three selected n-grams (n ranging from one to three) most correlated with five selected creditors.

Finally, the best-performing classification model, logistic regression, was used to determine the missing creditor names. As a result, out of the 270,000 ARs published before October 2012, we successfully assigned a creditor name to approximately 104,000 ARs (38.5%). We inserted the assigned creditors in the *File* entity of CID DB to fill in missing creditor names from the web application scraper. Using these classifications, we can now study the development of insolvencies since their adoption in 2008.

4.3.5 Extracting receivables' values

Another essential piece of information included in every AR is the debt value claimed by the creditor in the IP. Unfortunately, this detail is available only in the ARs and must be therefore extracted directly from the documents once again. For this purpose, we will reuse the texts we already obtained from running our OCR pipeline one very AR.

The claimed debt can be of two primary types, secured and unsecured. A collateral backs secured debts to reduce the risk associated with lending. This collateral typically corresponds to some asset the debtor owns, like a house or a

car. In the case of secured debt, the creditor has the right to seize the property and use the proceeds from selling it to recoup the losses he/she accrues when the borrower defaults on the loan. Secured debt usually represents larger loans such as mortgages or company loans.

On the other hand, unsecured debt is not backed by any collateral. As a result, the creditor has only limited options to mitigate his/her losses in case of a default. Unsecured debt often originates from smaller consumer loans, credit card debts, or unpaid phone and utility bills. A combination of both types of debt is possible, e.g., a part of a loan can be secured while the other is not.

The AR must state exactly how much money the debtor owes to the given creditor, and the overall debt corresponds to the sum of secured and unsecured receivables. (see Figure 4.5). For an AR to be valid, it must contain at least these three values (secured, unsecured and total).

47 Celková výše přihlášených pohledávek (Kč):	11 064 122,29
48 Celková výše nezajištěných pohledávek (Kč):	11 064 122,29
49 Celková výše zajištěných pohledávek (Kč):	0,00

Figure 4.5: The total value of the applied receivables ("Celková výše přihlášených pohledávek (Kč)", line 47) can be determined as the sum of unsecured ("Celková výše nezajištěných pohledávek (Kč)", line 48) and secured ("Celková výše zajištěných pohledávek (Kc)", 49) receivables.

The main challenge when extracting debt values is the lack of standardized document forms used for ARs, which was especially true in the early years of the IR. The lines and fields where these values occur are not fixed. In the example from Figure 4.5, these values occur on lines 47 through 49, but this does not hold for all ARs. The pages where these values appear also differ since ARs have variable lengths depending on the specific creditor and type of debt. As a result, we had to process the entire AR to extract the debt values.

Next, we had to tackle the inaccuracies introduced by the OCR process. The limited segmentation capabilities of Tesseract mentioned in Section 4.3.2 promise that we will likely extract the correct text. However, the precise spacing between words or the exact position is not guaranteed. Therefore, the most robust approach we identified was only to consider the text surrounding the target values in the document. Again, this is not a trivial task because the surrounding text is not fixed and is also affected by the previously mentioned OCR process inaccuracies.

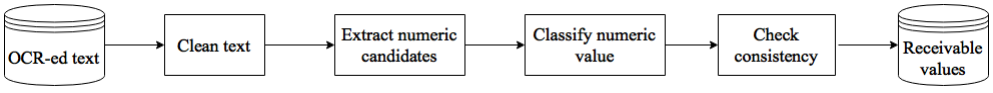


Figure 4.6: The process of extracting receivable values.

We illustrate the entire process of extracting receivable values in Figure 4.6. The first step, cleaning, consists of simple text preprocessing, including transformations such as removing redundant white spaces and newline characters and removing all diacritical marks from the text. As we were interested in numeric values only, no critical information is lost in this step. Furthermore, removing

diacritics makes the subsequent classification step more robust since OCR inaccuracies often involve incorrectly recognized diacritical marks.

Then, we extracted all numeric candidates representing a valid monetary value in Czech crowns. Because of the different conventions used to write monetary values, we had to cover multiple formats. We implemented this step by matching the preprocessed text against a series of regular expressions designed to capture all the monetary formats occurring in ARs (see Table 4.9).

Table 4.9: Examples of monetary formats used in ARs

Format	**Corresponding Regular Expression**
0.00	[0-9]1,3([ ][0-9]3)*[,.][0-9]1,2
1.000,00	[0-9]1,3([.][0-9]3)+[,][0-9]1,2
1 000,-	[0-9]1,3([ ][0-9]3)*,-

In the classification step, we used the text surrounding each numeric candidate (50 characters from the left and 50 from the right) to determine its category. Specifically, we introduced four categories: overall, secured, unsecured, and other. In the context of receivable values, the first three categories are self-explanatory, and the fourth one represented all the other numeric values that were irrelevant to our study.

We transformed the texts before and after each numeric candidate into a character-level n-grams feature space, where n ranged from 3 to 4. We selected character-level n-grams because they were more robust against typos introduced in the OCR extraction step. The resulting bag-of-n-grams vector consisted of 21,170 different n-grams. Since our goal was to detect the candidates' categories automatically, we had to create a manually labeled training dataset. For this purpose, we randomly selected 200 ARs from which we extracted approximately 7,000 numeric candidates, and we manually labeled each with its corresponding category. The majority of the candidates (6,400) were labeled as other.

Next, we used the training dataset to train and test multiple classification algorithms, including the Naive Bayesian classifier, logistic regression, linear kernel SVMs, and random forests. We summarize the performance of the individual classifiers in Table 4.10. The results show that all classification algorithms performed well on this task and reported an F-measure larger than 0.97. The best performing classifier in terms of F-measure was logistic regression which we adopted as our main algorithm for this task.

Table 4.10: 10-fold cross-validation results obtained on the candidate value dataset. We include Recall, Precision and F-measure for each of the considered algorithms along with the corresponding 95% confidence interval.

Classifier	**Recall**	**Precision**	**F-measure**	**Training time (sec.)**
Naive Bayes (multinomial)	0.974 ±0.006	**0.985 ± 0.006**	0.979 ± 0.006	**0.005**
Logistic Regression	**0.982 ± 0.001**	0.983 ± 0.001	**0.982 ± 0.001**	0.187
SVM (linear kernel)	0.981 ± 0.009	0.980 ± 0.009	0.980 ± 0.009	0.218
Random Forest	0.976 ± 0.008	0.973 ± 0.008	0.973 ± 0.008	27.609

The penultimate step in our extraction pipeline was designed to check the consistency of the extracted values. As we already established, all three values (overall, secured, and unsecured) must be present in each AR to be valid, and each of the values must be present exactly once. By definition, it must also be true that the sum of secured and unsecured values is equal to the overall value. Using this information, we grouped outcomes including the potential errors from the extraction process into the following categories: *OK*, *INCONSISTENT*, *PARTIAL* and *MISSING*. We provide the definition of these categories in Table 4.11.

Table 4.11: Consistency categories used to validate extracted receivable values.

All 3 numbers extracted?	**Secured + Unsecured = Total ?**	
	true	***false***
true	OK	INCONSISTENT
false	PARTIAL	MISSING

Lastly, we created a manually labeled evaluation dataset to estimate how the extraction pipeline would perform on real-life data. The evaluation dataset consisted of 500 randomly selected ARs. For each AR, we manually extracted the secured, unsecured, and overall receivable values, which served as ground truth. Then, we processed the ARs from the evaluation dataset with our extraction pipeline to obtain the following results: 84.1% of ARs were processed with status *OK*, 9.9% with status *PARTIAL*, 4.5% with status *MISSING*, and only 5.5% with status *INCONSISTENT*.

By examining the extraction errors on this evaluation dataset, we found that OCR failures caused almost all (99%+) errors. Furthermore, all the errors originated from ARs scanned with inferior quality. The percentage of correctly processed ARs (84.1%) is very close to the extraction accuracy measured in Table 4.2, which confirms that OCR-related errors dominate the extraction failures. Based on these results, we can conclude that our extraction process uses the obtained OCR results to their full potential. Unless we improve the OCR accuracy, it is impossible to improve the extraction accuracy of receivable values significantly.

The main benefit of the consistency check defined in Table 4.11 is that it can be performed automatically and integrated into the extraction pipeline. In addition, we can execute the consistency check after each pipeline run to track the ongoing extraction accuracy and detect potential model drift[12] over time.

We obtained the following results by running the extraction pipeline on all 3.5M ARs. First, the values were extracted with status OK in 63.2% of all cases corresponding to roughly 2.2 million documents. For the purposes of this study, values marked as OK can be mainly considered correct. Then, for the remaining part, 17.8% of ARs were processed as PARTIAL, 5.4% as INCONSISTENT, and 13.6% as MISSING. The overall extraction accuracy on all ARs is slightly lower than on our evaluation dataset, which is caused primarily by lower-quality ARs submitted before 2013. These low-quality ARs are especially prone to OCR errors and lead to a slight decrease in the observed extraction accuracy.

[12]By model drift, we mean the decay of models' predictive power due to the changes in real-world environments over time.

4.3.6 Extracting origin of debt

The next piece of information from ARs that is vital for this study is the debt origin. The debt origin is another mandatory field occurring in each AR and contains a text description detailing the reason for the receivable creation. Its purpose is to clarify how/when the debt was accrued. The insolvency administrator admits or denies a given receivable based on its debt origin. So it is in each creditor's best interest to state a reasonable and accurate debt origin.

Typical debt origins include mortgages, consumer loans, credit card debts, or insurance bills. We show an example of debt origin in Figure 4.7.

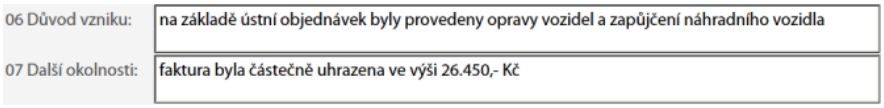


Figure 4.7: An example of a stated debt origin. Line 6 can be translated to English as: "Reason of origin: based on verbal orders, car repairs were done, and a replacement vehicle was provided" and line 7 as "Further circumstances: the invoice has been partially paid by the amount of 26.450,- Kč".

The approach we adopted to extract the debt origin was more straightforward than the approach we used to extract debt values. The reason is that in every receivable format currently used in the IR, debt origin is always preceded by a text label "Důvod vzniku" (reason of origin) and followed by "Další okolnosti" (other circumstances). Additionally, these labels uniquely identify the relevant section of the document, so simply locating these labels and extracting the text between them is sufficient to extract the debt origin reliably.

We implemented the extractor by scanning each document's text sequentially, locating the relevant labels, and extracting debt origin as the text between the labels. Again, to compensate for OCR-related errors like typos, we used Levenshtein[114] distance of at most 2 to perform label matching. Using this simple approach, we extracted the debt origin from approximately 66.3% of receivables, which corresponds to 2.32M of ARs (out of the total 3.5M). We did not store the obtained debt origins directly in the CID DB, but the extraction process can be invoked in an ad-hoc manner as needed.

4.4 Preliminary Data Analysis

The CID we prepared in this chapter is unparalleled in terms of both size and depth of information it provides about IPs. At the time, we were unaware of any other dataset from this domain that could be used to study bankruptcies at this level of detail. The data regarding bankruptcies and IPs are rarely publicly available, especially for IPs of natural persons in the wake of new privacy regulations such as GDPR[13]. The Czech Republic's IR is the most open insolvency register among all EU countries with regard to the data it provides.

[13]Find more information about GDPR at: `https://www.gdpreu.org/`

Since this dataset is entirely new, it is helpful to take a bird's eye view and understand it at the macro level before we proceed to the main experiments in Chapter 6. Furthermore, this preliminary analysis will be essential for interpreting the experiments' results.

Out of the 375,000 IPs commenced in the Czech Republic between 2008 and 2022, approximately 11% (40,000) were related to bankruptcies of organizations. The remaining 89% (335,000) were related to bankruptcies of natural persons. Out of the 335,000 insolvent natural persons, 74% (247,000) were self-employed.

The yearly development of the number of insolvencies commenced, and the volume of receivables claimed by the creditors is shown in Figure 3.1. We can see that the dynamics and trends in the IR can change significantly over the span of a few years. These changes are primarily the result of the local economy's development and the amendments introduced into the Insolvency Act. Just how radical the effect of a single legislative change can be is best showcased in Figure 3.5, which is related to one of the most recent amendments, *Act 31/2019 Coll. on Discharge and its application*. This amendment significantly relaxed the requirements for natural persons to enter an IP. Consequently, the number of IPs almost doubled from month to month.

Lastly, Figure 3.2 shows that most of the receivables claimed by creditors in terms of their value come from banks, and then a much smaller volume is driven by non-bank lenders, insurers, and other creditors.

4.4.1 Demographics

As part of the data processing pipeline, we also extracted basic demographical data about the debtors, such as their gender, age, and domicile. This information helps understand what parts of the population and regions are mainly affected by insolvencies.

Figure 4.8 shows the age distribution of debtors in the IR for both male and female debtors. The age of the debtor was determined at the time when his/her IP was commenced. We can immediately see that most debtors are between the ages of 30 and 50 and that indebtedness is generally more common among men than women. As of 2022, the retirement age in the Czech Republic is set to 65. We can also conclude that a group of approximately 17,500 (5%) debtors in the IR are already retired.

Recent amendments, such as *Act 31/2019 Coll. on Discharge and its application*, significantly relax the requirements for personal bankruptcy (discharge) for people older than 65. For instance, the amendment above reduces the payment calendar's duration from the typical five years to only three years. In addition, the minimum quota for the repaid debt, which used to be 30%, was dropped altogether. Given these favorable conditions, the number of pensioners with insolvencies is expected to grow in the following years.

The analysis of the regional distribution of IPs and its comparison to other macroeconomic statistics, such as the unemployment rate, also yields interesting insights. Since there are significant differences in population size (and the number

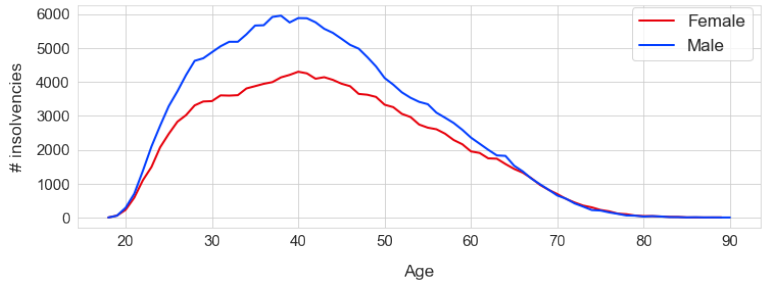


Figure 4.8: Age and gender distribution of debtors in the IR.

of companies) in individual regions, it is necessary to normalize the number of IPs by the number of citizens (or companies) in each region. We visualize the resulting IP and unemployment rates in Figure 4.9. The exact numbers used to generate the charts in Figure 4.9 are available in Table 4.12.

By first looking at the IP rates of natural persons across different regions, we can immediately see that the region with the most IPs relative to its population is the *Ústecký* region (5.15%). On the other hand, *Karlovarský* (4.54%) and *Liberecký* (4.05%) are the regions with the second and third largest IP rates. A partial connection between the IP rates and the unemployment rates in the regions above is noticeable since both *Ústecký* and *Karlovarský* regions also have one of the two highest unemployment rates in the whole country (5.26% and 4.05% respectively). The exception to this pattern seems to be the *Moravskoslezský* region which has the the second largest unemployment rate in the Czech Republic (4.87%) but has only slightly increased IP rates per capita (3.16%). As a result, a high unemployment rate does not necessarily have to indicate high IP rates.

Conversely, there is a certain correspondence between regions with the lowest unemployment rates and IP rates. For instance, *Prague* (Praha) has the lowest IP rates of all regions, with only 1.84%. It is closely followed by the regions with the second and third lowest IP rates, *Vysočina* and *Zlínský* regions, with only 2.29% and 2.51%, respectively. These regions also have relatively low unemployment rates: *Prague* 3.03%, *Vysočina* 2.69%, and *Zlínský* 2.64%.

Using company statistics from the Czech Statistical Office, we could also analyze company IP rates (Figure 4.9b). Company IPs are significantly more frequent than IPs of natural persons. Specifically, the company IP rates are mainly within the 5% to 10% range, while the IP rates of natural persons are mainly within the 2% to 4% range. The data shows that risks associated with starting and running a company are generally much higher than those individuals face in Czech society, such as unemployment or wrong financial decisions.

Interestingly enough, the two regions with one of the lowest unemployment rates, *Jihočeský* (2.63%) and *Královehradecký* (2.91%), have the highest rate of company IPs in the country (9.29% and 9.15% respectively). A similar pattern is observable in regions *Vysočina*, *Jihomoravský*, and *Zlínský*, which all have a relatively high rate of company IPs. An exception to this pattern is again the *Moravskoslezský* region which has the second highest unemployment rate in the Czech Republic (4.87%) and one of the highest company IP rates (9.39%).

(a) IPs of natural persons.

(b) IPs of companies.

(c) Unemployment rate.

Figure 4.9: Macro statistics across different regions in the Czech Republic: (a) IP rates of natural persons, (b) IP rates of companies, (c) unemployment rate. The population size of individual regions, the number of companies registered in each region and the unemployment rates were obtained from the 2022 reports of the Czech Statistical Office (source: `https://www.czso.cz`).

Table 4.12: Macro statistics used to generate all three maps shown in Figure 4.9.

Region	**Population**	**IPs of natural persons**	**IP rate (natural persons)**	**# of companies**	**IPs of companies**	**IP rate (companies)**	**Unemployment rate**
Jihočeský	637,047	20,189	3.17%	19,760	1,835	9.29%	2.63%
Jihomoravský	1,184,568	32,390	2.73%	64,183	5,969	9.30%	4.03%
Karlovarský	283,210	12,851	4.54%	9,346	620	5.63%	4.05%
Královéhradecký	542,583	20,945	3.86%	16,709	1,529	9.15%	2.91%
Liberecký	437,570	17,738	4.05%	13,916	1,067	7.67%	3.77%
Moravskoslezský	1,177,989	44,705	3.80%	38,505	3,615	9.39%	4.87%
Olomoucký	622,930	19,662	3.16%	19,903	1,226	6.16%	3.3%
Pardubický	514,518	13,697	2.66%	15,418	853	5.53%	2.54%
Plzeňský	578,707	19,139	3.31%	19,027	1,250	6.57%	2.73%
Praha	1,275,406	23,484	1.84%	219,578	12,342	5.62%	3.03%
Středočeský	1,386,824	35,065	2.53%	42,148	2,316	5.49%	3.10%
Ústecký	798,898	41,123	5.15%	20,233	1,603	7.92%	5.26%
Vysočina	504,025	11,521	2.29%	12,971	908	7.00%	2.69%
Zlínský	572,432	14,342	2.51%	19,283	1,614	8.37%	2.64%

4.4.2 Receivables and creditors

In this section we want to provide a more holistic view of the receivable market from the creditor's perspective. We want to show that the receivable market in the Czech Republic is very large and contains a large number of actors with different roles and motivations. This causes the receivable market to be very dynamic in nature which also directly affects the development of IPs in the Czech Republic.

The Czech National Bank (CNB)[14] reported that the total household debt in the Czech Republic reached 1.76 trillion CZK (76.52 billion USD) at the end of 2019. Approximately 75% of the household debt consists of housing loans and mortgages. The CNB also reported that company debts reached the total of 1.14 trillions CZK (49.55 billion USD) in the same period.

The default rate of consumer loans provided by banks[15] has mostly been in the range of 6% to 12% between years 2010 and 2020. For housing loans and mortgages the default rate was between 1.5% an 3.5% over the same period. Using these numbers we can estimate that sum of all defaulted loans is in the order of higher tens of billions of CZK, and likely over 100 billion CZK when we also include company defaults[16].

The defaulted debt is usually a subject of a collection process when the creditor tries retrieve his/her money from the debtor. Insolvencies represent only one of the stages in the debt collection process and they are usually preceded by an out of court collection stage, when the creditor tries to recover his financial losses by directly consulting the debtor. Out of court collection usually involves softer methods such as repeatedly messaging/calling the debtor, or debt refinancing.

If out of court collections fails then typically the court gets involved and what usually follows is an execution order. The execution order is issued by the court and is executed by the court appointed executors (or bailiffs). When the debtor reaches this stage, he/she usually already has several execution orders led against him/her. Based on the data provided by the Chamber of Executors of the CR[17], we know that at the end of 2019 there was approximately 790,000 persons with at least one execution order. A single debtor had an average of 5.8 execution orders and there have been 131,400 persons with more than 10 execution orders. Taking into consideration that CR has a population of 10.69 mil. we can infer that execution orders directly affect approximately 7.4% of population.

It is usually this stage when the debtor has multiple execution orders led against him/her when he decides to use the IP as the final solution to his/her debts. The moment the debtor officially files an insolvency petition, all the execution orders have to be immediately stopped. As part of the insolvency process all debtor's debts are reviewed and a suitable method of IP's resolution is decided by the court (see Section 3.1.4). Once the IP has been successfully finished, the debtor is free from all of his prior debts.

[14]CNB homepage: `https://www.cnb.cz/`

[15]CNB only gathers data on the performance of banking loans, which excludes all loans provided non-banking financial institutions.

[16]We only provide a rough estimation because the exact overall statistics for the given period was not available.

[17]Chamber of Executors of the Czech Republic homepage: `https://www.ekcr.cz/`

Now we will explain the process from the creditor's perspective and elaborate more on what happens when one of the creditor's receivables defaults. When a default occurs, the creditor usually chooses one of the following 3 general approaches to recover his receivable.

The first approach is when the creditor decides to handle the collection process by himself. In order for the creditor to take this approach, he/she must understand all the legal and procedural aspects of out-of-court and court collection. This approach of handling defaulted receivables internally is adopted mostly by larger creditors who are often financial institutions, such as banks, non-banking loan providers, or insurers. These companies usually have their own internal collection departments that focus solely on retrieving defaulted loans.

The second, probably most common approach for recovering defaulted debt is when the creditor decides to outsource the collection process to an external collection agency. Most collection agencies can cover all stages of defaulted debts and also represent the creditor in case of court proceedings. From the legal standpoint, however, it is still the creditor who owns the receivables.

The third and final option is for the creditor to transfer the ownership of the receivable to another entity (basically selling it). This is also very common for large financial institutions such as banks which typically sell whole portfolios of defaulted receivables through an auction process to the highest bidder. The buyers of these portfolios are usually institutions that specialize in investing in defaulted receivables. Since they provide the seller with immediate liquidity, they are able to buy these portfolios at a discount and then they run the collection process by themselves. The price in this transaction is typically between 10% and 15% of the original value of the receivable.

To shed more light on the insolvency portion of the market with defaulted receivables we created Table 4.13, which lists the top 20 creditors with the largest portfolios of receivables in terms of their value that have reached the insolvency stage. For reference, we also provide the number of all receivables and the average size per one receivable. The average size of the receivable can tell us more about the nature of receivables themselves. Since the value extraction process described in Section 4.3.5 is not successful in all cases, we also report how many receivables we managed to successfully process and obtain their value.

By looking at Table 4.13 we can immediately see, that the largest number of receivables does not necessary imply the largest portfolio size in terms of its total value. For instance, the largest creditors in terms of number of receivables are *Provident Financial* (77,958), *VZP* (69,605) and *GE Money Bank* (64,032). Additionally, *GE Money Bank* was re-branded as *Moneta Money Bank* in 2016 when the *GE* group decided to sell the bank and leave the local market. Between 2016 and 2022, *Moneta Money Bank* accrued an additional 27,820 receivables.

If we notice the average value of receivables of *Provident Financial* 41,790 CZK, *GE Money Bank* 99,040 CZK, and *Moneta Money Bank* 202,430 CZK, we can also tell that these are relatively small and have likely originated from consumer loans. Another non-banking financial institution in the list fitting the consumer loan pattern is *Profi Credit Czech* with 58,110 receivables and an average size of 160,400 CZK per receivable.

The top 4 creditors in the list correspond to the largest banks in the Czech Republic, namely *ČSOB*, *Česká spořitelna*, *Komerční banka*, and *Raiffeisenbank*. The average size of receivable for all of them is over 500,000 CZK. The two companies with the largest average receivable are banks *Hypoteční banka* (1,352.53 thous. CZK) and *ČSOB* (1,818.84 thous. CZK). The average size of these receivables indicates that they have likely originated from larger loans, such housing loans or mortgages.

The following major creditor in the list is *VZP* which is the Czech Public Health Insurance Company with 69,605 receivables and a smaller average size of 113.70 thous. CZK. These are smaller receivables which have likely originated from unpaid health insurance bills from individuals and self-employed entrepreneurs.

Another interesting creditor is *AB 4 B.V.* which is a Netherlands based collection agency owned by the *PPF* group which is one of the largest financial groups in the CR. *PPF* owns one of the major local banks *Airbank* and one of the largest non-banking loan providers *Home Credit*. The reason why none of these companies show up in the list is that *PPF* as a group transfers the ownership of majority of their defaulted receivables to *AB 4 B.V.* (or to *AB 5 B.V.*)

As we near the middle and the end of the list in Table 4.13 we can see companies such as *Bohemia Faktoring*, *Český inkasní kapitál*, *CP Inkaso*, and *IFIS investiční fond*. All of these companies specialize on investing in defaulted receivables and buy them from other creditors. The largest among the group is *IFIS investiční fond* with more than 11,391 receivables and a total value of 5.7 bil. CZK.

The last thing we would like to draw attention to in this section is Figure 4.10 which breaks down how the median size of IPs in terms of their total value (sum of all receivables per IP) have evolved over time. We calculated this breakdown separately for natural persons (Figure 4.10a) and companies (Figure 4.10c).

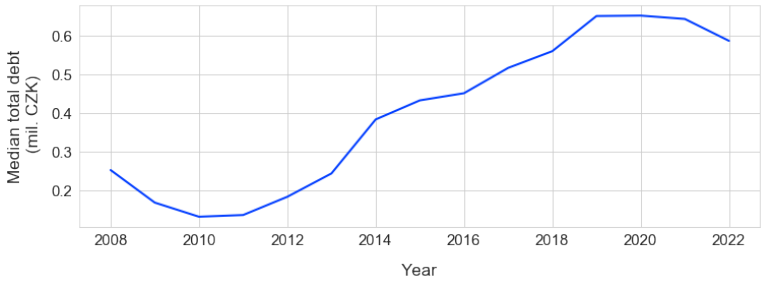
If we first look at natural persons, we can see a slightly elevated median total value of approximately 1 mil. CZK per insolvency in 2008. The following years 2009 and 2010 follow a decreasing trend which gets reversed in 2011. Since then the median size of insolvencies has been growing year-to-year up till 2020. This increasing trend can be explained by the fact that IPs became more widely adopted by debtors (especially those facing multiple execution orders) and also the fact that amendments added to the Insolvency Act in recent years had mostly relaxed the requirements for natural persons to enter an IP.

The effect of a single amendment is once again best demonstrated by *Act 31/2019 Coll. on Discharge and its application* which came into effect on June 1st, 2019. From Figure 4.10b we can see that this amendment had an immediate effect on the median size of IPs, which jumped from approximately 0.53 mil. CZK in May 2019, to 0.73 mil. CZK in June 2019.

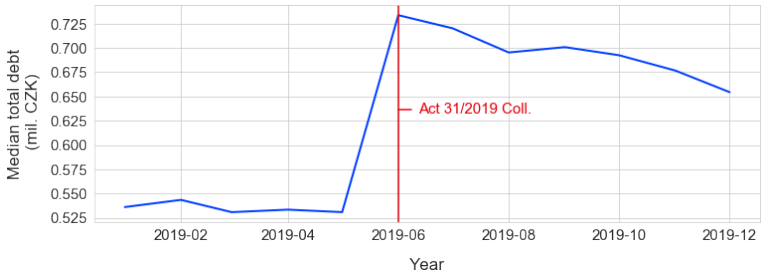
When we look at companies we can see significantly larger median sizes per insolvency which is not surprising since companies typically operate with much larger sums of money than a regular person. However, the general trend is very similar to the development of IPs of natural persons. We can also see an elevated total value of 1 mil. CZK in 2008, which then started dropping until 2010. Then, starting 2011 we can again see an increasing trend that lasts until 2018.

Table 4.13: Top 20 creditors with the largest portfolios of receivables in terms of their total value. For each creditor we report: the number of all receivables, the number of receivables from which we were able to extract the receivable value with *OK* status (see Table 4.11), the total value (sum) of all receivables, and finally, the average size of receivables.

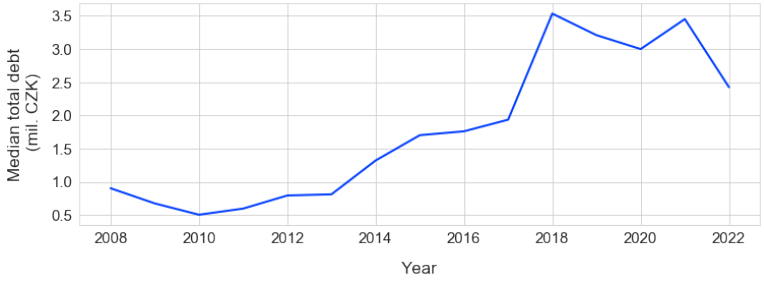
Creditor	**# all receivables**	**# of successful value extractions**	**Value of receivables (CZK)**	**Average receivable size (CZK)**
ČSOB	31,098	25,328 (81.45%)	46,067.67 mil.	1,818.84 thous.
Česká spořitelna	55,322	43,112 (77.93%)	32,801.04 mil.	760.83 thous.
Komerční banka	25,521	17,970 (70.41%)	13,775.63 mil.	766.59 thous.
Raiffeisenbank	18,676	16,010 (85.72%)	8,956.04 mil.	559.42 thous.
VZP	69,605	63,860 (91.75%)	7,899.48 mil.	113.70 thous.
Profi Credit Czech	58,110	45,131 (77.66%)	7,239.14 mil.	160.40 thous.
ČSSZ	42,782	12,358 (52.26%)	6,423.90 mil.	287.32 thous.
Hypoteční banka	7,765	4,370 (56.28%)	5,910.57 mil.	1,352.53 thous.
IFIS investiční fond	11,391	11,150 (97.88%)	5,659.55 mil.	507.58 thous.
Bohemia Faktoring	66,313	60,066 (90.58%)	5,549.73 mil.	92.39 thous.
Moneta Money Bank	27,820	25,049 (90.04%)	5,070.69 mil.	202.43 thous.
Českomoravská stavební spořitelna	10,864	8,882 (81.76%)	4,683.00 mil.	527.25 thous.
GE Money Bank	64,032	38,518 (60.15%)	3,815.06 mil.	99.04 thous.
Intrum Czech	16,455	16,011 (97.3 %)	2,915.96 mil.	2,915.96 thous.
Český inkasní kapitál	22,346	20,374 (91.18%)	2,901.69 mil.	142.42 thous.
Stavební spořitelna České spořitelny	8,985	7,761 (86.38%)	2,879.36 mil.	371.00 thous.
Provident Financial	77,958	66,832 (85.73%)	2,792.64 mil.	41.79 thous.
AB 4 B.V.	41,459	31,987 (77.15%)	2,449.05 mil.	76.56 thous.
CP Inkaso	28,121	26,563 (94.46%)	2,306.89 mil.	86.85 thous.
City Bank Europe	8,987	6,746 (75.06%)	2,052.88 mil.	304.31 thous.



(a) Median total debt of **natural persons**.



(b) Median total debt of **natural persons**, 2019 only.



(c) Median total debt of **companies**.

Figure 4.10: The median total debt is calculated by first adding up the value of receivables of individual IPs and then calculating the median over all IPs commenced in the given year. We calculated the median total debt separately for natural persons and companies. To show how the amendment *Act 31/2019 Coll. on Discharge and its application* affected the median size of debt of natural persons we also included more detailed view of 2019.

4.5 Reproducibility

The CID we prepared in this chapter is extensive, and reproducing it from scratch would be challenging. As stated in Section 4.2, just rerunning the web application scraper and the web service scraper would take 25 days. Then, rerunning the document processing pipeline on approx. 3.5M of ARs would take additional weeks or even months, based on the size of the cluster it would be deployed to. Furthermore, it would not even be possible to fully recreate our version of the CID because, as mentioned in Section 3.2.4, the insolvencies become unavailable in the IR 5 years after they have finished. As of 2022, most of the insolvencies between 2008 and 2018 can no longer be scraped from the IR.

To reduce some of the challenges stated above, we have been scraping and processing IR data continuously (daily) between 2014 and 2022. This way we

captured almost all insolvencies ever published in the IR. Additionally, this approach alleviated some of the hardware requirements too, because we would run the document processing pipeline only on the newly published ARs (approximately 1000 a day). As a result, we were processing the daily data increment on a single server instead of processing all documents in one large batch, which would require significant computational resources (tens of machines). Although, we performed one large OCR processing batch[18] at the start of our study to catch up with all the historical data. Since then, however, we were able to rely solely on these minor incremental updates.

The individual components of the daily update were executed in the following order: (1) Scrape new IPs from the IR Web Application, (2) Scrape and process all new events from the IR Web Service, and (3) Using the Web Service events, identify changed IPs and re-scrape them from the IR Web Application. In parallel with the scraping components, we ran IRES on all the newly published ARs.

For the reasons above, we included our version of the CID (i.e., CID DB) in the attachments of this thesis. This version of the CID was used to support all the experiments in Chapter 6. We describe the process of restoring the CID from the archive in Appendix B.1.

[18]The first batch included approximately 1.2M ARs, and the processing was done on AWS.

5. Definitions and tools

The main goal of this thesis is to model the data obtained from the IR using social networks. In particular, we are interested in how entities in the IR and their relationships evolve over time. To capture this evolution, we will use dynamic social networks. For an introduction to the topic of dynamic (or evolving) social networks, see Section 2.5.

As there are several approaches to the study of dynamic social networks, we will first formally define the term dynamic social network in the context of this thesis. Using this definition, we will then define a specific set of requirements that need to be met by the tools used to analyze such networks. Then, we will review how well these requirements are met by existing tools and frameworks focused on social network analysis. Finally, we will propose a new system for analyzing dynamic social networks that meets all our requirements and is named *GraphSlices*. GraphSlices is a fundamental tool for this work as it is used in the most critical experiments in Chapter 6.

5.1 Definitions

Informally, dynamic social networks can be described as graphs that change their structure over time, i.e., edges and vertices are added and/or removed from the graph at different times. Our point of view is edge-centric, meaning that the time information is only attached to the edges as a list of activity-related time spans. At any given point, the vertices thus exist in the network only if they are connected to an active edge. Internally, the input graph corresponds to a (static) multigraph with specific attributes assigned to the edges that indicate when the given edge was active in the network. Naturally, there can be multiple edges between the same two nodes at different times.

Definition 5.1.1. *A static social network is a directed graph $G = (V, E)$, where V is the set of vertices and $E = \{(v_k, v_l) | v_k, v_l \in V\}$ is the set of directed edges.*

Further, we will assume that we observe social networks across a series of discrete time points $t_1, \ldots, t_{n-1}, t_n$. At each time point t_i; $1 \leq i \leq n$, we observe an instance of a static social network $G_i = (V_i, E_i)$. The changes that may occur between two time points t_{i-1} and t_i; $1 \leq i \leq n$ include the addition or deletion of edges and the appearance or disappearance of vertices.

Definition 5.1.2. *Let n be the number of the considered time points and let us observe an instance of the static social network $G_i = (V_i, E_i)$ at each i. A dynamic social network is a directed graph $G_D^n = (V_D^n, E_D^n)$, where*

- *$V_D^n = \bigcup_{i=1}^{n} V_i$ is a set of vertices that have appeared at any of the considered time points $t_i; 1 \leq i \leq n$, and*
- *$E_D^n = \bigcup_{i=1}^{n} E_i$ is a set of edges that have appeared at any of the considered time points $t_i; 1 \leq i \leq n$, and all such edges are labeled by a sequence of*

ordered pairs $[i, j]$ *that specify the time points at which the edge was present in the network, i.e., from the time point* t_i *to the time point* t_j *in this case. The time interval* $[t_i, t_i]$ *can be abbreviated as* $[t_i]$.

To address the evolution of the network over time, we *expand* this graph into what we will refer to as the expanded graph. The expanded graph shall provide a transparent view of the network at each of the considered time points. Further, we will denote it as a slice. Expanded graphs thus incorporate multiple mutually inter-connected graphs based on the user-defined slicing function (e.g., hourly or daily). Each slice contains a copy of active edges and vertices from the original graph. New "meta" level edges are added to the network between the same vertices at different time periods to support information flow between the vertices and edges from different time slices. Figure 5.1 illustrates this unfolding process.

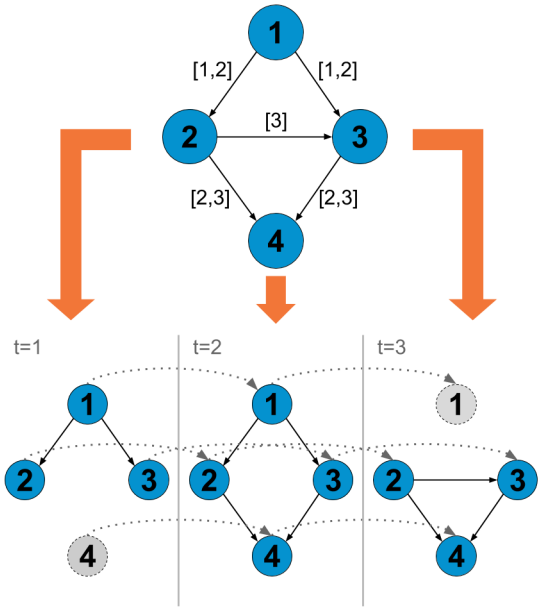


Figure 5.1: Unfolding of a dynamic social network into an expanded graph consisting of 3 time slices t_1, t_2 and t_3. A copy of each active vertex and edge is created for each time slice. Additional meta edges connect the copies of the same vertices across subsequent time slices to support information flow both within the network and across time.

We will denote the inverse operation to the expansion of the graph as *reduction* for the rest of this study. The reduction operation reverses the unfolding process to recover the original graph. It is up to the user to specify how the information from multiple copies of the original vertices/edges is supposed to be aggregated.

Definition 5.1.3. *An expanded graph of a dynamic social network defined as* $G_D^n = (V_D^n, E_D^n)$ *is a directed graph* $G_X^n = (V_X^n, E_X^n)$*, where*

- $V_X^n = \{v_k^i | v_k \in V_D^n \wedge 1 \leq i \leq n\}$ *is the set of vertices and*
- $E_X^n = \{(v_k^{i'}, v_l^{i'}) | i \leq i' \leq j$ *for any time interval [i,j] labeling* $(v_k, v_l) \in E_D^n\}$
  $\cup \{(v_k^i, v_k^{i+1}) | v_k^i \in V_D^n \wedge 1 \leq i < n\}$ *is the set of directed edges.*

The processing of dynamic networks, as defined here, obviously places new requirements on the graph mining tools used to analyze such networks. In particular, these requirements include:

1. Built-in support for the representation of evolving networks and their manipulation by time-dependent operations such as slicing.
2. Support for large-scale computation and parallelism, which allows processing very large graphs (with millions or tens of millions of edges).
3. Provides a sufficient collection of algorithms for analyzing social networks.
4. The tool is easy to use and extensible.

5.2 Existing tools

The recent rise in the availability of network-structured data, mostly from social networks (e.g., Facebook, Twitter, LinkedIn) and other sources, has created an increased demand for the processing of large graphs. However, most existing graph mining tools applicable to social network analysis are oriented toward static (unchanging) graphs. On a high level, the existing tools can be divided into two main categories: user-friendly libraries designed for single-machine processing and large-scale distributed graph processing systems.

There is an abundance of single-machine libraries designed to analyze static social networks. All these tools usually come with an extensive library of standard network analysis algorithms for assessing network properties such as diameter, connectivity, centrality measures, clustering, and many others. The two most notable tools in this category are *IGraph*[115] and *NetworkX*[116]. Unfortunately, the internal graph representation adopted by these tools does not allow them to scale to large graphs. There are also single-machine tools such as *SNAP*[117], which are highly optimized and can scale to very large graphs, especially when used on computational hardware with a large amount of memory. However, the other downside of these tools is that their graph abstraction does not directly support parallel computation. As a result, if the user wants to use parallelization to speed up computation, he/she has to do it separately for each algorithm.

Systems such as *Pregel*[118], *PowerGraph*[119] and *GraphX*[120] can scale to extremely large graphs by distributing the computation over a cluster of machines. However, they provide only a handful of standard graph algorithms. Therefore, the scope of the analysis they allow the user to perform is very limited compared to the single-machine systems. The main reason for this limited support for network algorithms is the somewhat restrictive gather-apply-scatter (GAS) computation model adopted by all these tools. Although the GAS model is great for scaling to massive graphs in a distributed setting, it is only well suited for the implementation of some specific algorithms such as PageRank, HITS, or connected components. The usability of GAS for other types of algorithms still needs to be improved [121].

All the above-discussed tools have been designed to analyze static graphs and do not incorporate any notion of dynamic graphs. For the user to perform dynamic analysis on a series of snapshots of a network, it is up to the user to construct these snapshots beforehand. This approach is tedious and requires to re-implement the snapshotting logic over and over again. This approach is incredibly ineffective for analyzing large graphs when every re-computation of snapshots takes a long time.

To overcome these disadvantages, several frameworks oriented specifically towards time-evolving graphs have been proposed recently, such as *Chronos*[122], *Kineograph*[123], or *GraphTau*[124]. However, their primary focus is on the distributed setting and high-performance graph storage. These systems lack both in terms of flexibility and implementation of a sufficiently large collection of network mining algorithms.

5.3 GraphSlices

Since none of the aforementioned frameworks meet all of the requirements for the effective processing of dynamic graphs we listed at the beginning of this section, we decided to design and implement our own framework named *GraphSlices*. This section will first describe the design consideration that led to the resulting architecture described in Section 5.3.2. Then, we will describe the implementation of this system in the Scala programming language and provide an overview of the interface GraphSlices provides for manipulating dynamic graphs.

5.3.1 Design considerations

We started with a similar observation as the authors of SNAP, which was that the capacity of RAM memories has drastically increased in recent years. This increase went hand-in-hand with a significant drop in the prices of available memory modules. This means that machines with hundreds of GBs of RAM became available and affordable to a broader user base. For instance, the largest memory-optimized Spot Instances *m4.16xlarge* provided by the AWS (see Section 4.3.3), which operate with 256GB of memory can be rented at a price of only $0.7 per hour[1]. Given this abundance of RAM, it is possible to load and process even the most extensive network datasets entirely in memory. Keeping the computation on a single machine significantly reduces the system's complexity since we do not have to worry about synchronizing the computation over multiple machines.

Regarding the basic internal graph representation, we drew inspiration from GraphX, which uses a straightforward relational representation. GraphX stores the whole graph in two tables: one is reserved for the list of nodes (vertices), and the second for the list of edges. On top of this representation, GraphX exposes a functional interface that provides a collection of expressive computational primitives that allow users to perform graph manipulation and computation. GraphX internally performs a very effective implementation of the join operation between

[1]Source: https://aws.amazon.com/ec2/spot/pricing/. Accessed at 12 October 2022

the nodes table and the edges table to pass information between nodes during computation. The graph computation is implemented using Apache Spark[125], one of the major distributed data processing engines today. However, the downside of GraphX is that it is only limited to the GAS computational paradigm, which restricts the scope of operations exposed in the functional interface.

The simple relational graph representation of using only two tables for vertices and edges is very effective for implementing graph transformations since the in-memory join operation can be implemented very easily (e.g., using a hashmap). This representation is also well suited for implementing graph-dependent operations such as graph slicing. Additionally, by restricting ourselves to in-memory computation, we can create a functional graph interface similar to what GraphX uses. However, we can also extend it beyond the GAS computational model. For instance, we can extend our abstraction by more generic computational primitives such as effective vertex/edge iterators, which allow high-speed sequential processing and can be used to build a wider variety of graph algorithms.

The last thing to consider is the platform and the programming language we will use to implement GraphSlices. We aimed to select a functional language that is expressive enough to implement our designed graph interface (see Listing 5.2). Also, the language must allow us to implement high-performance code and provide fast collection data structures and iterators, which we will need to implement all the graph operations. The language must have sufficient support for concurrent computation – even though we are not interested in distributed computation, we still want to be able to leverage multiple cores on the same machine. The language that best fits all of these requirements is the Scala programming language, which runs on top of the Java Virtual Machine (JVM) platform.

5.3.2 Architecture

We designed GraphSlices as a generic framework for the analysis of dynamic networks. GraphSlices needed to provide sufficient functionality in 3 key areas to accomplish this goal. The first area was a rich and high-performance graph abstraction that could be used to manipulate graphs and implement various graph mining algorithms. The second key area was implementing a sufficient collection of graph mining algorithms. Finally, the third area was to allow the user to easily generate synthetic graphs or load existing graph data from various formats.

The architecture of GraphSlices depicted in Figure 5.2 reflects the essential elements laid out in the previous paragraph. At the core of GraphSlices is the *Graph* interface, which encapsulates a dynamic graph defined in Section 5.1 and provides methods to manipulate the graph. The *Graph* interface currently has two implementations, a serial one using plain Scala core collections, and a parallel one which leverages Scala's parallel collections[2]. Graph computation is performed by chaining multiple operations of the *Graph* interface, and every operation returns a new transformed version of the original graph. To see how this interface can be used to implement node degree calculation, see Listing 5.3.

[2]Overview of Scala's parallel collections: `https://docs.scala-lang.org/overviews/parallel-collections/overview.html`. Accessed at 21 November 2023

To effectively slice the graph based on the user-defined function and manipulate the expanded graph, we created a fast multi-level index for quick access to nodes and edges. The current *Graph* interface and multi-level index implementation allow the user to slice/expand the graph over more than one dimension. In Figure 5.1, we show an expansion using the time information attached to the edges; however, this is not necessarily the only dimension that might be interesting when analyzing a dynamic graph.

For instance, let us assume we want to study a dynamic multi-graph combining data from multiple individual online social networks such as Facebook, Twitter, and e-mail. Since many people use two or even three online social networks, these separate networks share some of the same nodes, which can be interconnected by different edges (based on the type of communication). In this example, the other dimension (besides time) that might be of interest to us when we analyze it could be, for instance, how nodes' roles evolve in these separate social networks and how they affect each other.

To perform such an analysis, we could first expand the graph over the type of graph dimension (Facebook, Twitter, e-mail) and then the time dimension using the time stamps on the edges. Then, we could simply run one of the node importance algorithms, such as PageRank, and see how the development of nodes' rank in different online social networks affects each other.

Every user-friendly graph mining library needs a set of rich methods for generating graphs and/or loading existing graphs from various formats. For the former, we have the module called *Generators*, which contains methods for generating standard classes of artificial graphs such as random, complete, or Barabasi-Albert graphs. Graph loading is handled in the *Builders* module, which provides methods to load existing graphs from common formats such as a list of edges, *CSV* files, or the popular *GraphML*[3] format.

The last module called *Algorithms* contains the implementation of a collection of common graph algorithms for link analysis (PageRank, EvolvingPageRank, HITS), graph structure analysis (MaxIndependentSet, Coloring) and clustering (Label Propagation). All the implemented algorithms can be used directly on the expanded graph, which makes working with dynamic graphs straightforward.

5.3.3 Implementation details

In GraphSlices, graph computation is expressed as a series of transformations that are sequentially applied to a directed graph. Each such transformation yields a new graph, which is why the core data structure in the GraphSlices framework is an immutable graph. The underlying immutable data structure stores both the directed adjacency structure of the graph and arbitrary data associated with either the edges or vertices (using the *Vertex* and edge *Edge* classes shown in Listing 5.1). The transformation operations provided by the *Graph* interface can transform both the data associated with the vertices/edges and the adjacency structure of the original graph.

[3]The official site for the GraphML format: `http://graphml.graphdrawing.org/`. Accessed at 21 November 2023

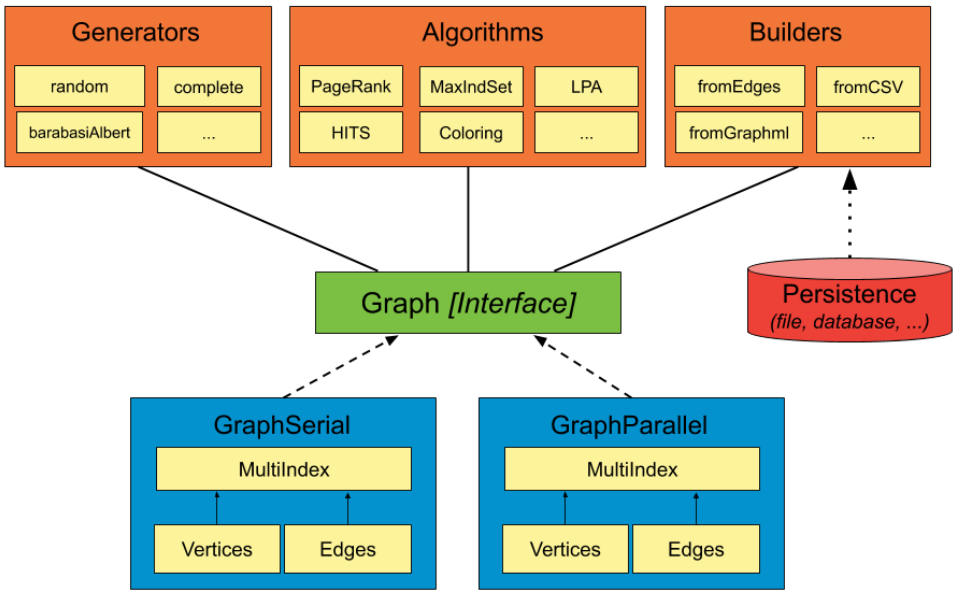


Figure 5.2: *Architecture of GraphSlices.* At the center lies the Graph interface, responsible for all graph manipulation operations. The two basic implementations this interface supports are the serial single-threaded implementation and the parallel multi-threaded implementation. The rest of the code is organized into three modules: Generators with various methods to construct synthetic graphs, Algorithms that contain a suite of techniques implemented for social network analysis, and Builders that load the graphs to/from memory.

.

Each vertex and edge is identified by a unique *Id*, which consists of a list (or sequence) of integer numbers. The reason why we need a list of numbers instead of just a single number is to be able to perform the expansion/reduction operations on temporal graphs using the *pushDimension* and *popDimension* methods described later in this chapter. Each graph expansion adds one new number to the *Id* of the vertex/edge to uniquely identify that vertex/edge across different slices. For instance, imagine a graph with a single vertex with $Id([1])$. Now, we perform an expansion operation over the time dimension and slice the graph using a yearly time window, so the resulting copies of the original vertex in our graph will get Ids of $Id([1, 2020]), Id([1, 2019]), ...$

Listing 5.1: Essential classes used to represent a graph in GraphSlices.

```
type Id = Seq[Long]

class Vertex[VD](val id: Id, val data: VD)

class Edge[ED](
    val id: Id, val srcId: Id, val dstId: Id, val data: ED
)
```

In Listing 5.2, we present the complete *Graph* interface, which exposes methods for manipulating the graph and performing graph computation. The methods *vertices* and *edges* provide a list view of the vertices and edges in the graph. Next, methods *vertexIndex()* and *edgeIndex()* provide access to the multi-dimensional

index used for keeping track of the components in the expanded graph. These are just convenience methods; the user does not have to use the indices directly. However, they are leveraged by the other methods in this interface.

In terms of graph manipulation, the *Graph* interface exposes methods *mapVertices* and *mapEdges*, which apply a user-defined function to vertices and edges and return a new version of the graph. To update the list of vertices/edges, the user can use the methods *updateVertices* and *updateEdges*, respectively. The method *outerJoinVertices* provides the ability to join the vertices with some external data using the vertex *Id* and then apply a user-defined function that can combine the existing vertex data with external data to create a new graph. The method *reverseEdges* simply changes the orientation of the edges in the graph. Lastly, the method *subgraph* takes as arguments two user-defined functions *edgePredicate* and *vertexPredicate*, which allow the user to filter edges and vertices to create a sub-graph of the original graph.

When performing graph computation, it is usually necessary to have direct access to both the edge and vertex data simultaneously. These methods require some sort of *join* operation between the list of edges and the list of vertices. In the case of GraphSlices, this join operation is handled by the individual implementations of the *Graph* interface (usually using a hash map). The user is not exposed to the underlying mechanism of joining the edge and vertex data; he/she is only concerned with the methods exposed by the *Graph* interface, which provides access to the necessary data. For instance, the method *triplets* provides a list view of all triplets in the graph in the format (source vertex, edge, destination vertex). The method *mapTriplets* allows the user to apply a user-defined function to each triplet to create a new graph.

Next, the *Graph* interface exposes the method *aggregateNeighbours*, which joins the vertex and edge data, applies the *mapFunc* on each edge and then performs a reduce by the destination vertex using the *reduceFunc* function. The *aggregateNeighbours* method allows the user to pass information between the vertices along the edges and then aggregate the information at the destination vertices. The *aggregateNeighbours* method is the essence of the GAS abstraction.

The last group of methods in the interface was designed to efficiently manipulate dynamic graphs using the expansion/reduction operations described in Section 5.1. The method *pushDimension* expands the current graph using the user-defined *mapToSubKey* function, which is applied to all the edges in the graph. The *mapToSubKey* function has access to all the data associated with a particular edge. It can, for instance, use the edge time stamps to map the edge into some time interval, e.g., yearly, monthly, or daily. GraphSlices does not provide direct access to the vertices during expansion since the copies of relevant vertices are created automatically for each slice (if that vertex is active, i.e., connected to some edge). The method *pushDimension* also provides an optional argument *keepAllNodes*, which ensures that a copy of each vertex is created in each slice, regardless of whether that vertex is connected to any edge. The *keepAllNodes=True* argument is handy when the user wants to fix the set of all vertices across slices.

The *popDimension* method performs the inverse operation, i.e., the reduction of the expanded graph using the least recent dimension added to the graph. To

perform reduction, the user needs to provide two functions *reduceFuncVertices* and *reduceFuncEdges* as arguments, determining how the data from different slices are aggregated. The most straightforward aggregation function can gather the information from each slice into a list, which the user can later explore. The *mapDimension* functions provide the ability to apply an arbitrary transformation function to the sub-graph in each slice of the most recent dimension.

We also want to point out that the current implementation of the *pushDimension* operation does not automatically add the meta-edges between the graph slices as formulated in Definition 5.1.3. However, these edges (and others) can be added to the expanded graph using the *updateEdges* method.

We want to stress that all three methods *popDimension*, *pushDimension*, and *mapDimension* create new versions of the original graph. However, the new graphs can still be accessed using the same *Graph* interface. This ability to provide a simple, transparent, and unified interface to manipulate dynamic graphs is the core idea behind GraphSlices. Many algorithms, such as PageRank or HITS, can be applied directly to the expanded graph; thus, get a dynamic version of those algorithms without any additional work. Furthermore, the current implementation allows users to use arbitrary dimensions when analyzing the graph. The only limitation of the analysis is the size of the available RAM.

Listing 5.2: The complete *Graph* interface as currently implemented in the GraphSlices framework. The methods in this interface are grouped into four categories: (1) methods for accessing graph data, (2) basic graph manipulation methods, (3) graph computation methods, and (4) methods for working with dynamic graphs.

```
class Graph[VD, ED] {
    // methods for accessing graph data
    def vertices(): Seq[Vertex[VD]]

    def edges(): Seq[Vertex[VD]]

    def vertexIndex(): MultiIndex[Vertex[VD]]

    def edgeIndex(): MultiIndex[Edge[ED]]

    // basic graph manipulation methods

    def mapVertices[VD2](map: Vertex[VD] => VD2): Graph[VD2, ED]

    def mapEdges[ED2](map: Edge[ED] => ED2): Graph[VD, ED2]

    def updateVertices[VD2](
        vertices: Seq[Vertex[VD2]]
    ): Graph[VD2, ED]

    def updateEdges[ED2](edges: Seq[Edge[ED2]]): Graph[VD, ED2]

    def outerJoinVertices[D, VD2](other: Seq[(Id, D)])(
        mapFunc: (Vertex[VD], Option[D]) => VD2
    ) : Graph[VD2, ED]

    def reverseEdges(): Graph[VD, ED]

    def subgraph(
```

```
        edgePredicate: EdgeTriplet[VD, ED] => Boolean,
        vertexPredicate: Vertex[VD] => Boolean
    ): Graph[VD, ED]

    // methods for performing graph computation
    def triplets(): Seq[EdgeTriplet[VD, ED]]

    def mapTriplets[ED2](
        mapFunc: EdgeTriplet[VD, ED] => ED2
    ): Graph[VD, ED2]

    def aggregateNeighbors[A](
        mapFunc: (EdgeContext[VD, ED, A]) => Seq[Message[A]],
        reduceFunc: (A, A) => A\
    ): Graph[A, ED]

    // methods for expanding/reducing graphs
    // and manipulation of expanded graphs
    def pushDimension[ED2](
        mapToSubKey: Edge[ED] => Seq[(Long, ED2)],
        keepAllNodes: Boolean = false
    ): Graph[VD, ED2]

    def popDimension[VD2, ED2](
        reduceFuncVertices: (Seq[(Long, Seq[Long], VD)]) => VD2,
        reduceFuncEdges: (Seq[(Long, Seq[Long], ED)]) => ED2
    ): Graph[VD2, ED2]

    def mapDimension[VD2, ED2](
        mapFunc: Graph[VD, ED] => Graph[VD2, ED2]
    ): Graph[VD2, ED2]
}
```

The GraphSlices framework currently provides two implementations of the *Graph* interface. The first serial implementation named *GraphSerial* is built using Scala's core collections only. The serial implementation relies mainly on the *Vector* data structure, which provides fast random access and updates, as well as very fast append and prepend operations[4]. GraphSlices also provides a parallel implementation called *GraphParallel* to leverage multi-core processors. This implementation uses Scala's parallel collections and in our case we rely primarily on the *ParVector*[5] trait/interface, which provides the ability to parallelize basic operations on *Vectors* such as map, reduce, and flatMap. Since parallelization comes with a particular amount of overhead compared to the standard collections, we recommend using it only for large graphs with at least millions of nodes/edges.

The complete source code for GraphSlices is included in the attachments of this thesis (Appendix B.2) and is also publicly available at `https://github.com/zviri/GraphSlices`.

[4] *Vectors* in Scala: `https://www.scala-lang.org/api/2.12.4/scala/collection/immutable/Vector.html`. Accessed at 10 October 2022

[5] Scala's parallel collections: `https://www.scala-lang.org/api/2.9.1/scala/collection/parallel/immutable/ParVector.html`. Accessed at 10 October 2022

5.3.4 Example usage

To showcase the expressiveness of the *Graph* interface introduced in the previous section, we will provide an example implementation of the PageRank algorithm in Listing 5.4. For clarity, we will describe this implementation line-by-line and explain the logic behind every transformation performed on the input graph. Then, in Listing 5.5 we will show how the implemented algorithm can be used on the simple dynamic graph shown in Figure 5.1.

Before explaining the primary implementation, we will first introduce two auxiliary methods required for implementing PageRank in Listing 5.3. These methods are *inDegreeWeighted* and out *outDegreeWeighted*, which, as their name suggests, compute the (weighted) in-degree and out-degree of nodes in the input graph. Both methods accept as input arguments (lines 2 and 15) an instance of a *Graph* with arbitrary vertex data (*VD*). However, the edges must be associated with only a single float (*double*) value. The edge value (or data) represents the edge weight, which can be used optionally to calculate weighted degrees in the graph. If the user only wants to calculate ordinary unweighted in/out-degrees, he/she can apply these functions to a graph with constant (1.0) edge weights.

To calculate the in-degree, we will use the *aggregateNeighbors* method provided by the *Graph* interface (line 4). This method requires two input arguments; the first is a *mapFunc* function, which accepts a parameter containing the edge context (including all its data) and returns a sequence of messages that are sent to other vertices. After all the messages have been sent, the function passed as the second argument *reduceFunc* is invoked on every vertex. This function takes as input two messages received by the given node and returns a single aggregate value. The *reduceFunc* method is applied on each message received by a specific node iteratively using an accumulator to obtain a single final value.

To implement the in-degree, we will use a *mapFunc* (line 5), which sends the weight associated with the current edge (*cts.edge.data*) as a message to its destination vertex. Then, the *reduceFunc* (line 6) will sum the weights received by each node. After the *aggregateNeighbors* method is finished, each node that is connected to at least one in-edge contains its in-degree. To fill in the zeroes for nodes with no in-edges, we will first create a list of tuples in format $(vertex\_id, vertex\_in\_degree)$ on line 7. Afterward, using the *outerJoinVertices*, which takes as an input argument a function (also called *mapFunc*) that is applied to each vertex in the graph. This *mapFunc* function provides access to the current vertex data and the new value provided by the join operation and returns the vertex's new value. In our case, we will replace the vertex value with the pre-computed in-degree if available. If the in-degree is unavailable, we are dealing with a vertex with 0 in-degree.

To calculate the out-degree of vertices (line 14), we will reuse the *inDegreeWeighted* method by passing it to the input graph with reversed edge orientation. To reverse the orientation of edges in the input graph, we will use the *reverseEdges* method in line 17.

Listing 5.3: Utility functions for calculating the in-degree and out-degree of nodes using GraphSlices.

```
1  def inDegreeWeighted[VD](
2      graph: Graph[VD, Double]
3  ): Graph[Double, Double] = {
4      val degrees = graph.aggregateNeighbors[Double](
5          ctx => Seq(ctx.msgToDst(ctx.edge.data)),
6          (a, b) => a + b
7      ).vertices.map(v => (v.id, v.data))
8
9      graph.outerJoinVertices(degrees) {
10       (v, d) => d.getOrElse(0.0)
11     }
12   }
13
14 def outDegreeWeighted[VD](
15     graph: Graph[VD, Double]
16 ): Graph[Double, Double] = {
17     inDegreeWeighted(graph.reverseEdges())
18 }
```

Using the methods defined in Listing 5.3, we can now proceed to the main implementation of the PageRank algorithm (PR). The implementation in this example will use the power iteration method. The *pagerank* method (line 1), in our case, accepts three arguments: (1) the input graph with weighted edges, (2) reset probability (or the damping factor) to use for calculating the PR, (3) the number of iterations to use.

The first step in the implementation is calculating the out-degree of vertices in the input graph, for which we will use the *outDegreeWeighted* method (line 6). Notice that if we want to calculate a weighted PR, we pass a graph with specific weights assigned to each edge. By default, we will assume the user passes a graph with constant (1.0) weights. Next, on lines $10 - 15$, we will initialize the graph variable *rankGraph*, which we will use to update the PRs iteratively. The *rankGraph* is initialized with vertices set to 1.0 (line 15) and edges set to $edge\_weight/source\_vertex\_degree$ (line 14).

Using the pre-initialized *rankGraph*, we will proceed to the power iteration implemented on lines 17 to 35. The rank updates work as follows. Using the *aggregateNeighbours* method on line 20, we will first send the current rank of each vertex over its outgoing edges (line 23). Then, using the *reduceFunc* we will sum the incoming ranks of every destination vertex on line 26[6]. Finally, we will update the ranks of the *rankGraph* using the *outerJoinVertices* method in combination with the *rankUpdates* computed in the previous step on line 29. In the map function on line 33, we will also apply the necessary damping factor to finish the rank updates correctly. These updates are repeated until we hit the predefined number of iterations passed in the *numIter* argument.

The typical PR implementation usually includes a convergence check, which ensures the rank updates are getting smaller, and once they fall below a predefined threshold ϵ, the iteration is stopped. We omitted the early stoppage to keep this example implementation as simple as possible.

[6]In Scala, the function $\_ + \_$ corresponds to $(a, b) => a + b$.

Listing 5.4: Implementation of the PageRank algorithm using GraphSlices.

```
def pagerank[VD](
    graph: Graph[VD, Double],
    resetProb: Double = 0.15,
    numIter: Int = 100
): Graph[Double, Double] = {
    val degrees = outDegreeWeighted(graph)
        .vertices
        .map(v => (v.id, v.data))

    var i = 0
    var rankGraph = graph.outerJoinVertices(degrees) {
      (vertex, degree) => degree.getOrElse(0.0)
    }.mapTriplets(
      triplet => triplet.edge.data / triplet.srcVertex.data
    ).mapVertices[Double](_ => 1.0)

    while (i < numIter) {
      i += 1

      val rankUpdates = rankGraph.aggregateNeighbors[Double](
        edgeCtx => Seq(
            edgeCtx.msgToDst(
                edgeCtx.srcVertex.data * edgeCtx.edge.data
            )
        ),
        _ + _
      )

      rankGraph = rankGraph.outerJoinVertices(
        rankUpdates.vertices.map(v => (v.id, v.data))
      ) {
        (vertex, prSum) => (
            resetProb + (1.0 - resetProb) * prSum.getOrElse(0.0)
        )
      }
    }
}
```

Finally, in Listing 5.5 we will showcase how the PR implementation can be used on the example graph from Figure 5.1. We will first load the dynamic graph structure, vertex, and edge attributes from a file using the *Builders* class on line 1. We will assume that the time data associated with the edges have the form of a list. Then, using the *pushDimension* method on line 3, which accepts as a first argument a *mapToSubKey* function applied to every edge in the graph. The *mapToSubKey* function returns a sequence of tuples of size 2, where the first element is the sub-key to which the current edge should be mapped, and the second is the data that should be assigned to the new edge. To calculate the PR for each time slice, we can apply the *pagerank* method we defined earlier to the expanded graph.

Listing 5.5: **An example for calculating the dynamic PageRank.** Using the example from Figure 5.1, we first load the (dynamic) graph structure, vertex, and edge attributes from a file. Afterward, assuming the time indicators have the form of a list, we create an expanded form of the network graph and compute the PageRank for every snapshot.

```
1 val graph = Builders.loadFromFile("...")
2
3 val slicedGraph = graph.pushDimension(
4     edge => edge.data.map(time => (time, 1.0))
5 )
6
7 val prGraph = Algorithms.pagerank(slicedGraph)
```

6. Experiments

6.1 Experiment 1: Insolvency process as a static social network

In this experiment, we will view the participants of the IPs as entities interconnected through various types of relationships. For example, these relationships can represent whether a specific debtor owes money to a creditor or whether the debtors are somehow related (e.g., married). Furthermore, our approach models insolvencies as a social network, where nodes represent individual participants and edges relationships between the participants. This novel approach can uncover new insights into the structure of debtors, particularly in the Czech Republic.

Depending on what relationships are important, there are different ways of constructing such a social network. However, the fundamental relationship in the network should be the "owes" relationship between a debtor and a creditor. Next, we are interested in the role of the insolvency administrator, which is a mediator between the creditors and debtors in the IPs. Lastly, we want to capture the direct relationships between the debtors who share the same domicile (these can be members of the same household).

We are particularly interested in obtaining new insights concerning the process of IPs and the structure of mutual relationships between the IP's participants. In particular, we are interested in answering the following questions:

1. Is it possible to build a social network from insolvency data?
2. What would the structure of the insolvency network be?
3. Is it possible to find groups of nodes with similar relationships among the data groups?

6.1.1 Dataset construction

For this experiment, we used the CID to construct the social network as follows. First, we used the debtors, creditors, and administrators to create nodes. Then, we included a directed edge from the node d to c if debtor d owes money to creditor c. The "owes" relationship in the CID is represented by the creditor having a receivable against the debtor. Similarly, we included a directed edge from node a to d if a administers the IP of d. Finally, we included bidirectional edges between all the debtors sharing the same address.

Given the large number of insolvencies, we decided to focus on a smaller subset of IPs only. Therefore, in this experiment, we only considered the IPs commenced between 2008 and 2014 from *Ústecký kraj*. The analyzed social network comprised almost 6000 nodes of 3 types (debtors, creditors, and administrators).

6.1.2 Results

We used Gephi [126] to construct, visualize, and analyze the insolvency network. To get a sense of the importance of individual nodes, we calculated the PageRank of each node. Finally, we used a community detection algorithm called the Method of Optimal Modularity [127] (MOOM) to partition nodes into groups based on their similarity in the network. We set the resolution parameter of MOOM controlling the granularity of the partitioning experimentally to 1.1. We show the resulting network in Figure 6.1.

We can see that the insolvency network is a dense and highly interconnected network where most nodes belong to one large connected component[1]. However, we can also notice that the network contains a small number of highly important nodes, most of which are large creditors like *GE Money Bank*, *CETELEM*, and *Česká spořitelna*. The network structure also shows that the creditors share most of their debtors with other creditors.

Using the MOOM algorithm, we discovered four large communities colored green, light blue, navy, and purple. Each of the found communities contains all three types of nodes: debtors, creditors, and administrators. The largest is the green community, which contains mainly large financial institutions like the three most important creditors mentioned in the previous paragraph. These creditors often participate in the same IPs. The second, light blue community, contains a larger number of less frequent creditors like *Telefonica CR* and *T-Mobile CR*.

At last, we want to bring attention to the highly interconnected clusters of nodes on the outer edges of the visualization. We denoted the largest clusters as $SA1$ and $SA2$. These groups contain debtors who all share the same address. However, based on their last names, the debtors in these clusters did not seem to be related. After a short investigation, we found that the shared addresses belong to the town councils of *Chomutov*, *Most*, and other cities. These debtors likely lost their homes and were assigned a domicile at their local town council.

6.1.3 Summary

In this experiment, we showed the applicability of social network analysis on data obtained from the IR. Furthermore, we managed to construct an insolvency social network that provided novel insights into the structure of debtors in the CR. We showed that the insolvency network is very dense and contains a small number of highly influential creditors that often participate in the same IPs. Still, after applying community detection analysis, we found two large yet distinct groups of IP participants with different characteristics. We also identified larger clusters of debtors who share their addresses and are not relatives but people who moved their domicile to the town council.

We only used a small part of the CID for this experiment, i.e., insolvencies from the *Ústecký* region. Therefore, in the following experiments, we will broaden the scope of our analysis to cover the entire CR.

[1] We removed several smaller disconnected components from the visualization because they were irrelevant.

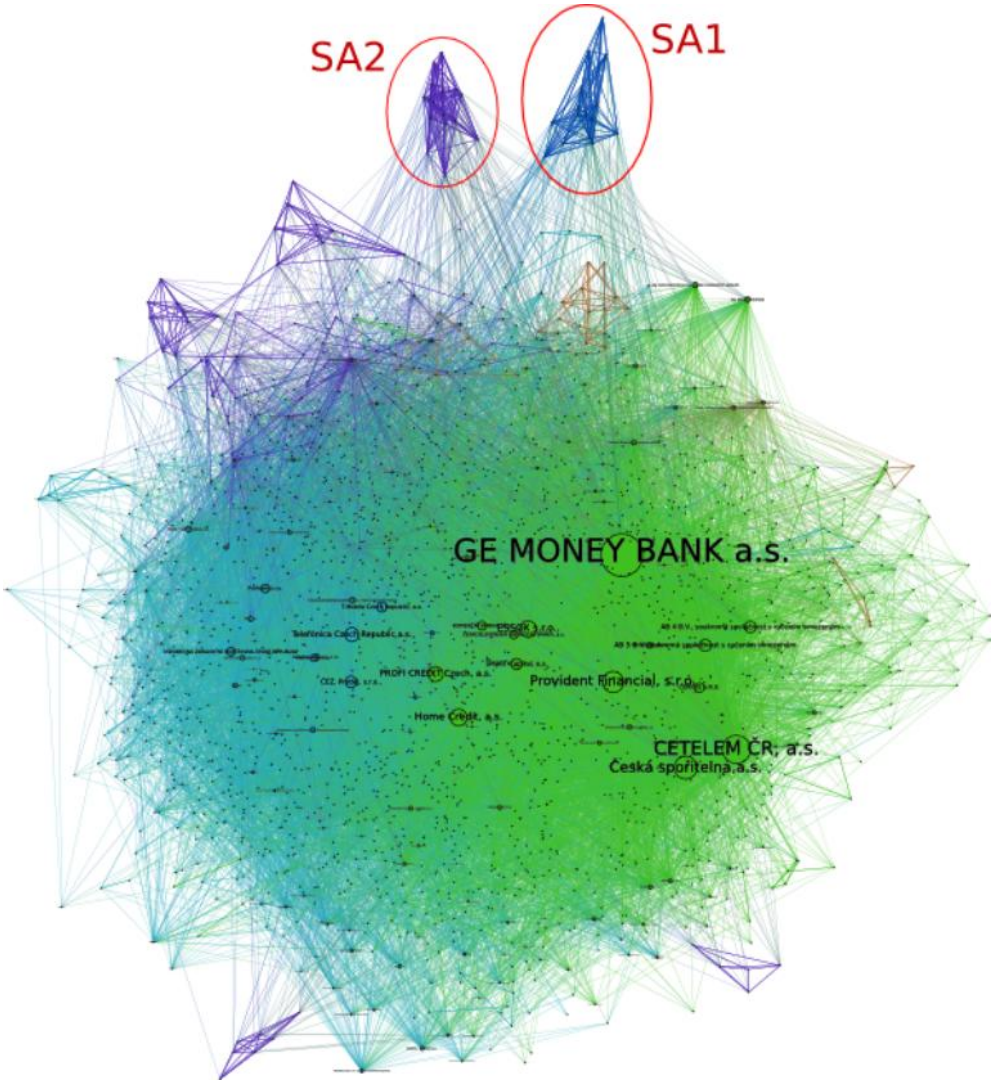


Figure 6.1: Visualization of the insolvency network constructed from insolvencies in the *Ústecky* region commenced between 2008 and 2014. The node size reflects the importance of individual nodes calculated using PageRank. The colors represent communities discovered by the MOOM algorithm.

.

6.2 Experiment 2: Insolvency process as a dynamic social network

In the previous experiment, we showed that subjects involved in insolvencies (debtors, administrators, and creditors) form a complex social network. This insolvency network is dense, and link analysis showed that it contains a small number of important nodes with different roles. This experiment aims to determine if and how the insolvency network evolves. We want to capture this evolution through the changes in the importance of nodes over time. We will be especially interested in the network's most important/influential nodes at any given time. Lastly, we will extend the studied network by an additional node type in the form of senates representing courts handling a particular IP.

Since almost every region in the Czech Republic has its regional court that exclusively handles the IPs commenced in the given region, we will also extend the scope of this experiment to every region (14 in total). Furthermore, we want to study the regions independently to capture potential differences in IPs.

Understanding the role of important nodes in the network and their evolution over time could provide helpful hints for future link prediction. That is why we will use association rule mining in the second part of the experiments to predict the emergence of future links in the insolvency network. These rules could help us predict, for example, what subjects are likely to participate in an IP together.

To summarize, in this experiment, we will be interested in answering the following questions:

1. How to identify prominent nodes in the insolvency social network?
2. Does the importance of nodes and the network structure change over time?
3. Can strong relationships found in the network predict the emergence of future links between the nodes?

6.2.1 Dataset construction

For this experiment, we selected all IPs from the CID commenced between 2008 and 2014. Then, for every region and every year, we created an individual network snapshot containing only insolvencies commenced in that year in the given region. As a result, we created $7 * 14 = 98$ different network snapshots. During the construction of the networks, we considered the following four types of nodes: creditors, debtors, administrators, and judicial senates. Lastly, we included edges between the nodes to capture the following relationships:

- a directed edge between debtor d and creditor c means that debtor d owes money to creditor c
- a directed edge between administrator a and debtor d indicates that administrator a manages the insolvency proceeding of debtor d
- a directed edge between senate s and debtor d shows that senate s handles the IP of debtor d.

6.2.2 Results

Social network analysis

Given the construction described in the previous section, we knew that some nodes in the network only contained incoming edges (e.g., creditors), and some nodes only contained outgoing edges (e.g., administrators). To capture the dynamics in the network over time, we wanted to calculate meaningful importance scores for both types of nodes. As a result, we used the HITS algorithm to calculate the authority and hub scores. Naturally, the creditors who did not contain

any outgoing edges represented the authorities in the network, and the administrators and senates formed, on the contrary, the network's hubs. For comparison, we also calculated the normalized degrees of every node.

Next, we used the *NetworkX* [116] Python software package to construct the individual networks and calculate all nodes' authority and hub scores. Then, to explore and visualize the networks, we used Gephi once more. Finally, we show the evolution of the authority scores from the region *Jihomoravský* between 2008 and 2014 in Figure 6.2. We can see that the number of IPs dramatically increased in that period, and the influence of the involved subjects (particularly the creditors) changed significantly over time. The number of subjects used to construct the network is shown in Table 6.1.

Table 6.1: The number of subjects involved in IPs in the *Jihomoravský* region between 2008 and in 2014 used to construct networks shown in Figure 6.2.

Year	**Debtors**	**Creditors** ***(the total number of all creditors if known)***	**Administrators**	**Senates**
2008	150	21	154	15
2009	371	21	226	19
2010	545	21	107	19
2011	1,057	21	133	30
2012	1,574	21 (*4,008*)	132	25
2013	1,927	21 (*4,113*)	188	34
2014	2,003	21 (*3,819*)	211	46

We can identify creditors whose initial dominance declined substantially over time, e.g., *General Health Insurance* (VZP), *Home Credit* lending company, or *Telefónica Czech Republic*. On the other hand, some creditors maintain their prominence over all seven years, e.g., *GE Money Bank*. We can, however, also notice a rapid rise in the influence of initially almost dormant creditors like *Provident Financial* (starting in 2011). This observation is in accordance with the authority scores computed for the *Jihomoravský* and *Karlovarský* regions, see Figure 6.3a. The network dynamics in other regions are very similar to these two regions and, therefore, will be omitted in this report.

In the case of administrators and senates, we can observe a different trend (Figure 6.3b,c). Initially, a small number of dominant (with respect to the number of IPs to be handled) nodes is present in the IR. However, with the growing number of insolvencies, their influence (measured as hub scores) tends to become more evenly distributed over time. An exception to this rule is the *Jihomoravský* region, especially in 2012 and 2014. In this case, a small group of nodes appears to yield higher hub scores. On the other hand, normalized node degrees did not reveal any meaningful trend in this respect, see Figure 6.3d.

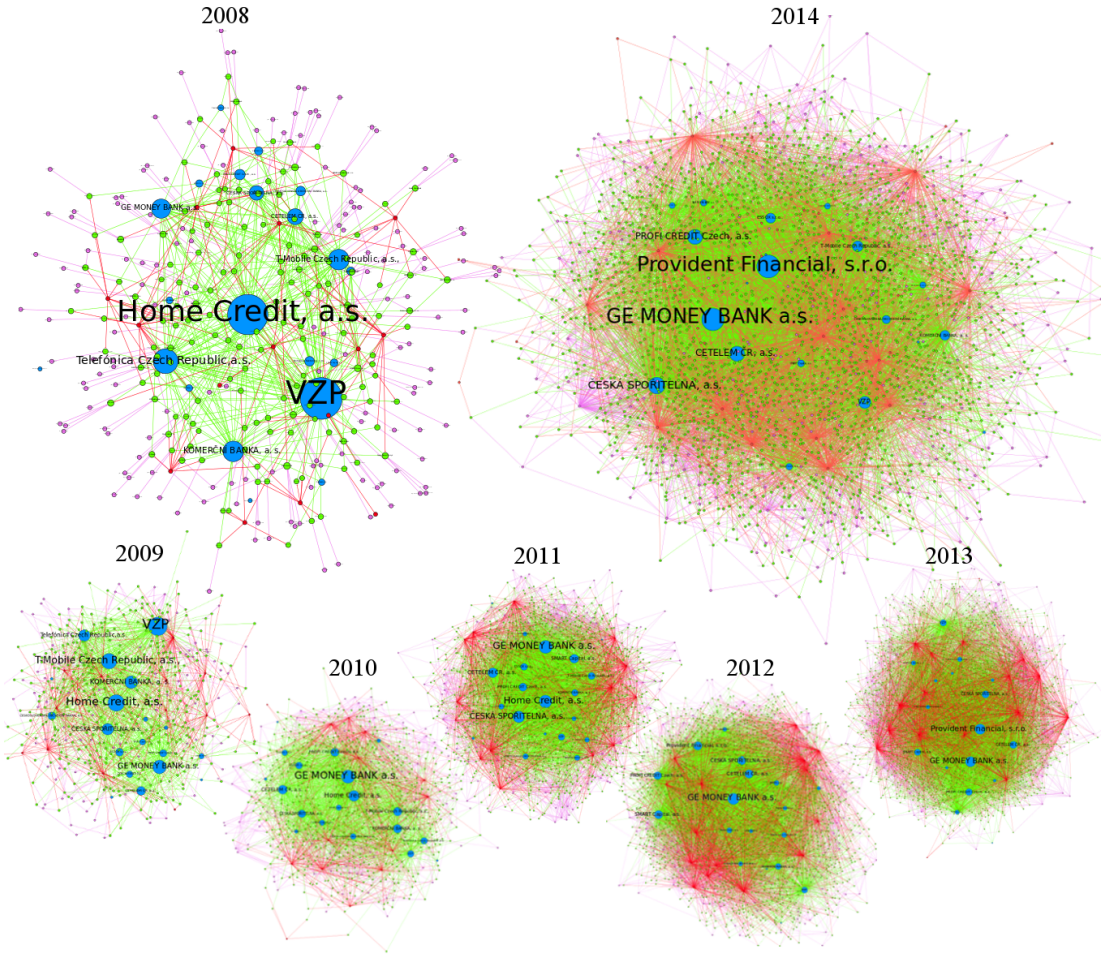


Figure 6.2: Authority scores of IP subjects from the *Jihomoravský* region between 2008 and in 2014. Creditors are marked blue, debtors green, administrators red and senates purple.

.

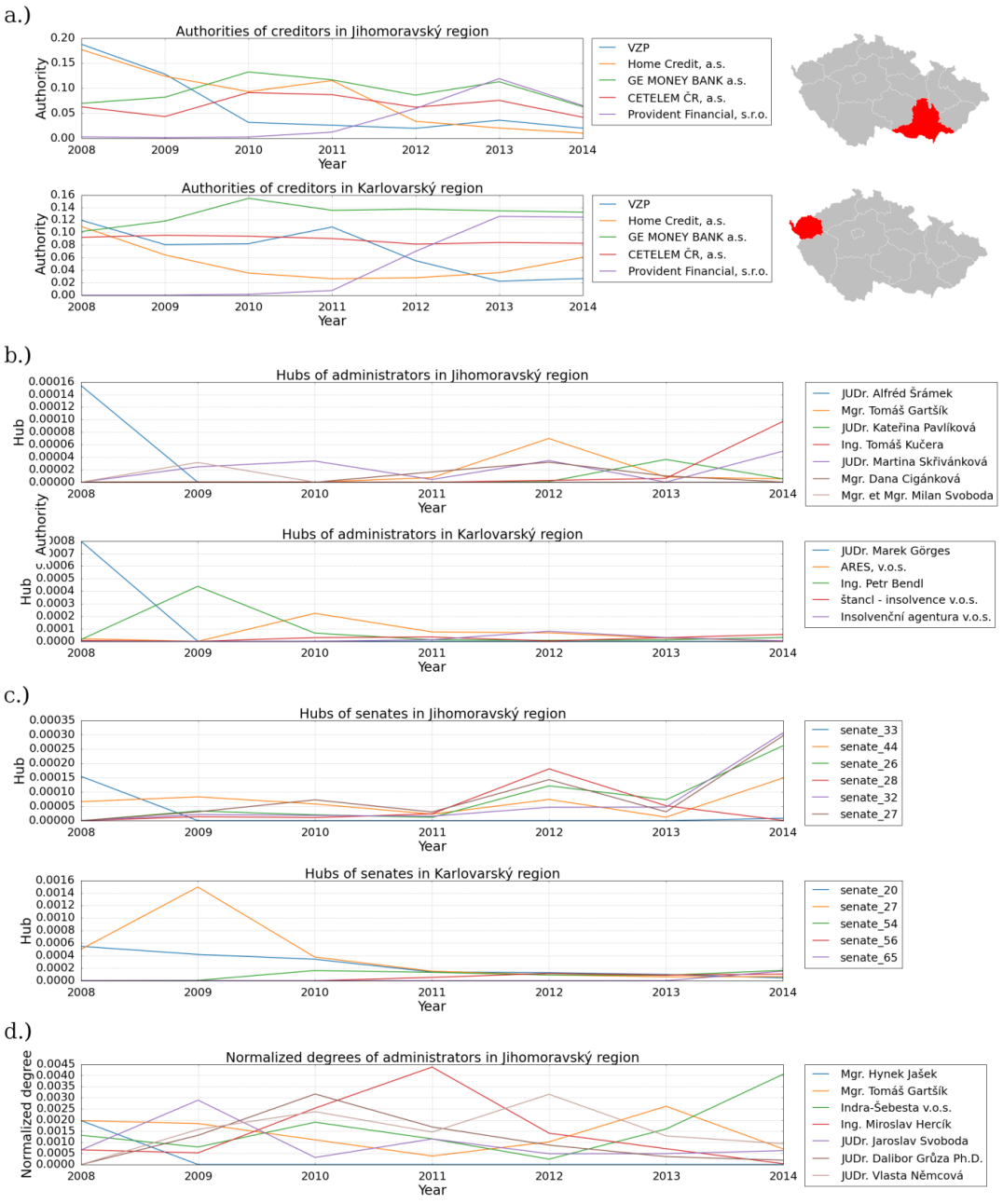


Figure 6.3: Importance evolution (authority scores, hub scores and normalized degrees) for the union of the top individuals over the considered period in the regions *Jihomoravský* and *Karlovarský*.

Mining association rules for link prediction

In the second part of this experiment, we used association rules mining with rules containing authority or hub nodes as items that could explain potentially emerging relationships in the network. These rules could help us predict what subjects will likely participate in an IP together. For this task, we used the open-source data mining toolkit *Weka* [128]. The itemset we used to mine rules contained between 400 and 600 different subjects (debtors, creditors, administrators, and senates), one item for each of the years 2008 to 2014, and two items representing the debtor's type: natural person (N) or legal entity (L). The transaction set contained approximately 9,000 transactions for the region *Jihomoravský* and 4,500 for the region *Karlovarský*. We created one transaction for one IP where at least one debtor, administrator, and senate participated.

Since our goal was to use association rules for link prediction, we were interested in very particular forms of rules. To ensure these rules would end up in the mined rules set, we had to set the minimum required support thresholds extremely low, for instance, 0.001, which corresponded to tens of transactions. As a result, this would lead to an exponential increase in the number of mined rules, which would cause the mining algorithms to use an excessive amount of memory and time for computation. For this reason, we adjusted the two main rule mining algorithms in Weka, Apriori, and FP-growth, to only mine rules in the forms we were interested in. These adjustments would lead to the pruning of the search space and the mined rules set. This strategy was effective and led to performance improvements in both algorithms, although the improvements for FP-growth were much more significant than for Apriori.

To demonstrate the performance improvements, when we were looking for rules matching the pattern *person_type, senate, year* $\Rightarrow$ *administrator*, both the Apriori and FP-growth algorithms returned the same rules for both regions within a few seconds. In the case of rules matching the pattern *creditor, person_type, senate, year* $\Rightarrow$ *creditor*, the FP-growth algorithm finished the search within a few seconds again, as looking for more specific rules decreased the number of paths to be searched for in the FP-tree. In the case of the Apriori algorithm, however, the system ran out of memory and could not generate any rules. The main reason Apriori remained less efficient is that the pruning of the search space in each step can only occur after all the candidates have been generated. Compared to FP-growth, we could implement the pruning procedure directly into the candidate generation function, which was much more memory efficient.

In both cases, we found several interesting rules with high confidence and lift scores even over 100. In the first set of rules, we found different administrators only on the right-hand side (RHS) of the rules for different years — see Table 6.2. This result means we did not find any strong association between the administrators, i.e., no administrators are likely to occur in IPs together. However, we identified several strong associations between specific senates (judges) and specific administrators. Based on the second set of rules involving creditors, we can conclude which creditors are likely to participate in future IPs together — see Table 6.3. For example, in the *Jihomoravský* region, legal entities owing money to *General Health Insurance* (VZP) seem to also owe money to the *Czech Social Se-*

curity Administration (Česká správa sociálního zabezpečení). In the *Karlovarský* region, instead, natural persons are affected by mounting amounts of debt arranged with several non-banking lending companies like *Provident Financial* or *Door Financial*.

Table 6.2: Examples of association rules of the form *person_type, senate, year* ⇒ *administrator* found for the regions *Jihomoravský* (senate codes starting with KSBR) and *Karlovarský* (senate codes starting with KSPL). Support corresponds to the number of transactions.

LHS	**RHS**	**Support**	**Confidence**	**Lift**
pt_N, sen_KSBR-24, y_2009	admin_Mgr. Vladimíra Zukalová	9	0.257	165.323
pt_N, sen_KSBR-24, y_2013	admin_Mgr. Tomáš Gartšík	11	0.066	4.361
pt_N, sen_KSPL-27, y_2014	admin_CITY TOWER, v.o.s.	11	0.080	3.266
pt_N, sen_KSPL-27, y_2014	admin_admin_Ing. Petr Bendl	12	0.087	3.186

Table 6.3: Examples of association rules of the form *creditor, person_type, senate, year* ⇒ *creditor* found for the regions *Jihomoravský* (senate codes starting with KSBR) and *Karlovarský* (senate codes starting with KSPL). Support corresponds to the number of transactions.

LHS	**RHS**	**Support**	**Confidence**	**Lift**
cred_VZP, pt_L, sen_KSBR-24, y_2011	cred_čssz	10	0.714	40.067
cred_VZP, pt_L, sen_KSBR-40, y_2012	cred_čssz	13	0.692	38.834
cred_VZP, pt_L, sen_KSBR-24, y_2012	cred_čssz	9	0.619	34.725
cred_bohemiafaktoring, pt_N, sen_KSPL-65, y_2014	cred_intrumjustitia	11	0.550	22.535
cred_gemoneybank, pt_N, sen_KSPL-65, y_2014	cred_intrumjustitia	15	0.172	7.064
cred_providentfinancial, pt_N, sen_KSPL-29, y_2013	cred_doorfinancial	10	0.179	6.097
cred_providentfinancial, pt_N, sen_KSPL-27, y_2014	cred_doorfinancial	10	0.175	5.990

6.2.3 Summary

In this experiment, we showed that the insolvency network is very dynamic in nature and undergoes significant change over time. This evolution can be effectively captured by analyzing the importance of individual nodes at different time frames. We have shown that the set of most influential nodes in the network changes as existing players become stagnant and new players enter insolvencies. Using the HITS algorithm, we could model nodes with different roles. Some nodes naturally became authorities, i.e., nodes with mostly incoming edges, such as creditors, and others became hubs, i.e., mainly nodes with outgoing edges, such as administrators or senates. We also determined that there are no significant differences between the Czech Republic's region regarding the evolution of their respective insolvency networks and that studying the entire insolvency network as a whole is appropriate.

In the second part of the experiments, we used rule mining to find associations that would capture the co-occurrence of subjects in the IPs. We found several

strong associations between certain senates and administrators and also associations among creditors who participate in the same IPs. The rules also showed that these associations change over time. As a result, association rule mining in the context of the insolvency network has proved applicable for link prediction.

6.3 Experiment 3: Understanding where debt originates

In the experiments so far, we focused solely on the nodes in the insolvency network and the relationships between them, i.e. purely structural information. The only information beyond the structure we used in the experiments was the time information about when individual relationships in the network were formed. However, as described in Section 4.3, a large amount of additional information about individual nodes or relationships is available in the ARs, and some of this data is already included in the CID. We will use this additional data to enrich our analysis and gain further insights into the process of IPs. In this experiment, we wanted to test if we could use the debt origins (see Section 4.3.6) from ARs to segment receivables based on the nature of their debt. Since debt origins only come in the form of natural text, we used basic NLP methods in combination with unsupervised learning to cluster the receivables based on their nature.

In this experiment, we will focus on answering the following questions:

1. Can unstructured data from documents be used to enrich the insolvency network and gain additional insights into IPs?
2. Can we deduce the circumstances that cause indebtedness, and how much does its overall structure vary across the Czech society?

6.3.1 Dataset construction

This experiment focused on the 1,200,000 ARs from the CID that were published in the IR between 2008 and 2016. However, given the nature of the analysis performed in this section, it was intractable for us to consider all ARs. As a result, we randomly selected 100,000 ARs with successfully extracted debt origins, representing approximately 8.3% of all ARs. The sampling process for selecting the ARs included stratification over the years the ARs were submitted and also over different regions, they were submitted. Thus the results obtained in this section are representative of the entire population o ARs.

6.3.2 Results

The debt origin is a unique text field that loosely describes the reason for the claimed receivable. For this experiment, we first transformed the extracted debt origin texts to TF-IDF vectors of size 5,000. Then, we used the vectors to train a SOFM [129] with 900 neurons organized into a 30 $\times$ 30 grid. Next, we

grouped the found weight vectors based on their similarities in possible reasons for debt. Finally, we estimated the appropriate number of these higher-level groups as the (local) maximum found by both the Dunn method [130] and the Silhouette coefficient [131], see Figure 6.4. As the (global) maximum equal to 2 tends to group distinct reasons for debt into larger clusters, the next viable option was 15, which we selected. The weights of SOFM neurons were thus grouped into 15 clusters employing Agglomerative Clustering [132]. We show the visualized SOFM map, including clustering, in Figure 6.5.

Figure 6.4: Evaluation of the (normalized) Dunn and Silhouette indicators. We set the appropriate number of clusters to 15, where both indicators reach a local maximum.

To interpret the revealed indebtedness structure, we selected a small set of representative keywords using the χ^2-statistics over all of the considered n-grams that were assigned to each cluster. Based on the results, we could also label individual clusters by the typical debt origin they represent. We show the seven most relevant clusters in Figure 6.6, including their absolute and relative size, assigned semantics, top creditors involved, representative keywords, and a brief description of the cluster (prototype).

The largest Cluster 3, which included 50% of all ARs, is characterized by credit card debts and loans. The ARs in this cluster were filed by various creditors, mainly financial institutions. The remaining 14 clusters are smaller and correspond to rather specific reasons for debt. For instance, Cluster 4 contains outstanding phone bills for *T-Mobile CR*. Although *T-Mobile CR* is the biggest operator and serves about 60% of the Czech population, these ARs come from the country's Eastern regions. Cluster 5 includes both outstanding electricity bills and seizures from western regions of the country. Typical creditors for these ARs are companies *Bohemia Faktoring* and *CEZ Prodej*. The debtors from Cluster 5 tend to suffer under a heavier load of ARs than is typical for other clusters. ARs grouped in Cluster 8 are specific and represent outstanding health insurance bills, mainly from two northwestern regions with high unemployment rates.

6.3.3 Summary

In this experiment, we showed that unstructured data (debt origins) obtained from documents could be used to deeper our understanding of the insolvency process in the CR. Furthermore, the methodology developed in this experiment allowed us to cluster ARs effectively based on the nature of the debt they represented. We could even assign meaningful semantic descriptions to individual clusters based on the prototype ARs we found and their most typical keywords.

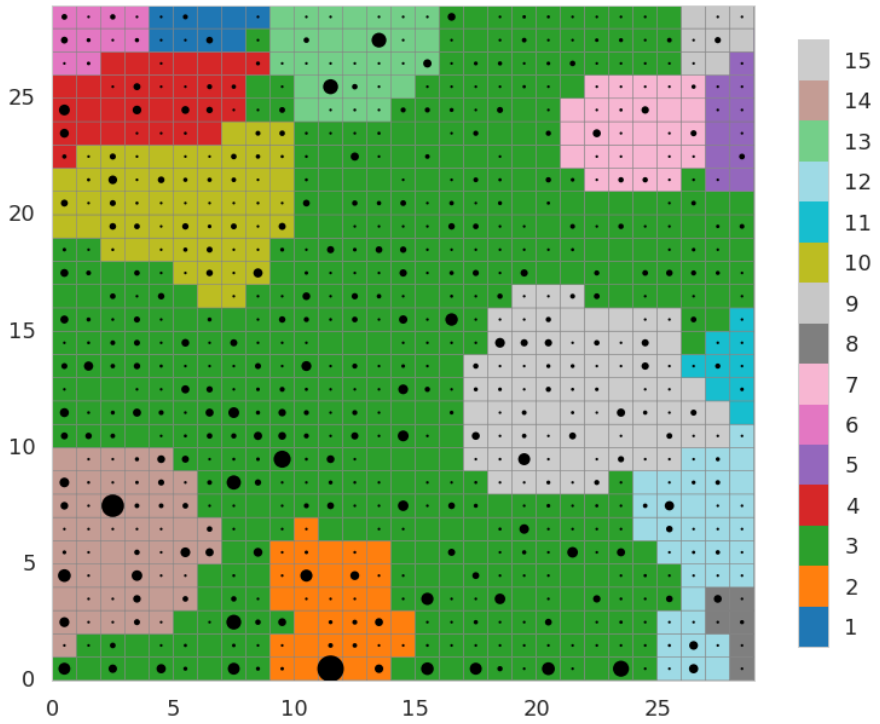


Figure 6.5: The SOFM map trained on debt origins and the 15 clusters obtained by using Agglomerative Clustering. The size of the circles reflects the number of ARs assigned to the given neuron.

The results confirmed that the indebtedness structure varies across the country and may include region-specific reasons for debt. Namely, we showed that the most significant chunk of ARs (50%) originated from financial products provided by banks and nonbanking lenders. However, we also found specific types of debts like phone bills or health insurance bills specific to certain regions.

Quite naturally, the information on the amount of the claimed debt would add a lot to the analysis of the underlying social structure of debtors. Therefore, analyzing the debt amounts will be the main focus of the experiment in the following section.

6.4 Experiment 4: Understanding the value of claimed debt

This experiment will continue exploring how additional data extracted from documents can be used to provide us with more insights into the process of IPs in the CR. In order to identify the main patterns of indebtedness across the Czech society, we will further aim at grouping together debtors who have similar creditors and also owe a similar amount of debt. For this purpose, we will use the claimed debt, i.e., receivable values extracted for ARs (see Section 4.3.5). We expect that the extracted value of the claimed debt will help us find meaningful clusters of debtors with clear semantic interpretations.

a.)

Cluster 4 Cluster 5 Cluster 8

b.)

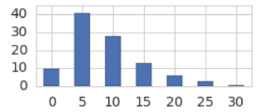
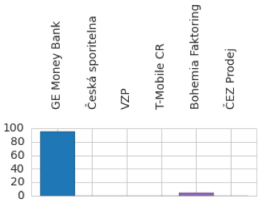
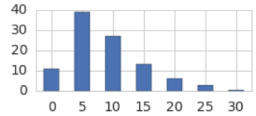
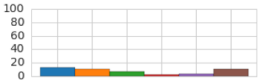
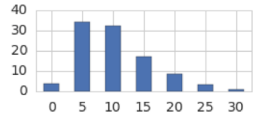
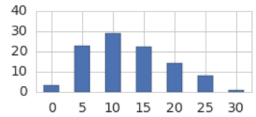
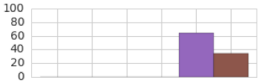
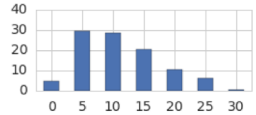
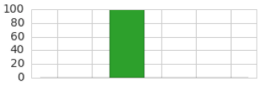
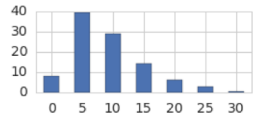
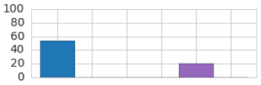
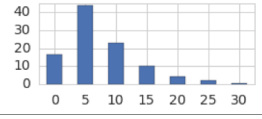
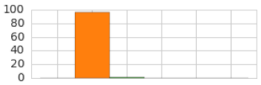
Cluster (Size)	Description	Keywords	Debts per person (in %)	Creditors (in %)
2 7 002 (7.0 %)	Credit card debts and loans	bank product service product flexi-account contract provision of banking from date credit credit card amount		
3 50 321 (50.3 %)	Credit card debts and loans	contractual issuing credit card revolving revolving credit unpaid		
4 3 110 (3.1 %)	Phone bills	contract fine default phone bill period		
5 1 043 (1.0 %)	Seizures and utility bills	seizure court order monetary fee electricity delivery		
8 1 606 (1.6 %)	Health insurance bills	insurance public health health insurance premium penalty		
14 9 765 (9.8 %)	Credit card debts and loans	credit card card loan usage of contract		
15 6 682 (6.7 %)	Credit card debts and loans	savings account loan card loan limit installment		

Figure 6.6: An overview of the clustering results, including the cluster identifiers and their brief (prototype) description by a set of keywords. The maps in (a) illustrate the distribution of cluster members over the country. Furthermore, in (b), we show the percentage of debtors with the corresponding number of debt obligations (or ARs) and the main creditors involved for each cluster.

For this experiment, we will use both structured data obtained from the IR and the extracted receivable values. In addition, we will be interested in answering the following questions:

1. How much money is claimed across the Czech society?
2. What amount of debt is usually claimed by one AR?
3. How much money and to whom do Czech debtors owe?

6.4.1 Dataset construction

To begin this analysis, we targeted all of the 1.5 million ARs in the CID that were published between 2008 and 2017. However, for this experiment, we also required complete information about which creditor submitted the given AR, which became available only in late 2012. Therefore, we decided to only focus on the top 18 creditors whose ARs we identified by creating a custom creditor classifier in Section 4.3.4. In the considered timeframe, these 18 creditors submitted approximately 300,610 ARs, from which we managed to extract the receivable value in 243,436 cases. Even though this subset only represents approximately 16% out of the total 1.5 million ARs, it provides us with complete information on receivable activities by the top 18 creditors between 2008 and 2017, which will be sufficient for this experiment.

6.4.2 Results

Total value of the claimed debt

The estimated debt value claimed against natural persons by the 18 largest creditors totals 28,622,258,855 CZK (USD 1,168,552,375). However, we determined the above amount based on 237,783 ARs out of the total 1.5 million ARs submitted in the considered timeframe. The overall debt value claimed across the Czech society is thus even larger. For the five biggest creditors, missing receivable values (extraction failure) amount to more than 5 billion CZK (see the estimated errors in Table 6.4). We estimated this error based on each creditor's average receivable value, which we then multiplied by the number of receivables for which the extraction status was not OK (see Section 4.3.5).

From Table 6.4, we can see that the biggest creditor in terms of the claimed debt against natural persons is *Česká spořitelna*, *Komerční banka* comes in as the second, and *GE Money Bank* as the close third. This finding was quite surprising at the time because, in Experiment 1, we showed that *GE Money Bank* was the most frequent creditor in terms of the number of debtors by far. However, its role has changed when we also consider the value of its receivables.

Similarly, when we switched our focus to debt claimed against company debtors by the top 18 creditors, we found 5,653 ARs with an estimated overall debt size of 9,705,055,947 CZK (USD 396,124,726). In this case, the missing receivable values accounted for 5,356,792,701 CZK in estimated error.

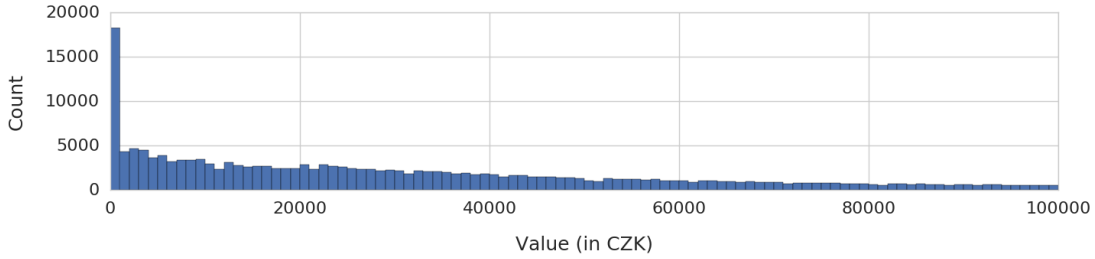
Clustering of insolvencies

For the analysis performed in this section, we used the same data extracted from 243,436 ARs submitted by the 18 most frequent creditors (with extraction status OK), i.e., 237 783 ARs for natural persons and 5,653 ARs for companies. We show the histogram of the values from these receivables in Figure 6.7a. The spike on the left-hand side of the graph indicates that many ARs claim less than 1,000 CZK, i.e., they represent just minor debts.

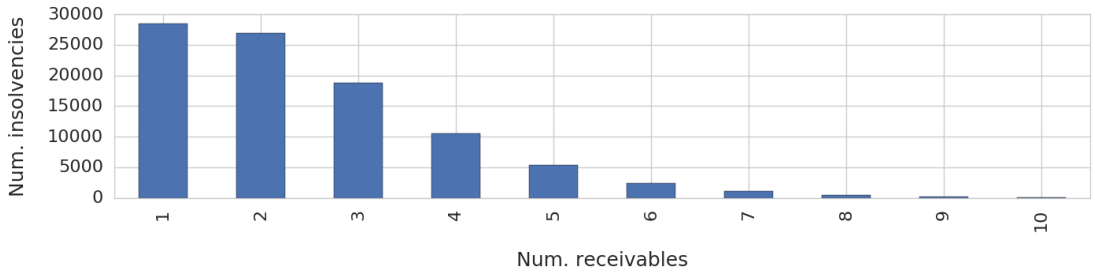
Table 6.4: The estimated amount of debt claimed by the five biggest creditors against natural persons.

Creditor	**Claimed Debt**	**Estimated Error**
Česká spořitelna	8,489.75 mil. CZK	1,815.81 mil. CZK
Komerční banka	4,006.79 mil. CZK	1,286.55 mil. CZK
GE Money Bank	3,457.88 mil. CZK	675.76 mil. CZK
Profi Credit	2,320.74 mil. CZK	986.45 mil. CZK
Raiffeisen Bank	2,142.93 mil. CZK	352.81 mil. CZK

The number of ARs decreases naturally with the increasing amount of the claimed debt. The histogram of the number of ARs per debtor, on the other hand, illustrates that most debtors have already accumulated several sources for debt (ARs), see Figure 6.7b.



(a) Receivables' values histogram.



(b) Histogram of the numbers of receivables per debtor.

Figure 6.7: Summary statistics of the considered 243,436 ARs.

To uncover the structure underlying the debts, we first grouped the debtors according to their creditors and the amount of debt they claimed. Before the actual clustering, we preprocessed the data related to the claimed debt as follows:

1. we aggregated the receivables from the same insolvency,
2. we grouped the overall claimed debt for each insolvency into nine intervals based on the following boundary values: 10,000; 20,000; 50,000; 100,000; 200,000; 500,000; 1,000,000; 5,000,000; and 10,000,000.

Afterward, we transformed the above groups into feature vectors for each debtor/insolvency by one hot encoding, i.e., we indicated the respective debtor's total debt amount by setting the feature corresponding to the given interval to

1, and setting the remaining features to 0. Insolvencies with an overall claimed debt larger than 10,000,000 were removed as outliers. Further, we extended the set of features by including one feature for each of the 18 creditors considered in the dataset. We set these features to the amounts claimed by the creditors against the given debtor and then normalized them, to sum up to 1. Overall, the new dataset covered 93 626 insolvencies, each characterized by 27 features.

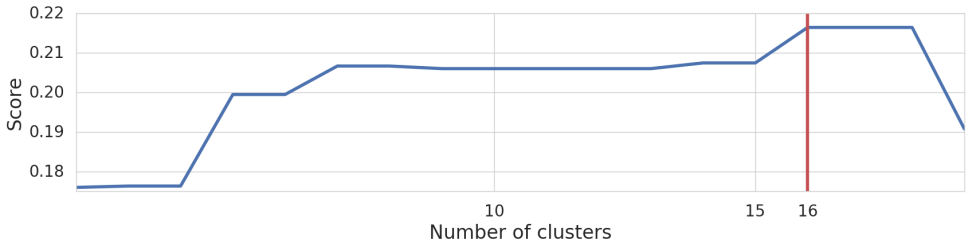


Figure 6.8: Cluster validity evaluation using the Dunn index: the appropriate number of clusters was set to 16 where the Dunn index reaches its maximum.

On this new dataset, we trained a self-organizing feature map (SOFM) with 15×15 neurons. Already within the trained SOFM, we noticed a fine-grained structure with several well-separated groups of insolvencies. To make these groups explicit, we further clustered the SOFM map using agglomerative clustering to make the semantic assignment to individual clusters easier. The cluster semantics was crucial for interpreting the uncovered debt structure. We estimated the appropriate number of clusters as the maximum (16) of the Dunn index, see Figure 6.8. We show the resulting SOFM with the labeled clusters in Figure 6.9.

The analysis of the individual clusters revealed several intriguing relationships. Figure 6.10, for example, depicts Cluster 2 of medium-sized debtors with an overall debt ranging between 100,000 CZK and 200,000 CZK. Based on the creditor frequency histogram shown in Figure 6.10a, these debts mainly include receivables from banks and non-bank lending companies. To investigate this observation further, we collected the total debts claimed by these two groups of creditors in the individual insolvencies from Cluster 2, normalized the values to sum up to 1, and visualized the result using a scatter plot. We show the scatter plot in Figure 6.10b. We can see that the debtors from Cluster 2 tend to owe money to both banks and non-bank lending companies and often to no one else. This picture might thus indicate the beginning of a debt spiral when the debtor first borrows money from a bank and then tries to compensate for his inability to repay his debt by borrowing money from less credible creditors.

Cluster 16, on the other hand, represents debtors with large overall debts ranging from 1 to 2 million CZK. In this cluster, the receivables are claimed mainly by the banks, see Figure 6.10a: *Česká spořitelna*, *Raiffeisenbank*, *Komerční banka*, and *GE Money Bank*. The scatter plot from Figure 6.10c further shows that most debt in this cluster originates from bank receivables, while at the same time, no or just a small amount of debt is owed to non-bank lending companies.

Finally, we also found interesting age distribution patterns for different clusters. Figure 6.11, e.g., illustrates that Cluster 3 contains younger debtors compared to Cluster 2. Although the debtors from Cluster 3 bear the same overall

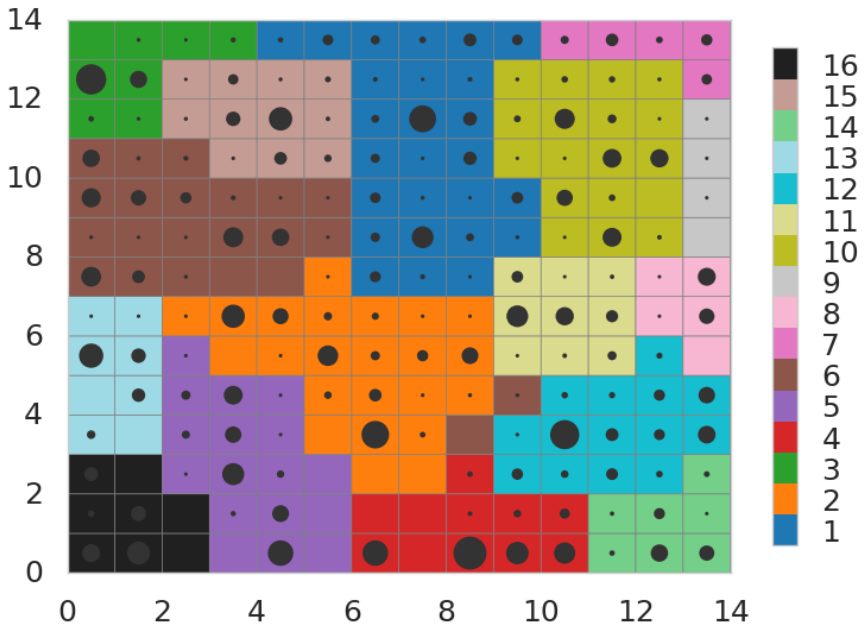


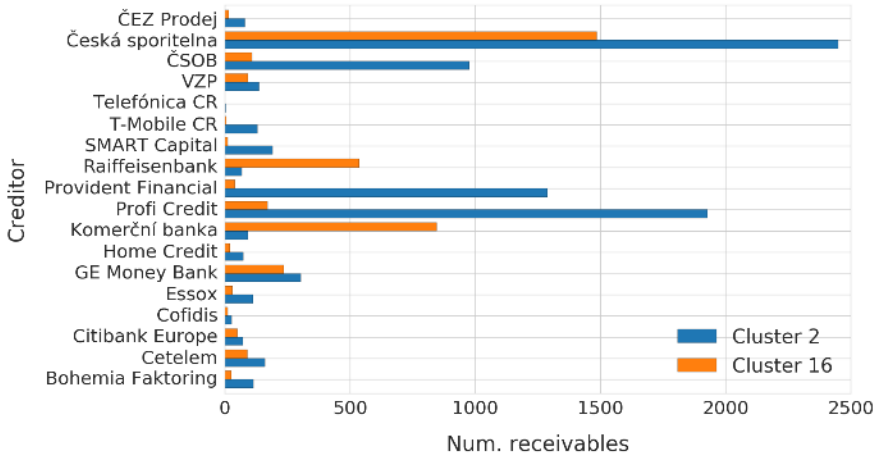
Figure 6.9: The trained SOFM map clustered by means of agglomerative clustering. The size of the circles reflects the number of ARs assigned to the given neuron.

debt as those from Cluster 2, most of the receivables are claimed by a single creditor — *Provident Financial*. Younger people likely represent one of the main target groups of Provident Financial. No other creditor from our dataset had shown a similar tendency.

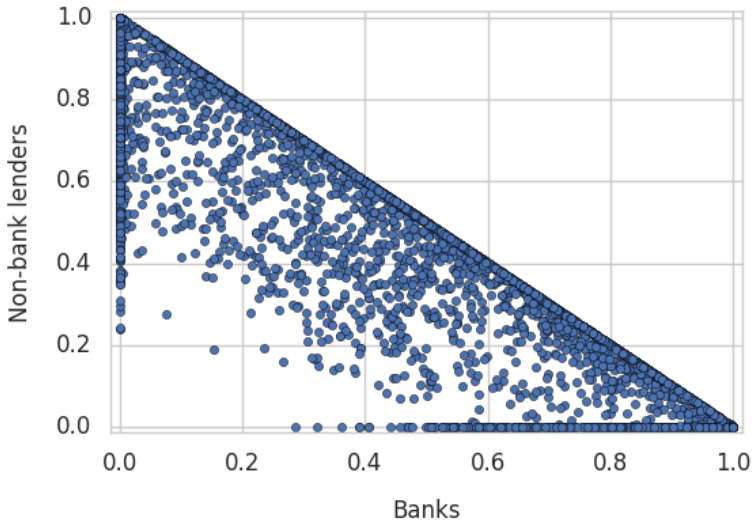
6.4.3 Summary

The values extracted from receivables in Section 4.3.5 allowed us to assess the amount of debt claimed over the Czech society and the overall debt owed to significant creditors. This analysis confirmed, e.g., that the most frequent creditors and the creditors with the largest debt do not have to be the same ones. The following clustering of individual insolvencies revealed various patterns common for the indebtedness structure and evolution. We identified clusters of debtors with different characteristics in both the size of their debt and the way they accumulated debt, eventually leading to bankruptcy.

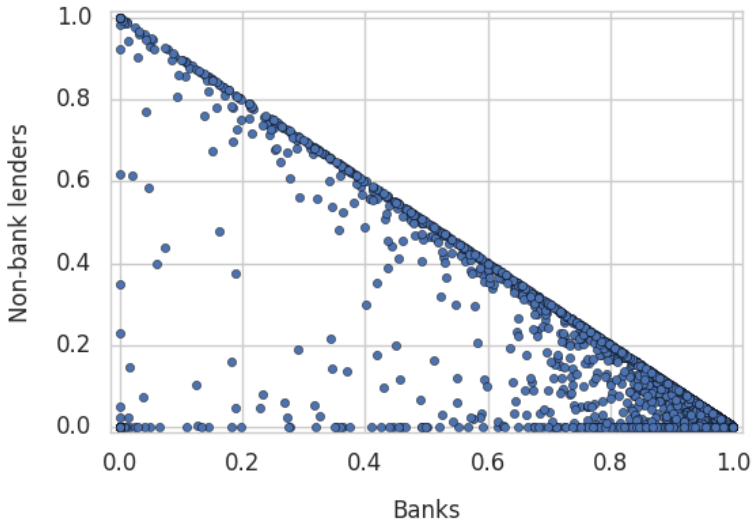
The knowledge of the claimed amount of debt also contributes to understanding the role and mutual relationship of the individual subjects participating in the IPs – the debtors, the creditors, the administrators, and the judicial senates. As we have already shown in the previous experiments, all these subjects form a complex social network of densely interconnected nodes. If we could enhance this social network by the knowledge of the claimed debts, such a model would offer a much clearer insight into the money flow within the network and the true role of its prominent nodes.



(a) Histogram of creditors for clusters 2 and 16.



(b) Scatter plot for Cluster 2



(c) Scatter plot for Cluster 16

Figure 6.10: Characteristics of selected clusters obtained using Agglomerative Clustering.

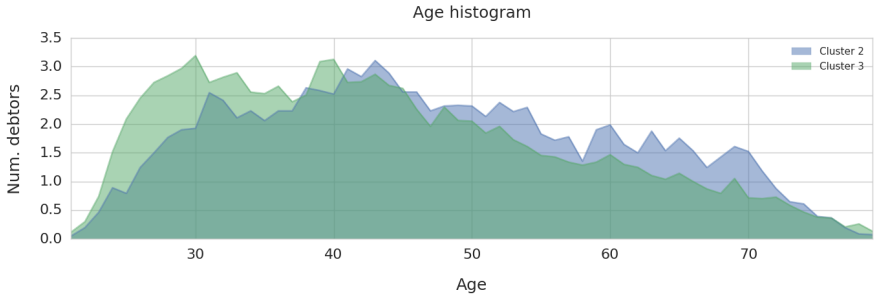


Figure 6.11: Age distribution comparison of Cluster 2 and 3.

6.5 Experiment 5: Assessing the future impact of subjects involved in insolvencies

In the previous experiments, we first showed that the subjects participating in IPs, such as debtors, creditors, and administrators, form a social network of densely interconnected nodes that evolves significantly over time. Typically, we studied the insolvency social network on a regional level. Then, we showed that the primary data extracted from the IR could be significantly enriched by data extracted from the documents, such as receivable values or debt origins. This additional data can be used to enrich the insolvency social network in the form of metadata attached to individual edges or nodes and provide a semantic layer on top of the purely structural data we have used to construct the network so far. In this experiment, we want to combine the approaches from the previous experiments by studying the largest insolvency network yet (300 000 IPs from all regions) and leveraging all the additional metadata that we already showed provides crucial insights into the process of IPs.

The central goal of this experiment is to build a reliable model for assessing the future impact of the subjects involved in IPs. Despite a large amount of social network data available for experimenting, this task remains challenging because data from real-life networks tend to be extremely skewed. Most of the subjects (nodes) do not exhibit much influence over time, yet there are (relatively few) players who significantly change their impact and affect others in the process. Furthermore, these players and their essential role in the entire process are not known in advance. Besides capturing the evolution trends, our goal was also to keep the model simple and interpretable, i.e., we wanted to identify impactful individuals and which inputs are essential for predicting future importance.

In the considered insolvency social network, we will once again model the subjects involved in IPs as nodes and their mutual relationships as edges. We will use artificial neural networks to capture the underlying evolution trends. Further, we will use sensitivity analysis to identify the most significant input features and omit the irrelevant ones to add to the final model's explainability. The rest of this section is organized as follows. We will first outline an original methodology to assess the nodes' impact in evolving social networks. Then, we will devote the second part to the analysis of the obtained results and their discussion. Finally, in the last section, we will summarize the achieved results.

6.5.1 Proposed methodology

In this section, we will outline a generic methodology for analyzing the evolution of individual nodes' impact in a social network. This methodology addresses various issues during the process, including data extraction from dynamic social networks or the pitfalls of using machine learning techniques on network data which are highly skewed terms of node degree distribution. In dynamic social networks, typically, a small number of nodes develop a vast number of ties over time, while most entities retain only a few of them. The entire methodology consists of the following five stages:

1. *Construct a relevant social network:* this phase integrates data extraction and preparation, network slicing for the considered time intervals, and significance assessment for individual nodes.
2. *Construct a dataset:* includes data extraction and cleaning, followed by the construction of other features.
3. *Preprocess the dataset:* consists of the dataset re-balancing, re-weighting, and oversampling with jitter.
4. *Build the model:* comprises model selection and training, iterative model pruning, re-training, and testing.
5. *Evaluate:* provides the evaluation of the obtained results and the interpretation of the found knowledge.

Below, we discuss the individual stages in more detail and apply them to the data from the CID.

Constructing a relevant social network

For this experiment, we used approx. 300,000 IPs from the CID commenced between 2008 and 2019, i.e., we covered a 12-year history of the IR. This data covers the IPs of about 188,000 individual debtors (a debtor can be involved in multiple insolvencies) and 900 administrators appointed by the court. Furthermore, approximately 170,000 unique creditors participated in the selected set of IPs. However, 118,000 (69%) creditors only occurred in a single IP. Since we wanted to study how the importance of nodes (esp. creditors) evolves, these low-frequency creditors were irrelevant for this experiment. For this reason, we only included the 1,000 most frequent creditors in the final network. These creditors occurred in at least 100 different IPs in the considered time period. Despite this selection process, the resulting insolvency network still included the data of more than 81% of all IPs commenced between 2008 and 2019, and it was the largest social network we studied in this work yet.

Based on the extracted data, we built a social network that mimics the expected financial flow among IPs participants. The debtors transfer money to the administrators managing their IPs through monthly installments or selling

their assets (e.g., a house). Afterward, the administrators distribute the collected means to the creditors. In the constructed network, we included directed edges between the respective debtors and their administrators and between the administrators and the involved creditors. We did not include any other edges in the final network. Additionally, we associated temporal information with each edge that reflects the period (start date and end date) when the particular relationship (e.g., IP administration) occurred. Given the possibility of repeated IPs by a single debtor, multiple edges between the same nodes, but in different time periods, were allowed. The constructed network effectively formed a multigraph.

We used PageRank (PR) to capture the individual nodes' significance. When calculating the PR for a specific time frame, we considered only the subset of edges relevant for the given period (we denote this step as slicing). After experimenting with different time granularities, we used one slice of the original network for each year between 2008 and 2019, i.e., 12 slices. Consequently, we also obtained 12 PR values for each node in the network. We employed the library GraphSlices we developed earlier (see Section 5.3) to perform slicing and PR calculation. We could load the entire multigraph representation of the explored network into GraphSlices, use the time-related information associated with the edges to slice the network dynamically with any given granularity and calculate PR scores in parallel for each slice.

Constructing a dataset

We approach the evolution of the node's impact as a regression task. Based on the past significance of nodes, we want to estimate their future influence. In principle, PR values reflect the network's nodes' importance (or prestige). The PR values lie between 0 and 1, and for all nodes in the network, they sum up to 1. Therefore, we can construct several training samples for each node using a rolling window function. For a rolling window of size n, we use past $n-1$ PR values to predict the node's n-th PR value. Given the 12-year history captured in our dataset and yearly slicing, we can create $12–n+1$ samples for each node in the network. This study considered rolling windows of sizes 3, 4, and 5 that produced training sets of about 1.85M (million), 1.65M, and 1.47M samples, respectively. We refer to them as $Dataset_i$, with i indicating the window size.

As shown in Figure 6.12, the PR distributions in the generated datasets were significantly skewed. Most PR values remained minimal in the studied insolvency network of roughly 189,000 nodes. In particular, the debtor nodes did not have any incoming edges, and the number of their connections did not change much over time. Similarly, the impact of other nodes with fewer connections showed a decreasing trend over time. On the other hand, the network grew significantly year by year (see Figure 6.13), and a few nodes developed a PR several times larger than the others (see Figure 6.12). Primarily, these nodes can reveal intrinsic patterns crucial for the evolution of the network.

The steady nodes with PRs that remain small over time were irrelevant to the prediction task and posed a potential problem for ML algorithms we wanted to employ, e.g., those that utilize SGD. For this reason, we eliminated the steady nodes from the considered dataset. In this step, we removed 99.75% of the least

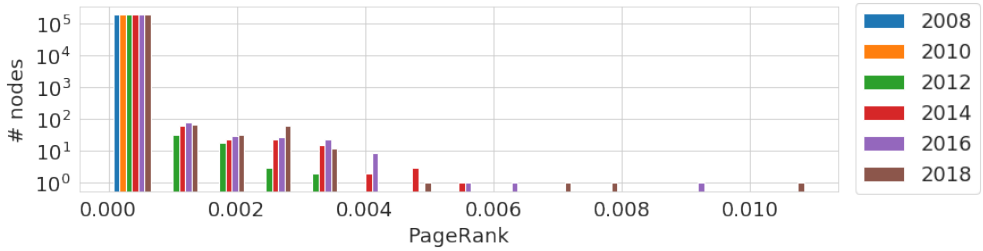


Figure 6.12: PageRank distribution in the insolvency network across years.

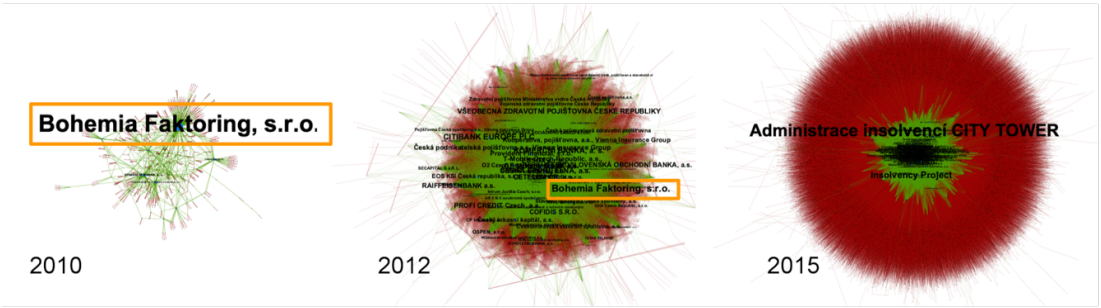


Figure 6.13: Snapshots of the insolvency network in different years. The size of the nodes correspond to their PageRank and different nodes are visualized as follows: debtors (red), creditors (green) and administrators (blue). For illustrative purposes the role of the creditor *Bohemia Faktoring* (*BF*) is highlighted (orange) in the first two snapshots.

relevant steady nodes, mostly debtors. As a result, the number of samples shrank to 4,175 for $Dataset_3$, 4,004 for $Dataset_4$, and 3,887 for $Dataset_5$. The final dataset retained node samples for all 100 creditors included in the network and about 600 administrators (out of the total 900 appointed by the court). Next, we rescaled their PR feature values, x, into the (0,1) interval (6.1) and transformed them using a sigmoid function (6.2):

$$rescale(x) = \frac{x - min(x)}{max(x) - min(x)} - 1 \tag{6.1}$$

$$transform(x) = \frac{1}{1 + e^{-rescale(x)}} \tag{6.2}$$

So far, the samples consisted of 2–4 features (depending on the chosen window size) corresponding to the transformed historical PRs and their target value. We have omitted the patterns generated by debtor nodes and smaller administrator nodes behaving similarly. The rest of this section will focus on building a model predicting the remaining nodes' future development.

To facilitate the application of more complex models, like neural networks, we have also constructed additional features beneficial for the node's future impact assessment. These features included the *evolution trends*, the ratio of *shared insolvency participants*, and the *debt origin*. Various external factors, such as economic growth, policy alteration, and amendments to the Insolvency Act, shape the trends in the development of insolvencies. As these trends may vary for different types of entities, such as administrators and creditors, we aggregated

yearly average PR statistics for each node type in the network. In addition, we used the identical size of the rolling window and preprocessing steps (6.1, 6.2) like the original PR data, which resulted in additional 2–4 features.

Shared insolvency participants In Experiment 2, we showed that the influence of nodes in the insolvency network could change considerably over time. A significant development shift can also be noticed in Figure 6.13, just over five years. Most observed creditors are financial institutions, such as banks, non-banking lenders, or insurance companies. Naturally, they might form groups providing similar financial products targeting similar customers. For creditors from such groups, it becomes very likely to share their debtors. When predicting future development for a given creditor, the information about shared debtors with other creditors can be beneficial. This overlap also occurs (yet less frequently) among administrators since multiple administrators can participate in the same insolvencies, e.g., in the case of corporate IPs.

For constructing this feature set, we considered the 100 most frequent creditors and 100 most frequent administrators and determined their ratios of mutually shared debtors. The new features corresponded to the percentage of debtors shared between entities e_i and e_j:

$$shared\_debtors(e_i, e_j) = \frac{|debtors(e_i) \cap debtors(e_j)|}{|debtors(e_i) \cup debtors(e_j)|} * 100 \tag{6.3}$$

We calculated the percentage of shared debtors separately for each year and each pair of creditors and administrators, respectively. Then, using the same rolling window function as before, we extended the data by additional 200–400 features depending on the window size.

Debt origin To further enrich the dataset, we wanted to include features that would differentiate between different types of debt, e.g., loans, utility bills, and others. We already showed in Experiment 3 that these categories could be obtained by clustering the debt origins extracted from the individual ARs. Therefore, we adopted a similar methodology for this experiment and applied it to approx. 2.5M ARs related to the 300,000 IPs in consideration. We used the obtained clusters to form logical categories to construct features that could capture the node's involvement with different types of debt.

Using the CID we obtained about 2M (80%) of debt origins from the overall 2.5M of ARs. Next, we removed all stop-words and unwanted numeric tokens, such as company IDs or account numbers, and generated word-level n-grams ($n \in 1, 2, 3$). Finally, we removed n-grams that occurred in more than 20% of debt origins or less than 100. The remaining n-grams built a dictionary of size 42,078 we used to generate a TF-IDF weighted bag-of-n-grams representation of all 2M debt origins. The following k-means clustering (k=15) revealed one large cluster (Cluster 1) with most of the ARs (63.78%) and a smaller Cluster 13 that contained 9.97% of ARs. The remaining clusters (including Cluster 12) were much smaller and covered between 1% to 3% of all ARs.

Table 6.5: Most relevant clusters obtained by clustering the debt origins obtained from approximately 2M of ARs.

	Cluster 1	**Cluster 12**	**Cluster 13**
Top 5 n-grams	contract; loan; credit; contractual; provide	social; national; social security; employment; state policy	distraint; court; proceeding; costs; distrainer
# receivables	1,594,165	58,283	249,195
% receivables	63.78%	2.33%	9.97%
Mean debt (USD)	$12,164	$13,465	$2,462
Mean debt as % of GDPPC[a]	52.7%	58.3%	10.7%

[a]GDP per Capita (GDDPC) of the Czech Republic was $23,101.778 in 2019, source: World Bank (https://data.worldbank.org).

The most frequent n-grams associated with the respective clusters (see Table 6.5) show that Cluster 1 contained generic defaulted loans, including mortgages, from various creditors. Cluster 13 instead consisted of the debts related to distraint (foreclosure) and mainly distrainers' fees. Cluster 12 represented debts caused by failed social security payments from employers. We counted the incidence frequency with the individual clusters for each creditor and administrator. Then, we transformed the frequencies into incidence rates by normalizing them, to sum up to 1. Using the same window function as before, we obtained 30 to 60 new features depending on the window size (15 for each year).

Preprocessing the dataset

Even after we removed the majority of steady debtor nodes from the considered data, the remaining dataset was still skewed regarding the target variable. Only a few sample nodes (of the order of tens) have retained a PR significantly larger than all the other nodes. On the other hand, these nodes exhibited the most noticeable changes over time. It was thus crucial to emphasize their role during training. However, most training algorithms tend to optimize against the prevalent class of available patterns, in our case, those nodes with smaller PRs. We employed a re-weighting scheme to mitigate this effect and assigned a weight to each sample in the dataset to emphasize rare patterns.

To each sample, we first assigned one of the three predefined categories (*small*, *medium*, and *large*), which we determined equidistantly according to the observed PR values. For instance, for the dataset generated using a rolling window of size 4, the maximum observed PR was 0.114, and the smallest was close to 0. Then, we divided this range among the three categories as follows: $small \sim (0, 0.0038]$, $medium \sim (0.0038, 0.0076]$, and $large \sim (0.0076, 0.0114]$. For $Dataset_4$, the category *small* comprised 3,970 samples, the category *medium* 29, and the last (*large*) 5. Then, we determined the actual sample weights using the weight category assigned to individual samples by employing the "balanced" heuristic from [7], accompanied by a log transform:

$$Weight(C) = log(1 + \frac{n_{samples}}{n_{categories} * countSamples(C)}) \tag{6.4}$$

Here, $n_{samples}$ denotes the total number of samples present in the dataset, $n_{categories}$ is the number of distinct categories (3), and $countSamples(C)$ counts the number of samples in the dataset belonging to a specific category ($C \in low, medium, large$). The log transform prevents underrepresented categories from obtaining weights that are too large. Because of too few samples from the class *medium* (29) and *large* (5), we further increased their number by oversampling with jitter. We generated two additional input patterns for each sample node from the dataset by applying uniform random noise of size at most 3% of the original value to each nonzero feature value.

Building the model

Changes that take place in the environment of insolvencies are rapid, non-linear, and difficult to capture. However, they usually only affect a small number of nodes simultaneously. For this reason, we chose a non-linear feed-forward neural network architecture with one hidden layer to predict the future impact of nodes (i.e., their PR). We have determined the number of neurons in the hidden layer for the considered neural network (NN) through a meta-parameter search using *Hyperopt* [133], see Table 6.6. First, we initialized the NN weights randomly using the normal distribution and chose the mean squared error (MSE) with L2 regularization as the loss function. Then, we trained the NN using the SGD algorithm implemented in *TensorFlow* [134]. Finally, we used the L2 regularized linear regression (LR) as a reference model.

Table 6.6: Learning parameters determined by Hyperopt.

	NN			**LR**
	# hidden neurons	Learning Rate	L2 reg.	L2 reg.
$Dataset_3$	25	0.929	4.579e-05	1.098
$Dataset_4$	26	0.908	4.575e-05	1.101
$Dataset5$	23	1.087	5.663e-05	1.102

Both models allow for some introspection to explain the development patterns encountered in the data. For LR, we can inspect the weights of the trained model and see which feature contributes most to the prediction. In the case of the NN, more complex methods, such as sensitivity analysis [135], are necessary. The mean sensitivity computed for each feature over the training set can be used to identify input features irrelevant to prediction. The proposed methodology iteratively prunes redundant inputs out of the network. After each pruning, the network has to be retrained shortly to adjust for missing inputs, and the method recalculates the model's performance on the test set. A significant drop in the model's accuracy would signal that we removed essential features from the model. Therefore, in our case, we stopped pruning when the test set's performance dropped more than 5% compared to the original model before pruning.

Algorithm 2 NN pruning and sensitivity-based re-training.

Input
    dataset split into *train*, *test* and *validation* parts, including category *small*, *medium*, *large* for each sample.
    max_drop is the maximum error rate drop allowed during the pruning procecure (in comparison to the orig. model), *default=0.05*
    step_size is the number of features pruned in each iteration, *default=10*

▷ Initialize the neural network using meta-parameters from Hyperopt
$model \leftarrow init\_nn()$

▷ Calculate sample weights using sample categories
$weights_{train} \leftarrow compute\_sample\_weights(data_{train})$
$weights_{valid} \leftarrow compute\_sample\_weights(data_{valid})$

▷ Fit the base model using SGD and employ early stopping using the validation dataset
$model.fit(dataset_{train}, dataset_{valid}, weights_{train}, weights_{valid})$

▷ Calculate the Mean Squared Error of the base model using the test dataset
$base\_error \leftarrow MSE(model, test)$

▷ Calculate the mean sensitivity of every feature across the training dataset and use it to sort the features from the least to the most sensitive ones
$feature_{sensitivities} \leftarrow sorted(mean\_feature\_sensitivity(model, train)$
$model_{pruned} \leftarrow model.copy()$

while $True$ **do**
    ▷ Take the *step_size* least sensitive features and remove them from the input layer of the NN (remaining weights stay the same) and the dataset
    $features\_to\_prune \leftarrow feature\_sensitivities.pop(step\_size)$
    $model_{iter}, datasest \leftarrow prune(model_{pruned}, dataset)$

    ▷ Retrain the neural net using the same meta-parameters
    $model_{iter}.fit(dataset_{train}, dataset_{valid}, weights_{train}, weights_{valid})$

    ▷ Re-evaluate the pruned model and if the error exceeds the maximum allowed limit, stop pruning
    $current\_error = MSE(model_{iter}, test)$
    if $current\_error > base\_error * (1 + max\_drop)$ **then**
        $break$
    end if
    $model\_pruned \leftarrow model_{iter}$
end while

▷ For each sample, calculate the mean sensitivity and use its squared root as the sample weight
$sens\_weights_{train} \leftarrow sqrt(mean\_feature\_sensitivity(model, dataset_{train})$
$sens\_weights_{valid} \leftarrow sqrt(mean\_feature\_sensitivity(model, dataset_{valid})$

▷ Re-train pruned model using sensitivity-based weights
$model_{sens} \leftarrow model_{pruned}.copy()$
$model_{sens}.fit(dataset_{train}, dataset_{valid}, sens\_weights_{train}, sens\_weights_{valid})$

return $model_{sens}$

On the other hand, samples with a considerable mean sensitivity might cause problems during prediction. A slight change in the input would cause a significant difference in the output. For this reason, we focused more on samples with large sensitivity during retraining. In our approach to retraining, we increased the samples' weights according to their sensitivity. In our case, the medium and large categories belonged to the most affected ones. Hence, we calculated each sample's mean sensitivity first, aggregating over the features' sensitivity coefficients. We adopted a similar re-weighting scheme to calculate the sample's actual weight as in the previous section. However, this time we divided the range of sensitivity values into five categories (instead of 3) and used the squared root transformation instead of the logarithmic one. Algorithm 2 summarizes the heuristic approach that we adopted for model training.

Evaluation

Hyperopt [133] framework helped us find the appropriate LR and NN parameters utilizing the default Tree-structured Parzen Estimator [136] (TPE) algorithm over 100 trials. The parameters we looked for comprised the number of neurons in the hidden layer, the learning rate, and the L2 regularization parameter. We used the scikit-learn [137] implementation of LR with an optimized L2 regularization parameter to train the baseline model. We evaluated each test using stratified 5-fold cross-validation (CV) with 80% training and 20% testing data. We used SGD with early stopping (limited to 2,000 epochs) to train the NN and evaluated each test with 5-fold stratified cross-validation (we used 60% of samples for training and 20% for each testing and validation). Both the NN and LR used the sample weights from Equation 6.4 during training. For the best hyper-parameters found, the NN was allowed to use up to 100,000 epochs (SGD), followed by pruning and sensitivity-based re-training according to Algorithm 2.

We repeated the experiments for three different datasets, with window sizes 3, 4, and 5. Table 6.6 presents the learning parameters obtained through the hyper-parameter search; they were similar for all three datasets. Figure 6.14 and Table 6.7 show that more expansive windows do not necessarily yield improved results. The future impact of a node, thus, is most affected by its recent development. We can observe a significant variance in the performance of LR across the datasets. Although not statistically significant, the NN model promises better overall performance with tighter confidence intervals. Consistently, we removed between 84% and 92% of the original features, and the subsequent sensitivity-based re-training further improved the network performance while performing best on $Dataset_3$. In both these aspects, NN utilizing L2 also outperforms training with dropout.

Table 6.7: 5-fold cross-validation results along with the 95% confidence intervals for MSE performance.

	# initial features	LR MSE ($\cdot 10^{-5}$)	$NN_{Dropout}$ MSE ($\cdot 10^{-5}$)	Pruned features	NN_{L2} MSE ($\cdot 10^{-5}$)	NN_{L2_Pruned} MSE ($\cdot 10^{-5}$)	Pruned features	$NN_{Sensitivity}$ MSE ($\cdot 10^{-5}$)
$Dataset_3$	234	317 ± 206	701 ± 121	13% (30)	279 ± 108	262 ± 090	85% (198)	**193 ± 072**
$Dataset_4$	351	466 ± 410	706 ±181	14% (48)	283 ± 135	270 ± 113	88% (310)	205 ± 096
$Dataset_5$	468	591 ± 512	796 ±136	22% (102)	321 ± 150	295 ± 118	92% (428)	244 ± 107

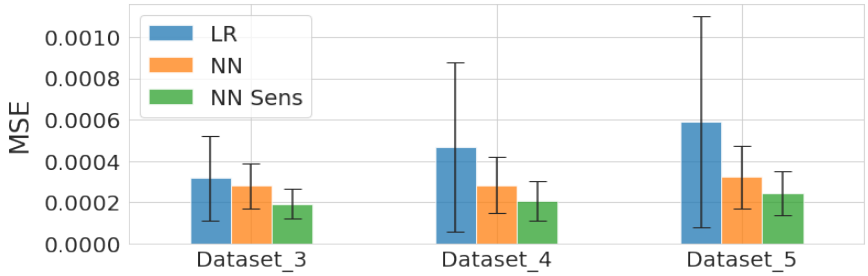


Figure 6.14: 5-fold CV results along with the 95% confidence intervals.

6.5.2 Results

Figure 6.15 shows the top 12 remaining features in the pruned NN trained for $Dataset_3$. The essential characteristics reported here remained consistent for the other datasets too. The results show that the most critical inputs to predict subsequent development were the PageRanks from the previous two years. Further essential inputs consisted of the shared nodes 13, 20, 6, and 7, referring to the corresponding most frequent creditor (or administrator). In the same order, these nodes were creditors *AB 5 BV*, *Czech Social Security Administration* (CSSA), *AB 4 BV*, and *Home Credit* (HC).

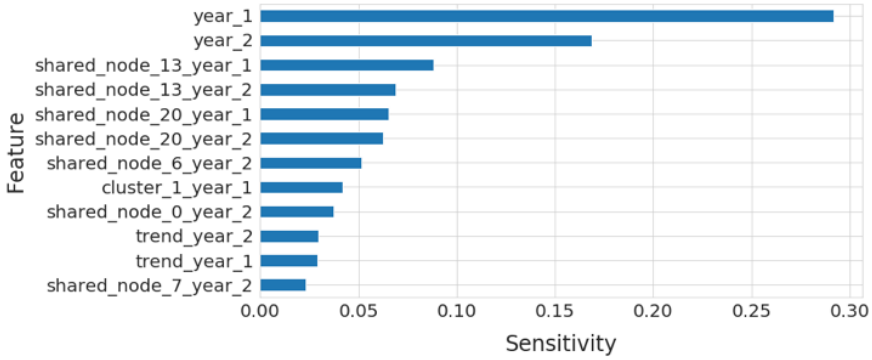
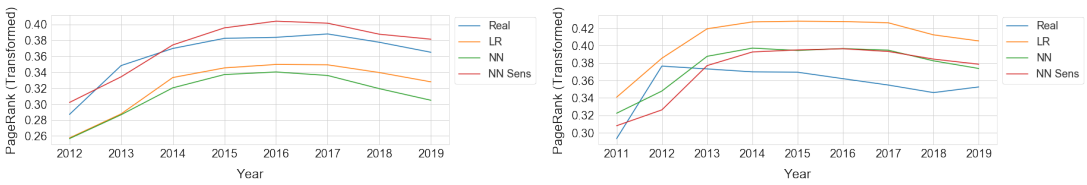


Figure 6.15: The mean sensitivities of top 12 features remaining after pruning one NN trained on $Dataset_3$.

AB 5 BV, *AB 4 BV*, and *Home Credit* all belong to the same financial group (*PPF*) known for frequent receivables ownership transfers among its entities. This observation might explain the predictive power of particular nodes in the network. For *AB 4 BV* and *Home Credit*, Figure 6.16 illustrates the performance of the individual models. For *AB 4 BV* (left), the proposed model (depicted in red) clearly outperforms the reference models and closely approaches the actual PR development (shown in blue). For *Home Credit* (right), the standard NN model (labeled by green) seems to match the performance of the proposed model; however, the sensitivity-based network achieved a similar accuracy with just a small fraction of inputs (15%). The proposed model also exploits information on debt's nature related to specific clusters like Cluster 1 and 13 (see Table 6.5). Cluster 12 refers to *CSSA* already encompassed by shared node 20 (as *CSSA* is the only receiver of social security payments in the Czech Republic). For this reason, Cluster 12 does not appear among the essential features.



(a) AB 4 B.V. (b) Home Credit

Figure 6.16: Real PR development of nodes *AB 4 BV* and *Home Credit* compared to the predictions made by individual models.

6.5.3 Summary

The social network we built in this experiment (and the prior ones) to capture the involved subjects' mutual relationships manifests several phenomena already known in dynamic social networks, such as densification. However, we still struggled to reveal the real relationship explaining the particular subjects' actual impact in the evolving system. To find such a relationship, we have proposed a general methodology that employs a relatively simple neural network as a viable alternative. Another option would be to consider, e.g., more recent but complex and computationally intensive deep learning approaches [138], [139]. The data analysis indicates that the actual influence (measured in the form of a PageRank) of the respective subjects (nodes) does not depend only on their previous values but also on other domain-specific features, such as the nature of the creditors and the type of debt. The main contribution of this experiment thus consists in:

1. Proposing a methodology that allows efficient analysis of influence evolution in social networks. An essential part of the developed approach represents a robust model of an artificial neural network. The found model succeeded in capturing the nodes'/actors' behavior and outperformed linear regression used for reference. The methodology also addresses skewed data with non-balanced data class distribution.

2. Identifying input pattern features that are crucial for an accurate model of social network evolution. We have used the notion of sensitivity to assess how much would (even slightly) changed in input affect the neural network's output. High sensitivity values indicate an inclination to swift changes in output values for even mildly altered inputs. Input features with low sensitivity can be, on the other hand, considered irrelevant, and we can prune them from the network. The remaining network benefits from short re-training afterward.

3. The evaluation of the developed approach on a real-life dataset extracted from the Czech Insolvency Register. The obtained results show the importance of both previous PageRank values and other characteristics describing the type of debt the considered subjects (creditors or administrators) share in common. In this context, the overlap appears to play a more significant role than the actual debt type.

Conclusion

In this thesis, we employed social network analysis (SNA) to comprehend the current structure of indebtedness in Czech society. We first designed and implemented a data processing pipeline capable of scraping, processing, and storing nearly all structured data from the IR (we omitted primarily sensitive data such as birth certificate numbers and dates of birth). Recognizing the significance of the 3.5 million applications of receivables (ARs) among all the 20 million documents submitted to the Insolvency Register (IR), we devised a custom document extraction technology named IRES. The primary purpose of IRES was to extract crucial details about the debt from the ARs, such as the debt value, its reason, and initial default dates. As a result, we prepared one of the most extensive insolvency-related datasets in the world (to our knowledge) named the *Czech insolvency dataset* (CID). The CID covers over 370,000 insolvency cases initiated between January 1, 2008, and December 31, 2022.

To further facilitate the experiments performed as a part of this thesis, we developed a new network processing framework named GraphSlices. This framework, implemented in Scala, addresses some of the shortcomings of existing frameworks (listed in Chapter 5) and enables the effective analysis of social networks represented by multigraphs. GraphSlices incorporates parallel implementations of key algorithms used in this thesis, namely PageRank and HITS, and other algorithms for graph structure analysis and clustering.

Utilizing the CID, we constructed an unweighted static social network that modeled the interactions among the subjects involved in insolvencies, including debtors, creditors, insolvency administrators, and insolvency courts (insolvency actors). Even with this simple network, we were able to draw novel insights into the structure of indebtedness in the Czech Republic. Notably, the static insolvency network demonstrated high density and revealed a small number of highly influential nodes (creditors) consistently participating in the same insolvency proceedings. We further confirmed this finding by applying a community detection algorithm to the static insolvency network, uncovering two distinct groups of creditors with diverse characteristics. Financial institutions constituted the first group, while nonfinancial creditors, such as phone providers, formed the second major group.

Subsequently, we showed that insolvency-related data can also be effectively modeled as a dynamic social network. This was achieved by enriching the static insolvency network with edge timestamps, indicating the occurrence of edges (or relationships) in the network. Utilizing this enriched insolvency network, we analyzed the network's structural evolution over time, assessing the importance of nodes in various time snapshots. Our findings revealed significant changes in the insolvency network over time, indicating a highly dynamic nature. The set of most influential nodes exhibited notable changes, with established actors becoming stagnant and new participants emerging in the IR. In insolvency proceedings, rule mining can be applied to underlying network data to predict future edge occurrences, allowing us to capture associations among insolvency actors in the form of rules.

Next, we utilized unstructured data extracted from ARs, specifically the textual description of the reasons for debt, to employ clustering to categorize debtors into different groups based on the nature of their debt. Moreover, we identified the most typical keywords for each cluster to assign meaningful semantic descriptions to the categories of debtors. Our findings revealed that over 50% of ARs in insolvencies originated from financial products provided by banks and non-banking lenders.

Then, we utilized the debt amount, also extracted from ARs, to quantify the money claimed across Czech society. The estimated debt value claimed against individuals only by the 18 largest creditors exceeded 28 billion CZK (1.1 billion USD). Leveraging the debt amount of individual claims, we estimated the extent of individuals' indebtedness to different creditors. This analysis unveiled that the most frequent creditors and those with the largest amount of debt did not necessarily align. For instance, the largest creditor in terms of the number of ARs was GE Money Bank (over 60,000), but in terms of the total debt claimed against natural persons, it was Česká spořitelna (8.5 billion CZK).

The subsequent clustering of individual insolvencies revealed groups of debtors with different characteristics, encompassing variations in the size of their debt and how it was accumulated. For example, one of the identified clusters contained medium-sized debtors with an overall debt ranging between 100,000 CZK and 200,000 CZK, and these debts originated mainly from consumer loans provided by non-bank lenders. Another cluster represented debtors with large overall debts ranging from 1 to 2 million CZK, mainly constituted by mortgages from the largest banks in the Czech Republic, such as Česká spořitelna, Raiffeisenbank, and Komerční banka.

Finally, we proposed a methodology based on neural networks to assess the insolvency network's future development by predicting the significance of subjects involved in insolvency proceedings measured as PageRank. To construct the prediction model, we developed a methodology that addressed the primary challenges associated with using predictive models, specifically neural networks, on social network data. These challenges encompassed handling extreme imbalance present in the training data, as the network typically only contains a very small number of highly impactful nodes. Our methodology employed a feed-forward neural network architecture, and we utilized sensitivity analysis to interpret the final model, identifying key variables influencing the future development in the network.

Our findings indicated that the current influence of nodes, measured as PageRank, depends not only on previous influence (i.e., previous PageRank values) but also on other domain-specific features, such as the nature of creditors and the type of debt. This underscores the significance of incorporating all three facets present in the insolvency data: the structure of the relationship, the timestamp of the occurrence of edges, and additional metadata about the nodes/edges. Our approach results in more robust and accurate models for predicting future development in the insolvency network.

Thus, by applying SNA to insolvency-related data we were able to:

1. Effectively model the insolvency process using a static social network approach.
2. Effectively model the insolvency process using a dynamic social network approach.
3. Gain additional insights into the insolvency process by enriching the insolvency network with additional metadata extracted from the IR, such as the size of the debt and its nature.
4. Incorporate all three facets of insolvency data, namely, the structure of relationships, the timestamps marking the occurrence of relationships, and domain-specific metadata to build a superior model for predicting future development in the insolvency network.

Future work

While our research has yielded valuable insights into insolvency proceedings, several directions for future exploration and development emerged. We propose three potential directions to enhance the potential of this methodology further.

One of the primary limitations we encountered in this study was the restricted scope of data available in the Insolvency Register, which primarily encompasses information regarding insolvency proceedings. Data such as financial statements, which are publicly accessible through the Company Register of the Czech Republic and credit risk databases, could significantly enrich our understanding of nodes' historical development prior to the bankruptcy and their overall financial health. This additional data could lead to even more accurate models for predicting future development in the insolvency process.

In recent years, there has been a notable shift towards applying deep learning techniques, particularly graph neural networks, to analyze and extract information from graph-structured data. Graph neural networks [140] have demonstrated remarkable progress in their capabilities and expressive powers. Future research in this domain can explore integrating advanced graph neural network models into our methodology and using them to discover more complex patterns.

Beyond the area of insolvency analysis, the methodologies and insights developed in this research have the potential to find practical applications in various other domains. Future research can explore how similar approaches can be adapted to analyze social networks in other contexts, such as corporate mergers and acquisitions or supply chain management.

Bibliography

[1] Charu C. Aggarwal and Philip S. Yu. Online analysis of community evolution in data streams. In *Proceedings of the 2005 SIAM International Conference on Data Mining (SDM)*, pages 56–67, 2005.

[2] Brian W. Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

[3] Mark E. J. Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 69:026113, 2004.

[4] Stijn van Dongen. *Graph Clustering by Flow Simulation.* PhD thesis, University of Utrecht, 2000.

[5] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 137–146, 2003.

[6] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. In *Proceedings of the 7th International World Wide Web Conference*, pages 161–172, 1998.

[7] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM '03, page 556–559, 2003.

[8] Yang Zhou, Hong Cheng, and Jeffrey X. Yu. Graph clustering based on structural/attribute similarities. *Proc. VLDB Endow.*, 2(1):718–729, 2009.

[9] Lisa Getoor, Nir Friedman, Daphne Koller, and Benjamin Taskar. Learning probabilistic models of link structure. *J. Mach. Learn. Res.*, 3:679–707, 2003.

[10] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

[11] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[12] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '99, pages 251–262, 1999.

[13] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, page 177–187, 2005.

[14] Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S. Tomkins. The web as a graph: Measurements, models, and methods. In *Computing and Combinatorics: 5th Annual International Conference*, pages 1–17, 1999.

[15] Mark E. J. Newman. Power laws, pareto distributions and zipf's law. *Contemporary Physics*, 46(5):323–351, 2005.

[16] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Diameter of the world-wide web. *Nature*, 401:130–131, 1999.

[17] Jure Leskovec and Eric Horvitz. Planetary-scale views on a large instant-messaging network. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, page 915–924, 2008.

[18] Gary W. Flake, Steve Lawrence, C. Lee Giles, and Frans Coetzee. Self-organization and identification of web communities. *Computer*, 35(3):66–71, 2002.

[19] Michelle Girvan and Mark E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[20] Alexander J. Smola and Risi Kondor. Kernels and regularization on graphs. In *Computational Learning Theory and Kernel Machines, 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop*, volume 2777 of *Lecture Notes in Computer Science*, pages 144–158, 2003.

[21] David Aldous and James A. Fill. *Reversible Markov Chains and Random Walks on Graphs*. 2002.

[22] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, 1998.

[23] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. Extrapolation methods for accelerating pagerank computations. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 261–270, 2003.

[24] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.

[25] Glen Jeh and Jennifer Widom. Simrank: A measure of structural-context similarity. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, page 538–543, 2002.

[26] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[27] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social Networks*, 25:211–230(20), 2003.

[28] John C. Gower. A general coefficient of similarity and some of its properties. *Biometrics*, 27(4):857–871, 1971.

[29] Hawoong Jeong, Sean Mason, Albert-Laszlo Barabási, and Zoltan N. Oltvai. Lethality and centrality in protein networks. *Nature*, 411(6833):41–42, 2001.

[30] Janos Podani, Zoltan Oltvai, Hawoong Jeong, Albert-Laszlo Barabasi, and Eörs Szathmáry. Comparable system-level organization of archaea and eukaryotes. *Nature genetics*, 29:54–6, 2001.

[31] Jordi Bascompte, Pedro Jordano, Carlos Melian, and Jens Olesen. The nested assembly of plant-animal mutualistic networks. *Proceedings of the National Academy of Sciences of the United States of America*, 100:9383–7, 2003.

[32] Jennifer Dunne, Richard Williams, and Neo Martinez. Network structure and biodiversity loss in food webs: Robustness increases with connectance. *Ecology Letters*, 5:558 – 567, 2002.

[33] James Abello, Panos M. Pardalos, and Mauricio G. C. Resende. On maximum clique problems in very large graphs. In *External Memory Algorithms*, volume 50 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 119–130, 1998.

[34] Mark E. J. Newman. Assortative mixing in networks. *Physical Review Letters*, 89(20):208701, 2002.

[35] Ramon Ferrer i Cancho and Richard V. Solé. The small world of human language. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 268(1482):2261–2265, 2001.

[36] Noel M. Tichy, Michael L. Tushman, and Charles Fombrun. Social network analysis for organizations. *The Academy of Management Review*, 4(4):507–519, 1979.

[37] Einat Sprinzak, Shmuel Sattath, and Hanah Margalit. How reliable are experimental protein-protein interaction data? *Journal of Molecular Biology*, 327(5):919–923, 2003.

[38] U Kang, Charalampos E. Tsourakakis, Ana P. Appel, Christos Faloutsos, and Jure Leskovec. Radius plots for mining tera-byte scale graphs: Algorithms, patterns, and observations. In *Proceedings of the SIAM International Conference on Data Mining, SDM*, pages 548–558, 2010.

[39] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[40] Rangaramanujam Kannan, S. Santosh Vempala, and Adrian Veta. On clusterings-good, bad and spectral. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 367, 2000.

[41] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On finding graph clusterings with maximum modularity. In *Graph-Theoretic Concepts in Computer Science*, volume 4769 of *Lecture Notes in Computer Science*, pages 121–132. 2007.

[42] Michael R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified np-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976.

[43] Mark E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69:066133, 2004.

[44] Aaron Clauset, Mark E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 70:066111, 2004.

[45] Ulrike Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[46] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(98):298–305, 1973.

[47] Sylvain Brohée and Jacques van Helden. Evaluation of clustering algorithms for protein-protein interaction networks. *BMC Bioinform.*, 7:488, 2006.

[48] Venu Satuluri and Srinivasan Parthasarathy. Scalable graph clustering using stochastic flows: Applications to community discovery. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, page 737–746, 2009.

[49] Venu Satuluri, Srinivasan Parthasarathy, and Duygu Ucar. Markov clustering of protein interaction networks with improved balance and scalability. In *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology*, pages 247–256, 2010.

[50] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE Trans. on Knowl. and Data Eng.*, 15(3):515–528, 2003.

[51] Olfa Nasraoui, Cesar Cardona Uribe, Carlos Rojas Coronel, and Fabio Gonzalez. Tecno-streams: Tracking evolving clusters in noisy data streams with a scalable immune system learning model. In *Proceedings of the Third IEEE International Conference on Data Mining*, ICDM '03, page 235, 2003.

[52] Tanja Falkowski, Anja Barth, and Myra Spiliopoulou. Studying community dynamics with an incremental graph mining algorithm. In *Proceedings of the 14th Americas Conference on Information Systems (AMCIS)*, page 29, 2008.

[53] T. Falkowski, J. Bartelheimer, and M. Spiliopoulou. Community dynamics mining. In *Proc. of 14th European Conference on Information Systems*, 2006.

[54] Tanja Falkowski, Jörg Bartelheimer, and Myra Spiliopoulou. Mining and visualizing the evolution of subgroups in social networks. In *Web Intelligence*, pages 52–58, 2006.

[55] Myra Spiliopoulou, Irene Ntoutsi, Yannis Theodoridis, and Rene Schult. Monic - modeling and monitoring cluster transitions. In *Proc. of KDD'06*, pages 706–711, 2006.

[56] Jure Ferlez, Christos Faloutsos, Jure Leskovec, Dunja Mladenic, and Marko Grobelnik. Monitoring network evolution using MDL. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pages 1328–1330, 2008.

[57] Deepayan Chakrabarti, Ravi Kumar, and Andrew Tomkins. Evolutionary clustering. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 554–560, 2006.

[58] Purnamrita Sarkar and Andrew W. Moore. Dynamic social network analysis using latent space models. In *Advances in Neural Information Processing Systems*, pages 1145–1152, 2005.

[59] Yun Chi, Xiaodan Song, Dengyong Zhou, Koji Hino, and Belle L. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *Knowledge Discovery and Data Mining*, page 153–162, 2007.

[60] Ata Kaban and Mark Girolami. A dynamic probabilistic model to visualise topic evolution in text streams. *Journal of Intelligent Information Systems*, 18(2/3):107–125, 2002.

[61] Satoshi Morinaga and Kenji Yamanishi. Tracking dynamics of topic trends using a finite mixture model. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 811–816, 2004.

[62] André Gohr, Alexander Hinneburg, Rene Schult, and Myra Spiliopoulou. Topic evolution in a stream of documents. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 859–872, 2009.

[63] Yu-Ru Lin, Yun Chi, Shenghuo Zhu, Hari Sundaram, and Belle L. Tseng. Analyzing communities and their evolutions in dynamic social networks. *ACM Trans. Knowl. Discov. Data*, 3(2), 2009.

[64] Hisashi Kashima and Naoki Abe. A parameterized probabilistic model of network evolution for supervised link prediction. In *Proceedings of the Sixth International Conference on Data Mining*, pages 340–349, 2006.

[65] Jérôme Kunegis and Andreas Lommatzsch. Learning spectral graph transformations for link prediction. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 561–568, 2009.

[66] Chao Wang, Venu Satuluri, and Srinivasan Parthasarathy. Local probabilistic models for link prediction. In *Proceedings of the 7th IEEE International Conference on Data Mining*, pages 322–331, 2007.

[67] Aaron Clauset, Cristopher Moore, and Mark E. J. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453:98–101, 2008.

[68] Sisay Fissaha Adafre and Maarten de Rijke. Discovering missing links in wikipedia. In *Proceedings of the 3rd international workshop on Link discovery*, LinkKDD '05, pages 90–97, 2005.

[69] Jianhan Zhu, Jun Hong, and John G. Hughes. Using markov models for web site link prediction. In *Hypertext*, pages 169–170, 2002.

[70] Zan Huang, Xin Li, and Hsinchun Chen. Link prediction approach to collaborative filtering. In *JCDL '05: Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, pages 141–142, 2005.

[71] Valerio Freschi. A graph-based semi-supervised algorithm for protein function prediction from interaction maps. In *LION*, volume 5851 of *Lecture Notes in Computer Science*, pages 249–258, 2009.

[72] Janardhan Rao Doppa, Jun Yu, Prasad Tadepalli, and Lise Getoor. Learning algorithms for link prediction based on chance constraints. In *Machine Learning and Knowledge Discovery in Databases*, pages 344–360, 2010.

[73] Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, and Mohammed Zaki. Link prediction using supervised learning. In *In Proc. of SDM 06 workshop on Link Analysis, Counterterrorism and Security*, 2006.

[74] Joshua O'Madadhain, Jon Hutchins, and Padhraic Smyth. Prediction and ranking algorithms for event-based network data. *SIGKDD Explor. Newsl.*, 7(2):23–30, 2005.

[75] Matthew J. Rattigan and David D. Jensen. The case for anomalous link discovery. *SIGKDD Explorations*, 7(2):41–47, 2005.

[76] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[77] Seyda Ertekin, Jian Huang, and Lee Giles. Active learning for class imbalance problem. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 823–824, 2007.

[78] Grigoris Karakoulas and John Shawe-Taylor. Optimizing classifiers for imbalanced training sets. In *Proceedings of the 11th International Conference on Neural Information Processing Systems*, page 253–259, 1998.

[79] Foster J. Provost and Tom Fawcett. Robust classification for imprecise environments. *Mach. Learn.*, 42(3):203–231, 2001.

[80] Chan-Yun Yang, Jr-Syu Yang, and Jian-Jun Wang. Margin calibration in SVM class-imbalanced learning. *Neurocomputing*, 73(1-3):397–411, 2009.

[81] John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pages 61–74, 1999.

[82] Nilotpal Chakravarti. Isotonic median regression: A linear programming approach. *Mathematics of Operations Research*, 14(2):303–308, 1989.

[83] Gary M. Weiss. Mining with rarity: A unifying framework. *SIGKDD Explor. Newsl.*, 6(1):7–19, 2004.

[84] Gwangbum Pyun, Unil Yun, and Keun Ho Ryu. Efficient frequent pattern mining based on linear prefix tree. *Knowledge-Based Systems*, 55:125–139, 2014.

[85] Joo-Chang Kim and Kyung-Yong Chung. Mining based time-series sleeping pattern analysis for life big-data. *Wireless Personal Communications*, 105:475–489, 2018.

[86] Shashi Shekhar, Michael R. Evans, James M. Kang, and Pradeep Mohan. Identifying patterns in spatial information: A survey of methods. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1:193–214, 2011.

[87] Xifeng Yan and Jiawei Han. gSpan: graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining*, pages 721–724, 2002.

[88] Diane J. Cook and Lawrence B. Holder. Graph-based data mining. *IEEE Intelligent Systems*, 15(2):32–41, 2000.

[89] Mohammed J. Zaki. Efficiently mining frequent embedded unordered trees. *Fundam. Informaticae*, 66:33–52, 2004.

[90] Karsten M. Borgwardt, Hans-Peter Kriegel, and Peter Wackersreuther. Pattern mining in frequent dynamic subgraphs. *Sixth International Conference on Data Mining*, pages 818–822, 2006.

[91] Abhijin Adiga, A. Vullikanti, and Dante Wiggins. Subgraph enumeration in dynamic graphs. *2013 IEEE 13th International Conference on Data Mining*, pages 11–20, 2013.

[92] U. Redmond, M. Harrigan, and P. Cunningham. Identifying time-respecting subgraphs in temporal networks. In *In Proceedings of the 3rd International Workshop on Mining Ubiquitous and Social Environments*, 2012.

[93] Michele Berlingerio, Francesco Bonchi, Björn Bringmann, and Aristides Gionis. Mining graph evolution rules. In *Machine Learning and Knowledge Discovery in Databases*, pages 115–130, 2009.

[94] Cane Wing ki Leung, Ee-Peng Lim, D. Lo, and Jianshu Weng. Mining interesting link formation rules in social networks. *Proceedings of the 19th ACM international conference on Information and knowledge management*, page 209–218, 2010.

[95] Wolters Kluwer. *Insolvency Act.* 2011.

[96] Marie Paseková. Personal bankruptcy and its social implications. *International Advances in Economic Research*, 19(3):319–320, 2013.

[97] Brian Suda. *SOAP Web Services.* 2003.

[98] Antonino Virgillito. *Publish/Subscribe Communication Systems: from Models to Application.* 2003.

[99] Henry S. Thompson et al. *XML Schema Part1: Structures (Second Edition).* 2004.

[100] W3C. XML Path Language (XPath) 2.0. `https://www.w3.org/TR/xpath20/`, 2011. Accessed: 2023-12-12.

[101] Ray Smith. An overview of the Tesseract OCR engine. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02*, ICDAR '07, pages 629–633, 2007.

[102] Chirag Patel, Atul Patel, and Dharmendra Patel. Article: Optical character recognition by open source OCR tool Tesseract: A case study. *International Journal of Computer Applications*, 55(10):50–56, 2012.

[103] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[104] Ahmad P. Tafti, Ahmadreza Baghaie, Mehdi Assefi, Hamid R. Arabnia, Zeyun Yu, and Peggy Peissig. OCR as a service: An experimental evaluation of Google Docs OCR, Tesseract, ABBYY FineReader, and Transym. In *Advances in Visual Computing*, pages 735–746, 2016.

[105] Wojciech Bieniecki, Szymon Grabowski, and Wojciech Rozenberg. Image preprocessing for improving OCR accuracy. In *2007 International Conference on Perspective Technologies and Methods in MEMS Design*, pages 75–80, 2007.

[106] Matteo Brisinello, Ratko Grbić, Matija Pul, and Tihomir Anelić. Improving optical character recognition performance for low quality images. In *2017 International Symposium ELMAR*, pages 167–171, 2017.

[107] Michael Still. *The Definitive Guide to ImageMagick (Definitive Guide).* 2005.

[108] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision in C++ with the OpenCV Library.* 2nd edition, 2013.

[109] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[110] Nahum Kiryati, Yonina Eldar, and Alfred Bruckstein. A probabilistic hough transform. *Pattern Recognition*, 24(4):303 – 316, 1991.

[111] Olfa Nasraoui. Web data mining: Exploring hyperlinks, contents, and usage data. *SIGKDD Explor. Newsl.*, 10(2):23–25, 2008.

[112] Vladimir N. Vapnik. *The nature of statistical learning theory*. 1995.

[113] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1):489–501, 2006.

[114] Sascha Schimke and Claus Vielhauer. Similarity searching for on-line handwritten documents. *Journal on Multimodal User Interfaces*, 1:49–54, 2008.

[115] Gabor Csardi and Tamas Nepusz. The Igraph software package for complex network research. *InterJournal, Complex Systems*, (5):1–9, 2005.

[116] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, 2008.

[117] Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Trans. Intell. Syst. Technol.*, 8(1), 2016.

[118] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, page 135–146, 2010.

[119] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, page 17–30, 2012.

[120] Reynold S. Xin, Joseph E. Gonzalez, Michael J. Franklin, and Ion Stoica. Graphx: a resilient distributed graph system on spark. In *First International Workshop on Graph Data Management Experiences and Systems*, pages 2–8, 2013.

[121] Vasiliki Kalavri, Vladimir Vlassov, and Seif Haridi. High-level programming abstractions for distributed graph processing. *IEEE Transactions on Knowledge and Data Engineering*, 30(2):305–324, 2018.

[122] Wentao Han, Youshan Miao, Kaiwei Li, Ming Wu, Fan Yang, Lidong Zhou, Vijayan Prabhakaran, Wenguang Chen, and Enhong Chen. Chronos: A graph engine for temporal graph analysis. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, 2014.

[123] Raymond Cheng, Ji Hong, Aapo Kyrola, Youshan Miao, Xuetian Weng, Ming Wu, Fan Yang, Lidong Zhou, Feng Zhao, and Enhong Chen. Kineograph: Taking the pulse of a fast-changing and connected world. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, page 85–98, 2012.

[124] Anand Padmanabha Iyer, Li Erran Li, Tathagata Das, and Ion Stoica. Time-evolving graph processing at scale. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*, GRADES '16, 2016.

[125] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, page 10, 2010.

[126] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An open source software for exploring and manipulating networks. *Proceedings of the International AAAI Conference on Web and Social Media*, 3(1), 2009.

[127] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[128] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.

[129] Teuvo Kohonen. *Self-Organizing Maps*. Springer Series in Information Sciences. 2012.

[130] Joseph C. Dunn. Well-separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4(1):95–104, 1974.

[131] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[132] Juha Vesanto and Esa Alhoniemi. Clustering of the self-organizing map. *IEEE Transactions on Neural Networks*, 11(3):586–600, 2000.

[133] James Bergstra, Daniel Yamins, and D. David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page 115–123, 2013.

[134] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, page 265–283, 2016.

[135] Jacek M. Zurada, Aleksander Malinowski, and Ian Cloete. Sensitivity analysis for minimization of input data dimension for feedforward neural network. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 447–450, 1994.

[136] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, page 2546–2554, 2011.

[137] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[138] Hoon-Keng Poon, Wun-She Yap, Yee-Kai Tee, Wai-Kong Lee, and Bok-Min Goi. Hierarchical gated recurrent neural network with adversarial and virtual adversarial training on text classification. *Neural Networks*, 119:299–312, 2019.

[139] Cosimo Ieracitano, Nadia Mammone, Amir Hussain, and Francesco C. Morabito. A novel multi-modal machine learning based approach for automatic classification of EEG recordings in dementia. *Neural Networks*, 123:176–190, 2020.

[140] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

List of Figures

List of Tables

List of Abbreviations

AR Application of receivables

CR Czech Republic

CID Czech insolvency dataset

GCC Giant connected component

HITS Hyperlink-induced topic search

IRES Insolvency register extraction system

IR Insolvency Register of the Czech Republic

IP Insolvency proceeding

MCL Markov clustering algorithm

ML Machine learning

NN Neural network

PR Page Rank

SNA Social network analysis

SOFM Self-organizing feature map

List of publications

- Iveta Mrázová and Peter Zvirinský. Mining the czech insolvency proceedings data. *Procedia Computer Science*, 36:308–313, 2014. Complex Adaptive Systems Philadelphia, PA November 3-5, 2014.
- Iveta Mrázová and Peter Zvirinský. Czech insolvency proceedings data: Social network analysis. *Procedia Computer Science*, 61:52–59, 2015. Complex Adaptive Systems San Jose, CA November 2-4, 2015.
- Iveta Mrázová and Peter Zvirinský. Extraction and interpretation of textual data from czech insolvency proceedings. In Leszek Rutkowski, Marcin Korytkowski, Rafal Scherer, Ryszard Tadeusiewicz, Lotfi A. Zadeh, and Jacek M. Zurada. *Artificial Intelligence and Soft Computing*, pages 116–125, Cham, 2017.
- Iveta Mrázová and Peter Zvirinský. Dynamic Social Networks and Their Analysis. In Jaroslav Zendulka, Mária Bieliková, Radek Burget, Zbyněk Křivka. *Data a znalosti & WIKT 2018*, pages 231–236, 2018.
- Iveta Mrázová and Peter Zvirinský. Czech insolvency proceedings: Extraction of numerical information and its analysis. In *Proceedings of the 2018 10th International Conference on Machine Learning and Computing*, ICMLC 2018, page 150–156, New York, NY, USA, 2018.
- Iveta Mrázová and Peter Zvirinský. Subjects involved in czech insolvency proceedings: An assessment of their future impact. *Procedia Computer Science*, 185:63–72, 2021. Big Data, IoT, and AI for a Smarter Future.

A. Dataset schema documentation

This section contains detailed descriptions of individual entities considered in the database schema designed for the *Czech insolvency dataset* (CID), including individual fields we store in the database.

A.1 Insolvency

Insolvency is the core entity of the proposed relational model and represents a single insolvency proceeding. This entity contains all the basic information related to an IP, most of which can be found on the IP's detail page (see Figure 3.3). We list the attributes of the Insolvency entity in Table A.1.

Table A.1: Insolvency entity attributes

Attribute	**Description**	**Example**
id	A unique identifier generated from the IP's reference number.	msphins9997/2010
debtor_name	Name of the debtor.	TAKING s.r.o.
ico	Organization identification number, if the subject is a business entity.	27204316
reference_number	Official reference number of the IP.	MSPH 93 INS 9997 / 2010
proposal_timestamp	Date and time when the IP was commenced and the insolvency petition was submitted.	2010-09-01 16:42:00
url	URL to the IP's detail page (eventually becomes unavailable).	`https://isir.justice.cz/isir/ueu/evidence_upadcu_detail.do?id=AF7BD0355DAD6E...ADE05333F21FAC39CD`
gender	Debtor's gender (if available).	M
debtor_address	Debtor's domicile (if available).	Praha 10 Vršovice, Moskevská 617/58, PSČ 101 00
year_of_birth	Debtor's year of birth (if available).	1960
region_id	Id if the region extracted from debtor's domicile.	1

A.2 Subject

Subject is a generic entity aggregating all the subjects involved in any IP in any role (debtor, creditor, or administrator). Its purpose is to provide detailed information about subjects only accessible from the IR's web service. All the attributes of the subject entity are listed in Table A.2.

Table A.2: Subject entity attributes

Attribute	**Description**	**Example**
name	Name of the subject	Ing. Jana Vodrážková
ico	Organization identification number, if the subject is a business entity.	27204316
form	Specifies whether the subject is a natural person (F) or a legal entity (P).	P
legal_form	Legal form of the subject, if the subject is a legal entity.	a.s.
address_form	The type of address provided by the subject, e.g. permanent residence (∼ "TRVALÁ"), temporary residence, etc.	TRVALÁ
address_city	City/town component of the address.	Prague
address_street	Street component of the address.	Štepánska
address_country	Country component of the address.	Česká republika
address_zip_code	ZIP code component of the address.	150 00

A.3 Administrator

One or more administrators are associated with every IP that has reached at least the *Bankruptcy* stage (see Figure 3.4). Unfortunately, the IR provides minimal information about each administrator (see Table A.3). We can only obtain the administrator's name and role in the given IP.

Table A.3: Administrator entity attributes

Attribute	**Description**	**Example**
id	Administrator's internal ID (automatically generated).	1
administrator	Name of the administrator.	Ing. Jana Vodrážková
administrator_type	Type of the administrator.	Insolvenční správce

A.4 Insolvency State

The Insolvency State entity captures the entire history of all the states a particular IP has gone through. One instance of the Insolvency State entity represents a state change of exactly one IP. The attributes of this entity are listed in Table A.4.

Table A.4: Insolvency State entity attributes

Attribute	Description	Example
insolvency_id	ID of the IP the state is connected to.	msphins9997/2010
state	The new state of the IP (as defined in Figure 3.4)	Bankruptcy
state_change_timestamp	Date and time when the state change occurred.	2010-09-01 16:42:00
action_id	ID of the event published by the Web Service which moved the IP into this state.	1234

A.5 File

The File entity represents all files (or documents) related to individual IPs published in the IR. This entity only stores the metadata about the files (like type and publication date) and the textual content obtained by applying OCR on some of them (mostly ARs). We list all the File entity attributes in Table A.5.

Table A.5: File entity attributes

Attribute	Description	Example
id	Internal ID of the document (automatically generated).	1
file_type	Type of the document.	Přihláška pohledávky *(∼ application of receivable)*
url	Publicly accessible URL of the document.	`https://isir.justice.cz/isir/doc/dokument.PDF?id=1592027`
insolvency_id	ID of the IP to which this document is related to.	msphins9997/2010
publish_date	Date and time when the document was published in the IR.	2010-09-01 16:42:00
document_section	Document section as described in Section 3.2.2.	A
content	Text extracted from the document using OCR.	Soud Městský soud v Praze...

A.6 Receivable

The Receivable entity represents data related explicitly to receivables, such as the creditor's name and all the additional meta-data we extracted using our extraction pipeline, namely the receivable values. We provide the list of all attributes in Table A.6.

Table A.6: Receivable entity attributes

Attribute	**Description**	**Example**
file_id	ID of the document this receivable is related to.	1
creditor	Name of the creditor.	HOLUB Roman s.r.o.
total	The total value of the receivable (or debt value).	100 000 Kč
secured	The value of secured receivables.	50 000 Kč
unsecured	The value of unsecured receivables.	50 000 Kč
status	Validation status of the extracted values as defined in Table 4.11.	msphins9997/2010

A.7 WS Event

The WS Event entity corresponds to all the events published in the IR's web service. Every WS Event represents a discrete change of a single insolvency. We list the WS Event's attributes in Table A.7.

Table A.7: WS Event entity attributes

Attribute	**Description**	**Example**
id	Unique ID of the event.	1
event_timestamp	Date and time when the event was published.	2010-09-01 16:42:00
insolvency_id	ID of the IP the event is connected to.	msphins9997/2010
event	The event's data in the original XML format.	`<localId>2538</localId>`

B. Attachments

B.1 Czech insolvency dataset

The CID (i.e., CID DB) used in this thesis is too large to be stored directly in the attachments; that is why it is included on the physical SD card attached to the printed version of this thesis and also published publicly at `https://github.com/zviri/czech-insolvency-dataset`.

To restore the CID DB from the SD Card, follow these steps:

1. Copy the archive `cid_db.sql.gz` into a local folder.
2. Decompress the database SQL archive:
   `gzip -d cid_db.sql.zip`
3. Restore the CID DB into an empty PostgresSQL database:
   `pg_restore -U $USERNAME -d $DB_NAME -1 cid_db.sql`

B.2 Source code

In the attachment, we provide the most relevant source code for this thesis, organized in the following folder structure:

- *IRES* — All source codes related to the Insolvency Register Extraction system, including all the scrapers and the main OCR engine.
  - *isir.data-access* — Common functionality shared by other projects.
  - *isir.html-scraper* — Scraper for extracting insolvency data from the IR's web application.
  - *isir.ocr* — Main OCR engine, including functionality for information extraction.
  - *isir.ocr-reliability-tes* — OCR engine evaluation functionality.
  - *isir.sql-runner* — Simple project for running scheduler SQL queries.
  - *isir.ws-cache-scraper* — Scraper for downloading insolvency events from IR's web service.
  - *isir.ws-scraper* — Scraper for updating changes insolvencies.
  - *cid_db* — SQL init script for the CID DB.
- *GraphSlices* — Full implementation of GraphSlices, including tests.
- *PredictionModel* — Complete implementation of the prediction methodology used in Section 6.5.