UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Madis-Karli Koppel

Large Scale Feature Extraction from Linked Web Data

Master's Thesis (30 ECTS)

Supervisor: Pelle Jakovits, PhD
Supervisor: Peep Küngas, PhD

Tartu 2017

Acknowledgment

To Maarja, for being so understandable about the long nights of writing and for being a motivator throughout these two years. Many Thanks!

I am grateful to my supervisors, Pelle Jakovits and Peep Küngas, who went above and beyond to provide feedback and guidelines for a lost student.

I would like to thank Pärt for helping with the hardest parts of this thesis - commas and words.

I am thankful to IT Academy for their continuous financial support during my studies.

I would like to thank my employer Tarkvara Tehnoloogia Arenduskeskus (STACC) who supported this thesis.

This research has been supported by European Regional Development Fund under the grant no EU48684.

Large Scale Feature Extraction from Linked Web Data

Abstract:
Data available on the web is evolving, and the way it is represented is changing as well. Linked data has made information on the web understandable to machines. In this thesis we develop a proof of concept pipeline that extracts linked data from web crawling and performs feature extraction on it. The end goal of this pipeline is to provide input to machine learning models that are used for credit scoring. The use case focuses on extracting product linked data and connecting it with the company that offers it. Built solution attempts to detect if two products from different web sites are the same in order to use one representation for both. Information about companies and products is represented as a graph on which network metrics are calculated. Network metrics from multiple different web crawls are stored in time series that shows changes in graph over time. We then calculate derivatives on the values in time series.

The developed pipeline is designed to handle data in terabytes and built with scalability in mind. We use Apache Spark to process huge amounts of data and to be ready if input data increases 100 times.

Keywords:
Linked data, microdata, Apache Spark, feature extraction, big data, network algorithms

CERCS: P170

Suuremahuline tunnusehõive veebiandmetest

Lühikokkuvõte:
Veebiandmed on ajas muutuvad ning viis, kuidas neid esitatakse muutub samuti. Linkandmed on muutnud veebis leiduva info masinloetavaks. Selles töös esitame me kontseptsioonitõenduseks lahenduse, mis võtab veebisorimise andmetest linkandmed ja teostab nende peal tunnusehõivet. Esitletud lahenduse eesmärgiks on luua sisendeid masinõppe mudelite treenimiseks, mida kasutatakse firmade krediidiskoori hindamiseks. Meie näitelahendus keskendub toote linkandmetele. Me proovime ühendada toodete linkandmed, mis esitavad samat toodet aga pärinevad erinevatelt veebilehtedelt. Toodete linkandmed ühendatakse firmadega, mille lehelt tooted pärit on. Informatsioon firmadest ja nende toodetest moodustab graafi, millel me arvutame graafimeetrikuid. Erinevate ajahetketede veebisorimisandmetel arvutatud graafimeetrikud moodustavad ajaseeria, mis näitab graafi muutusi läbi aja. Saadud ajaseeriatel rakendame me tunnushõive arvutamist.

Loodud lahendus on planeeritud suurte andmete jaoks ning ehitatud ja disainitud skaleervust silmas pidades. Me kasutame Apache Sparki, et töödelda suurt hulka andmeid kiiresti ning olla valmis, kui sisendamete hulk suureneb 100 korda.

Võtmesõnad:
Linkandmed, mikroandmed, Apache Spark, tunnusehõive, suurandmed, võrgualgoritmid

CERCS: P170

Contents

List of Figures

List of Tables

1 Introduction

Recent hype in big data has led to new technologies, such as Apache Spark[1] and Hadoop[2], for enabling processing data at Web scale. At the same time, advances in machine learning and advanced analytics have led to new algorithms, which are capable of handling large quantities of data. Data is now advocated by opinion leaders as being the new oil, and companies, that have the capacity to mine and refine it properly, are most likely to dominate current markets. This is why both old economies and emerging markets have turned into building capacities to process and understand big data. Furthermore, the breakthrough in technologies have created new opportunities in data science by enabling advanced analytics on Web data. Web data is widely available compared to domain-specific data, which has been collected over decades and is limited to major market players. However, in order to take full advantage of the opportunities the way of thinking on how advanced analytics models are created needs to be altered as well. For instance, instead of only using "local" features of data entities (features suggested by domain experts), it is possible to use features related to the "relationships" between such entities.

Usage of Web data is challenged by being incomplete, heterogeneous and possibly unreliable. While incompleteness and reliability are mostly related to the availability and usability of Web data, heterogeneity is an issue related to variety in data representation formats, data models and semantic interoperability. In order to handle heterogeneity, several schemas and formats have been proposed for Web data annotations. Such initiative is largely driven by search engine providers, who have started construction of knowledge graphs from annotations in Web data and are using it when displaying search results. This behavior has triggered significant increase in the volume of annotations in Web data. These annotations are often called microdata, markup, rich snippets, etc and collectively they represent linked data, which has Resource Description Framework (RDF) [LS99] as representation model. Such representation provides effective means to tackle heterogeneity from the representation and semantic interoperability perspective. Furthermore, by combining linked data from a variety of Web sources, graphs can be built which represent rich knowledge sources for data mining.

Linked data [BHBL09] is data that is linked together from multiple sources and its purpose is to provide the means to easily join data from multiple sources into one. In 2012, Mika and Potter [MP12] showed that it was used in over 30% of the websites, including the most popular ones like Facebook, Yahoo and Google. Websites differ in quality of linked data they use: some use it only to specify the general information of a web page while others use linked data to join items on a web page into one connected entity. The main issue with linked data is that there is no one defined standard, as such collecting

[1] Apache Spark - `https://spark.apache.org/`

[2] Apache Hadoop - `http://hadoop.apache.org/`

information from different sites is not trivial [ACORH18]. Due to this a challenge arises - how is it possible to compare linked data from different sites. Challenges with using linked data from different websites and joining them to one database are described in Chapter 3.1.

Feature extraction is the process of converting more complex data objects into a set of distinctive features to reduce the dimensionality and redundancy of data and improve generalization. Feature extraction in combination with feature selection (selecting the features to use for training machine learning models) helps reduce the dimensions of input data while not sacrificing performance of machine learning models. In fact, they increase the performance of machine learning models as they remove unneeded information from the data set and this makes the models faster to train and more accurate. Tang et al. [TAL14] identified three main challenges with feature selection: scalability, stability and working with linked data. The authors raised two questions that should be answered in the context of feature extraction from linked data: "how to exploit relations among data instances" and "how to take advantage of these relations for feature selection".

Some effort has been made in extraction of features from linked data in the context of machine learning. However, such studies were limited to extraction of a few features from linked data. Cheng et al. [CKG+11] described a solution that is based on the hierarchical relationships of entities. Their solution requires the user to define queries on data to extract features, which means the number of features is limited by user's knowledge of what is represented in linked data. Another solution for extracting features from linked data was proposed by Mahule and Vyas [MV16]. Their solution used properties defined in linked data and because of this the number of features is limited by the number of properties in linked data.

In this thesis we propose a proof of concept pipeline for extracting thousands of features from knowledge graphs constructed from linked data. In particular, we are interested in extracting company performance related features from Web data to be able to facilitate data driven credit scoring. This thesis was born out of the needs of Inforegister OÜ who has a set of requirements for the outcome. The whole process must be faster than one week for one crawl of Estonian web - a few terabytes (TBs) in size. This is due to the fact that one crawl takes about a week and the results should be obtained before the next crawl finishes. We expect that the solution is able to extract linked data from about 39% of all crawled URLs, since WDC statistics [BMP17] show that this would be an expected result. The solution will be evaluated against the following requirements:

1. Total run time of the solution must be less than one week on input of 5 terabytes of web crawl data (approximately 100 million web pages).

2. The solution must be scalable.

3. The solution must support graphs that are 100 times bigger than the one initially

constructed.

4. The solution must support using a set of feature extraction functions developed by Inforegister OÜ that are introduced in Chapter 4.

The case study is conducted on linked data extracted from Estonian Web and focuses on data about products. The pipeline begins with extracting linked data from web archive files and continues by connecting extracted linked data to Estonian companies. Then product linked data (linked data about products that companies offer) is analyzed to define relationships between companies that offer the same product, so that if two companies offer the same product then they are connected through that product. Next step in the pipeline is using this data to construct a graph that represents the relationships between Estonian companies. Network metrics will be calculated on this graph to generate company relationship related features. Network metrics from different crawls are connected to form time series and features are extracted from these. Every step of the pipeline is constructed with scalability in mind. At each step guidelines will be provided on how to further improve the accuracy and performance of the solution. As a result, the proposed pipeline consists of the following steps:

1. Extract RDF data from Internet archive files.
2. Connect Estonian companies to RDF data.
3. Perform product matching to detect same product offered by different companies.
4. Build Company-Product graphs.
5. Calculate network metrics on built graph.
6. Use a script provided by the company to extract features from calculated network metrics.

The thesis mainly focuses on building the pipeline from start to finish. Specifically, it contributes to the domain by providing an overview of current state-of-the-art and practical implementation of:

- How to resolve the challenges of extracting features from large scale linked data.

Next chapter describes the terminology used in this thesis: we introduce the concept of linked data and its most used formats. We continue by giving an overview of Apache Spark - a framework for working with big data. Then we describe what graph engines are used for working with linked data. Second chapter ends with an overview of feature extraction from linked data. In Chapter 3 we cover previous work in entity linking, interlinking and feature extraction in the context of linked data. In Chapter 4 we describe

the architecture and implementation of the proposed pipeline. We give an overview of what is the goal of each part of the pipeline and how different parts of the pipeline are connected to each other. We give examples of how the data looks like at each step of the pipeline. Chapter 5 gives a detailed overview of the results that we got when we ran the pipeline on a web crawl of Estonian Web data. We validate our pipeline and verify how it would handle bigger input data. Final chapter provides a conclusion of this thesis and discusses potential future research directions in this domain.

2 Background

This chapter gives an overview of linked data, large scale processing and feature extraction from web data. We begin by describing what is linked data and how it is used in modern Internet, we cover most used vocabularies and syntaxes. Then we introduce the concepts of large scale processing and the tools used when data is getting too big for regular processing engines. After this we give an overview of feature extraction.

2.1 Linked Web Data

Linked data is the process of publishing data in a way that data from different sources can be linked together. Bizer et al. [BHBL09] define linked data as data from two sources that have not been connected historically but are now represented as one. Linked data is machine-readable and it is possible to link it to other foreign data sets. Linked open data is linked data that is available to all. One of the most popular linked data formats used on the Internet is Resource Description Framework (RDF) [LS99], a model created by W3C[3]. RDF is also known as metadata.

Linked data is a structured way to present knowledge, it gives information about how data on a page is linked together. This makes information more machine-understandable and allows data from different sites to be linked together. In 2012, Mika and Potter showed that linked data was used in over 30% of the websites [MP12]. It is used by the most visited websites, including Facebook, Yahoo and Google. Figure 1 shows how linked data is used by Google in search results. It provides a quick and informative overview of the search results, and is machine readable. Even though the size of available linked data is constantly growing, there is still a lot of uncovered territory. Even if websites use linked data there is a big discrepancy in the quality of data that is used - some use it to only specify the general information on page while others use it to connect items on their web page into one.

There are several vocabularies to represent linked data - most common ones are schema.org and Open Graph protocol (OGP). RDF is embedded into HTML code using different formats, the most common are Resource Description Framework in Attributes (RDFa) and microdata. There are multiple formats to represent extracted linked data, most popular is the triples representation - a line-based textual format that contains lines with subject, predicate (how object and predicate are related) and object.

Websites use linked data because research has shown that "early publishers of structured eCommerce data benefit more due to structured data being more readily search engine indexable" [ACORH18]. Usage of linked data has grown rapidly since the introduction of Metadata data model in 1999. Different sources put the portion of sites that use it between 21-30 % (2012-2014) [Meu17], 30% [MP12] (2012), according to WDC

[3] https://www.w3.org/

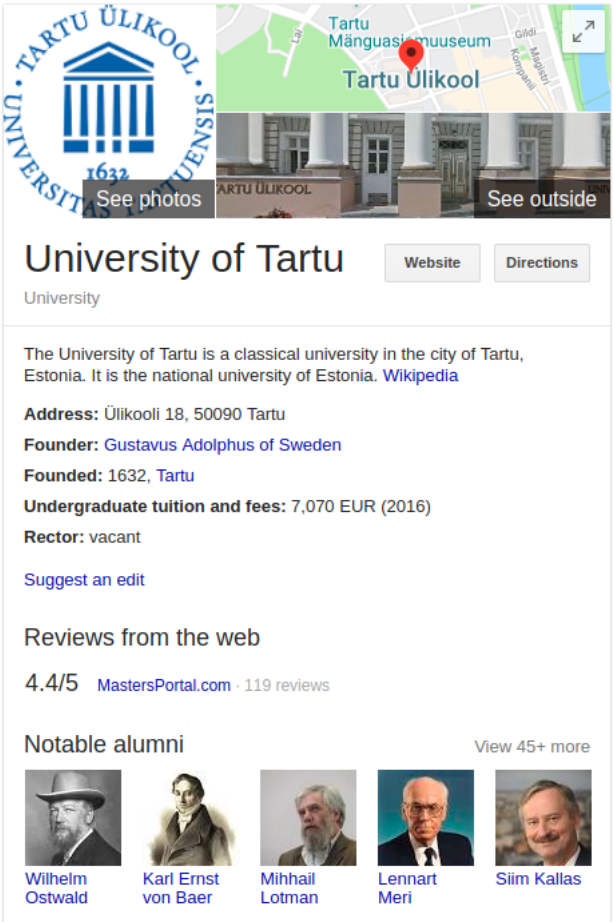
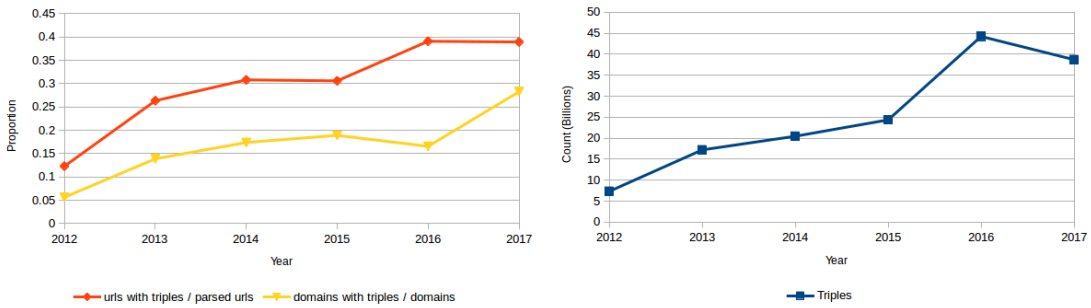


Figure 1. Metadata about University of Tartu from Google search results

it is 39% (2017) [BMP17]. Figure 2a shows how proportion of websites that use lined data increases year by year. Red line shows how number of URLS that contain triples has increased from 12% in 2012 to 39% in 2017. Yellow line shows that in 2017 28% of domains contain structured data, up from 5% in 2012. Figure 2 shows that the number of triples has increased 5 times in 5 years. Slight drop in total number of triples in 2017 is because fewer sites were crawled that year.

2.1.1 Linked Data Formats

There are multiple formats to describe linked data. Two most popular are RDFa and Microdata. Usage of linked data formats in 2017 is shown in figure 3.



(a) Usage of linked data

(b) Count of Triples

Figure 2. Linked data usage according to WDC [BMP17]

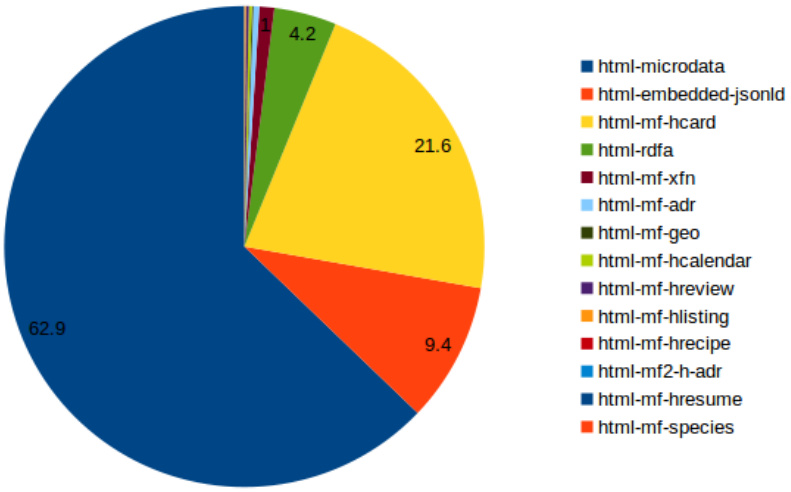


Figure 3. Popularity of linked data formats according to WDC [BMP17]

RDFa. RDFa [ABMP08] was born out of the need to extend HTML with RDF support. RDFa allows the description of vocabulary, prefix, resource, property and typeof attributes that can be used to represent almost all RDF expressions. According to WDC [BMP17], RDFa was the most used format until 2014. Currently it represents only 4.2% of total microdata on the web, as can be seen in figure 3.

In listing 1, a standard plain HTML representation of a product is shown. It is human readable, but hard for computers to understand. Listing 2 shows the same data represented in RDFa. The data is now easily understandable to machines without any additional parsing.

```
<div>
  <div>Apple Macbook Air (2017)</div>
  <div>Mid 2017, 1.8GHz dual core Intel ...</div>
  <div>1000 euros</div>
</div>
```

Listing 1. HTML without linked data

```
<div vocab="http://schema.org/" typeof="Product">
  <div property="brand">Apple</div> <div property="name">Macbook Air
      (2017)</div>
  <div property="description">"Mid 2017, 1.8GHz dual core Intel ..."
      </div>
  <div property="price">1000</div><div property="currency">euro</div
      ></div>
</div>
```

Listing 2. HTML with RDFa

Microdata. Microdata [C+18], also extends HTML with methods to represent linked data. It is, according to WDC, the most used linked data format in 2017, representing 63% of all linked data on the web. Microdata has less attributes than RDFa, and this makes it easer to understand and reduces errors [Meu17]. Other than that, there are no big differences between Microdata and RDFa.

Listing 3 shows information from listing 2 represented using Microdata.

2.1.2 Linked Data Vocabularies

Robert Meusel [Meu17] defined vocabulary, in the context of linked data, as a set of terms that defines a meaning for one or more topics. Two most used vocabularies to represent linked data are schema.org and Open Graph Protocol.

Schema.org microdata is used by search engines to display results. Due to this

```
<div itemscope itemtype="http://schema.org/Product">
  <div itemprop="brand">Apple</div> <div itemprop="name">Macbook Air
      (2017)</div>
  <div itemprop="description">"Mid 2017, 1.8GHz dual core Intel ..."
      </div>
  <div itemprop="price">1000</div><div itemprop="currency">euro</div
      ></div>
</div>
```

Listing 3. HTML with microdata

it is beneficial for web sites to represent data using this vocabulary in order to gain better position in search results. Schema.org is also used by e-mail clients to confirm reservations or bookings [Meu17].

Open Graph Protocol (OGP) is promoted by Facebook. The vocabulary was developed to make it easier to represent objects within a social graph [Meu17]. The goal of OGP is to make the usage of linked data simple for websites, in order to increase adoption. The biggest use case of OGP is the like button outside Facebook [Meu17]. OGP is also used by media who apply it to give article title, description and image, so that it can be easily represented in social media.

2.1.3 Linked Data Syntaxes

Syntaxes are used to store and serialize linked data. These include, but are not limited to, N-triples, JSON-LD and RDF/XML.

N-triples

N-triples [CS14] is a syntax for representing RDF information. One triple consists of subject, predicate and object that are separated by tabs or spaces. A triple ends with a dot. Biggest advantage of N-triples is the simplicity of the formats. Triples also support using prefixes (*@prefix* in the following example) to shorten the representation. It is easy to read and write without additional tools, the format is easily parsable as it is a line-based format. Following is N-triple representation of the object described in listings 2 and 3.

```
@prefix syntax: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

_:laptop <http://schema.org/Product/brand> "Apple" .
_:laptop <http://schema.org/Product/name> "MacBook Air (2017)" .
_:laptop <http://schema.org/Product/description> "Mid 2017, 1.8GHz ..." .
_:laptop <http://schema.org/Product/price> "1000" .
_:laptop <http://schema.org/Product/currency> "euro" .
_:laptop syntax:type <http://schema.org/Product> .
```

JSON-LD

JSON-LD [C$^+$14], is the representation of linked data in JSON format, LD stands for linked data. This representation is designed for use in web applications that already use JSON, converting from JSON to JSON-LD is simple. JSON-LD does not require special storage and can be kept in JSON-based storage solutions. The example from listings 2 and 3 represented in JSON-LD:

```
{
  "@context": "http://schema.org/",
```

```
  "type": "Product",
  "brand": "Apple",
  "name": "MacBook Air (2017)",
  "description": "Mid 2017, 1.8GHz ...",
  "price": "1000",
  "currency": "euro"
}
```

RDF/XML

RDF/XML [BM04] is XML based format for storing linked data. The goal of RDF/XML is to allow usage of linked data in technologies that are based on XML without big changes to the architecture. The example from listings 2 and 3 represented in RDF/XML:

```
<rdf:Description>
    <schema:brand>Apple</schema:brand>
    <schema:name>MacBook Air (2017)</schema:name>
    <schema:description>Mid 2017, 1.8GHz ...</schema:description>
    <schema:price>1000</schema:price>
    <schema:currency>euro</schema:currency>
    <rdf:type rdf:resource="http://schema.org/Product"/>
</rdf:Description>
```

2.2 Large Scale Data Processing

There is no concrete size limit on what is big data, instead it is defined by features of the data. One possible definition of big data was proposed by Laney [Lan01] where he defined it through three V-s: volume (size), velocity (frequency of incoming data) and variety (uncertainty). Standard solutions are too slow to process big data and because of this it requires new frameworks. As it is economically more viable to use many computers, instead of one very powerful one, then most state of the art solutions are built for use in a distributed environment.

2.2.1 Apache Spark

Apache Spark [Spa16] is a framework used for large-scale data processing. Spark processes data in memory, when its main competitor, Apache Hadoop [Had09], uses disk processing. This allows Spark to be many times faster since memory operations are faster than reading and writing to disk, as long as data fits into memory. Apache Spark follows popular map reduce programming paradigm where input is split into parts, on each part a method is ran, and the results are then brought together.

Spark supports both batch processing and streaming. It also supports its own dialect of SQL, called Spark SQL [AXL+15]. Many implementations of machine learning methods can be found at Spark MLlib [MBY+16]. Spark's GraphX [XGFS13] adds extends Spark with graph processing solutions.

Spark uses Resilient Distributed Dataset (RDD) [ZCD+12], a distributed dataset partitioned across executors [4] of the cluster. It is a parallel data structure that is designed specifically for distributed environment. Most Spark operations consists of transforming RDDs. RDD does not provide much information about the data it contains (no row names for example), Spark SQL adds dataframes that hold more information about its contents.

Spark uses Hadoop Distributed File System (HDFS)[B+08] that is, similarly to RDD, built with distribution in mind. HDFS distributes data such that it is easily accessible by multiple executors of the cluster. The main advantage of the file system is that it reduces the time spent on IO, one of the most costly operations in computing. HDFS is also very fault-tolerant and built with low-cost hardware in mind.

The resource management and job scheduling in Apache Hadoop is done using Yet Another Resource Negotiator (YARN) [VMD+13]. YARN allows user to fine tune the cluster for their personal needs, for example applications that are CPU heavy require more CPUs while applications that use a lot of memory need better RAM allocation. YARN distributes the jobs between different machines and makes sure that each step of the process is completed.

Spark can be used from multiple popular programming languages. The framework is built in Scala and due to this, the Scala version of Spark is the most up to date. Java version is not far behind and is well supported as well. Connectors to Python and R do not always have the latest features but are still very useful. The connector of Spark for Python is called Pyspark.

Our designed pipeline must handle multiple terabytes of web data every week. Out of this data hundreds of millions of triples are extracted that must to be processed further. There are also time constraints on the run time. These two requirements mean that a distributed environment for parsing is required. Since our problem, and the triple format, is easy to divide into independently executable tasks then Apache Spark is used for processing triples, building graphs and calculating graph metrics.

2.3 Graph Engines for Linked Data

There are multiple RDF triple graph engines and databases and selecting one to use is difficult. Some engines are built only for Spark and are specialized on RDF data. Others are more general, can be used for other kinds of input data and use Spark only to scale when data gets too big.

[4] Executor - worker in Spark

Distributed Graph Engines

Abdelaziz et al. [AHKK17] compared different RDF engines in built for Apache Spark. According to their results the two most useful are AdPart [HAK+16], a fast processing engine running on Apache Spark, and H2RDF+ [PTK+14], a slower engine running on Hadoop that uses less memory. Both graph engines are built on-top of GraphX. However, all compared by Abdelaziz et al. RDF engines are experimental and their real world use is difficult.

A popular graph engine that supports RDFs and can be used in a distributed environment is Neo4j [Neo]. The application has been used with 3 billion nodes [Agr]. Neo4j uses its own query language, Cypher. The graph engine has a connector to Apache Spark called Neo4j-Spark-Connector that allows to use Apache Spark from Neo4j. For example Spark can be used to read and preprocess data before it is loaded into Neo4j.

Other distributed graph processing frameworks that are not mentioned include Pregel [MAB+10], PowerGraph [GLG+12], Stanford's Graph Processing System [SW13] and Mizan [KAA+13].

Graphframes

Graphframes [DJL+16] is a graph engine built on-top of Spark's API for graphs - GraphX. It simplifies the creation of graphs in Apache Spark by taking care of low level repetitive tasks. Graphframes reduces implementation time since the user does no longer need to define classes for basic elements, such as vertices and edges, and can focus on building graph algorithms. Main reason to use graphframes is that GraphX only supports Scala at the moment but Graphframes gives developer the ability to use GraphX in Java and Python.

Graphframes extends GraphX with dataframes functionality. These new features include "motif finding, DataFrame-based serialization, and highly expressive graph queries" [Gra]. Graphframes also has multiple graph algorithms already implemented. These include simpler ones, such as degree of a vertex and more complex ones, such as finding PageRank.

2.4 Feature Extraction from Linked Data

Feature extraction is the process of building derived values from initial values. Objective of feature extraction is to increase generalization and accuracy of machine learning models. Feature extraction, and feature selection, is the process of selecting only important data and throwing away input that does not give value, that in turn reduces the dimensions of the data. This reduces the training time of machine learning models as input data is smaller and of higher quality, which in turn, increases the accuracy of model outputs as only good features are used.

Tang et al. [TAL14] identified three main challenges with feature selection: scalability, stability and linked data. Scalability is a problem when data gets too big and standard models do not work anymore. For example, model performance will deteriorate when input data does not fit into memory. Stability is a problem with calculations that do not give an exact result but approximate the solution. This is a problem with more stochastic algorithms whose output can change with small changes in input data. Third mentioned issue was working with linked data. Machine learning algorithms have the requirement that input data should be independent and identically distributed (IID). It defines that variables should be mutually independent and come from the same probability distribution. With linked data this requirement is not satisfied and as such, using it goes against the assumptions that many machine learning models are built on. Tang et al. [TAL14] raised two questions regarding the usage of linked data in feature selection: "how to exploit relations among data instances" and "how to take advantage of these relations for feature selection". When developing feature extraction and feature selection solutions on linked data these requirements must be taken into account.

Linked data can be represented as a graph where objects are nodes (vertices) and links between objects are edges. It is easy to understand and describe smaller graphs but for bigger graphs we need network metrics that provide an overview of the graph and help identify more important nodes. These include simple statistics such as degree (number of connections of a node) and shortest path between two nodes. An example of a more advanced graph statistic is PageRank that measures importance of nodes based on nodes that are connected to it. We would also like to know how graphs change over time. For that we can store graph metrics calculated on different time periods in time series - sequence of data points that are ordered according to time.

The number of network metrics that can be extracted from graphs is limited by the number of algorithms that are used. To describe time series specific properties it is possible to derive new time series specific features. For example, it is possible to calculate average, peak, season and entropy of a time series data. These features provide information about the network metrics and how they change in time. They are general and can be run on any time series so it is possible to easily increase the number of extracted features hundreds of times.

In the next chapter we give an overview of related work.

3 Related Work

In this chapter we give an overview of solutions developed to solve similar problems. We first examine entity linking solutions that focused on linked data. Then we describe solutions that use interlinking to connect linked data from two or more datasets. Last we describe solutions that have been developed to extract features from linked data.

3.1 Entity Linking in Linked Data

Standard web data from different sites is not connected to each other. Linked data gives us the opportunity to link together data from different web sites to use the full potential it offers. For example, connecting websites to companies that own them gives us an ability to see connections on company levels, not only on website levels. Linking products between different websites helps us see which two companies offer the same products to help detect competitors. There are two main problems - entity linking (entity disambiguation), to help detect same object from multiple web pages, to use one representation for it, and interlinking (adding new links to data) to link real world companies to objects represented using linked data. Linking products between different websites is entity linking, linking websites to companies is interlinking. When we have linked products and websites then we can build product-company graphs that helps us see how companies are connected through products in order to detect competitors and partners.

Product Entity Linking Using Text

Ghani et al. [GPL+06] used Naive Bayes classifier and semi-supervised co-EM algorithm to perform entity linking on product RDFs. They tested their solution on attribute-value pairs of apparel products extract from web. Their solution had accuracies between 44% and 63% on test set consisting of 76 000 products. Another approach was offered by Van Bezu et al. [vBBR+15], who used different string matching algorithms and hierarchical clustering on title and attribute similarities. Their test set consisted of data from four TV shops, in total 1 629 products. The precision with this method was 45%. Bing product search, described by Kannan et al. [KGAF11], had accuracy of 90%+ on multiple different electronic products. Their constructed pipeline used semantic parsing and tagging, followed by Similarity Feature Vectors to calculate scores. Lastly, binary logistic regression was used for matching. In total they used 20 000 labeled products that belonged to 54 categories.

Product Entity Linking Using Linked Data

Researchers have also looked into using linked data for product entity linking. Petrovski et al [PBB14] used genetic algorithms and achieved an accuracy of 85%. Their pipeline consisted of extracting microdata using Web Data Commons (WDC) extractor[5] that was based on Apache Any23 (described in more detail in Chapter 4.2). Microdata was categorized, and then entity linking was performed using genetic algorithms on linked data extracted from three billion web pages. In total their dataset consisted of 1.9 million products. In order to complement information that was not present in linked data they used NLP methods to extract more properties from title and description. Petrovski et al. discovered that the extraction of brand (manufacturer, e.g. Apple) property was the most accurate, while the Processor (CPU) was the least accurate. The authors said that "markup is mostly not as fine-grained as desirable for applications that aim to integrate data from large numbers of websites" [PBB14].

Using deep learning gives even better results as described by Petar et al. [PPPH16]. They describe how a website for finding data about different companies that offer a product - xploreproducts.com - was created. The authors used deep learning to match and categorize products. Firstly, they used neural networks on HTML microdata. Secondly, recurrent neural networks were used with product images. Best algorithm, random forest, had an accuracy of 88%. Even though the total dataset was small - 110 000 products, 10 000 used for testing- the run time was measured in days.

Categorizing Products to Increase Entity Linking Accuracy

One important step that multiple product matching methods use is categorization before matching. This step was used by Petrovksi et al. and Petar et al.. By categorizing items before matching it is possible to cut down on the number of comparisons needed and speed up the process. Categorization also helps to make the model more precise by focusing on small differences. By only matching products in one category it is possible to use smaller similarities. Products in the same category have similar properties that determine, if two products are the same or different. E.g. it is simpler to compare two washing machines (properties: allowed load, brand) than a washing machine and a phone (properties: camera, brand, storage space). This means that the models are smaller, simpler and faster.

Creating Databases of Products

Product matching techniques have been used to create databases that show companies that offer the same product. These databases can be used to find the cheapest company

[5]WDC microdata extractor `https://subversion.assembla.com/svn/commondata/WDCFramework/trunk/src/main/java/org/webdatacommons/structureddata/processor/WarcProcessor.java`

that offers the product in order to get the best deal. There is no good general database that would link together different products and this is "impeding the creation of a semantically interlinked eCommerce Web" [ACORH18].

Vandic et al. [VvDF12] created a product search platform using RDFa microdata. They did this to overcome the issue that other multi site product search engines lacked - there are multiple standards and connecting several different standards is complicated. One of the many challenges that one has to overcome is heterogeneity of data - languages, currencies, etc.

One approach for creating product databases, used by Bing [KGAF11], was described earlier. The resulting high accuracy was due to the fact that they had some data beforehand. Data from the web was matched with product catalogs, but creating product catalogs is expensive and requires constant updates. Bing product search is, as of April 2018, offline and replaced with product ads in search results.

The site xploreproducts.com, created in Petar et al., is also currently offline.

Conclusion

Matching products is not a trivial task. State of the art systems do get accuracies of 85% but these require product catalogs and fine tuned models whose training times are measured in days. Several systems use linked data for entity linking with good results.

Most solutions that had high accuracy algorithms used categorization. Categorizing also speeds up the process as it is not required to compare products that are from different categories. State of the art systems use multiple NLP methods, including word embeddings, tagging, parsing. Other methods that offer good results are getting additional data from title and description, using neural networks, logistic regression and genetic algorithms. There have been several attempts at creating databases that allow the users to compare different providers of the same product.

The solutions described above are state of the art complex systems with long implementation and run times. We did not use them in our work but they have provided us with guidelines on what to consider when performing entity linking. They have also given multiple directions on how to improve entity linking in future work.

3.2 Interlinking in Linked Data

When we have data in separate databases and it is not connected then we cannot use the full potential that the data offers. For example, if we join data about products from multiple web shops that sell a laptop we want, then we can easily see who sells it for the lowest price. Interlinking is the process of joining data from two or more databases in order to use knowledge from multiple sources.

Bizer et al. [BHAR07] used interlinking to connect eight big linked data datasets into one connected entity. The datasets contained several billion triples that were connected

using 150 000 RDF links, the links connect subject URI of one RDF and object URI of another RDF. Links between different companies are shown in figure 4. For example, in the presented dataset user can choose a Semantic Web Browser (a browser, or an extension to existing browser, that helps discover linked data) to discover how data is connected by navigating from an author to his/her book and from the book to the reviews.

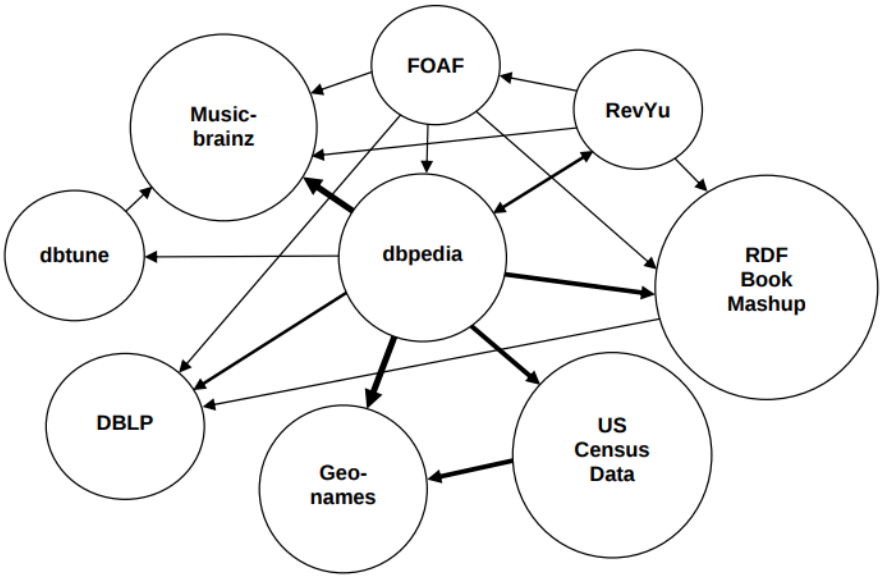


Figure 4. Interlinked RDF knowledge bases [BHAR07]

Graph presented in figure 4 is from year 2007. Since then, a lot more new data has been made available and interlinked. The size of Linked Open Data Cloud [CJ18] in 2011, when there were 295 datasets, was 24 billion triples [SBP14]. Currently it connects 1 184 datasets. State of linked open data cloud in April 2018 is shown in figure 5.

Scharffe, Liu and Zhou [SLZ09] created a tool for managing the integration of RDF datasets. Datasets are first processed - all literals translated to English and names converted to a common format. This is followed by optional filtering based on some RDF tag to reduce the number of elements further down the pipeline. After that, matching is performed where linked data objects are joined based on the similarities calculated on their properties. Next step is interlinking where "sameAs" tags are created between linked data objects that are deemed the same. Last step in the presented solution is fusion, where two datasets are joined according to "sameAs" tags. Fusion complements the objects that are joined, for example if one object had price and other object had offer, then object in joined database has price from first and offer from other. As a downside their application requires user input at each step.

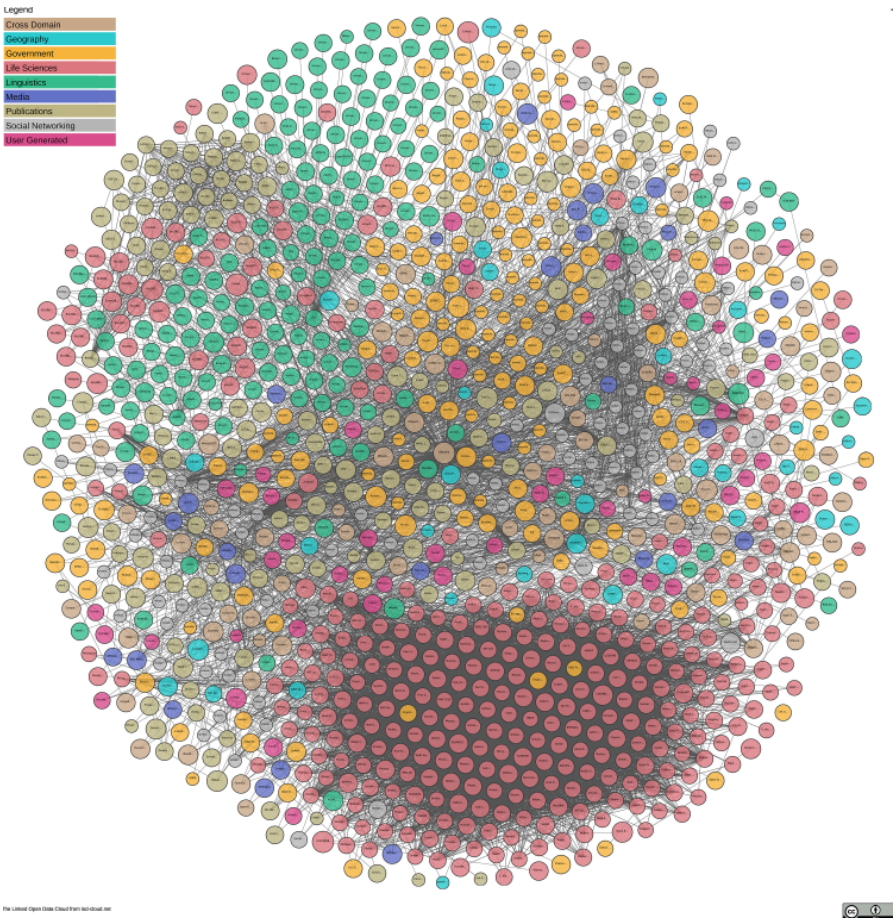


Figure 5. State of Linked Open Data Cloud in April 2018 [CJ18]

Our solution uses interlinking to join data from our knowledge base of URLs and companies to a knowledge base of triples extracted from web archive files. We use a method for creating links that is similar to the one suggested by Bizer et al. [BHAR07] - we take URL form the subject of the triple and connect it to company URL. What sets our solution apart from the tool created by Scharffe et al. [SLZ09] is that our solution does not require human input at any step. However, our solution uses a simpler logic for creating link than the one they presented.

3.3 Feature Extraction from Linked Data

Linked data provides connections between entities in the data. Some effort has been made in extraction of features from linked data in the context of machine learning. However, such studies were limited to extraction of a few features from linked data.

Cheng et al. [CKG+11] described a scalable solution for extracting features from linked data that is based on the class-subclass hierarchies of entities. Their constructed solution used "isA", "type" and "subClass" relationships found from linked data, which are used to automatically extract features from input data. The hierarchical data was used to create feature vectors for input to machine learning models. In doing that the authors applied *restricted entity domains* concept to make sure only similar objects are compared. *Restricted entity domains* assure that vectors created from similar domains are compared to each other, for example movies and products are not compared to each other - this way only entities that are relevant for given task are used. Features were extracted using SPARQL (a RDF query language) [HBF09] that was extended with NAGA [KSI+08] that adds regular expression capabilities. User still needs to write SPARQL queries, even though the solution was described as automatic. This means that the number of features is limited by user's knowledge of input data, SPARKQL features and domain knowledge.

Mahule and Vyas [MV16] proposed another solution for extracting features from linked data that uses neighbours of a linked data objects as features. Vectors created in this approach are very limited as the creation process only uses properties of linked data objects that are limited in nature. However, the algorithm takes advantage of the sameAs relationship - instead of creating features for all objects, it reduces the execution time by using the features of sameAs object; sameAs is also used to link data from different sources together. Their system is not dynamic - it requires the definition of features that are extracted, meaning for every new linked data type, they need to define new features that are extracted.

Our proposed solution uses similar relationships, "type" and "sameAs". We also use *restricted entity domains* by separating linked data objects by type. Our solution extracts network metrics from constructed knowledge and calculates their derivatives, while Cheng et al. [CKG+11] used SPARQL queries, that are similar in that they also work on graphs but writing SPARQL queries to get more advanced metrics, such as PageRank is complex, if not impossible. The solution proposed by Mahule and Vyas [MV16] requires input from the user between parts of the their pipeline, while our proposed solution does not.

4 Implementation of Feature Extraction Pipeline

This chapter begins by describing the overall architecture and how different steps of the pipeline are connected to each other. Next, we describe the technical implementation of each step of the pipeline. We give an overview of tools and technologies used and what the result of each step looks like. Full source code of the solution is presented in Appendix III.

All the challenges of feature extraction, mentioned in Chapter 2.4, must be addressed in design and implementation. The three main challenges were scalability, stability and working with linked data. Since we are working with linked data we have to keep in mind that the extracted features are not independent and identically distributed (IID). The two questions raised by Tang et al. [TAL14], "how to exploit relations among data instances" and "how to take advantage of these relations for feature selection", must be taken into account.

4.1 Architecture of Pipeline

Here we describe the design of the proposed pipeline. The pipeline begins with extracting linked data from WARC files and continues with connecting companies to triples to form a graph. Next step is calculating network metrics on the graph, followed by collecting and connecting metrics from different snapshots to form time series. Finally, calculating derivatives on time series data is described. Full pipeline is illustrated in figure 6. In the figure we have mentioned if a step uses Spark.

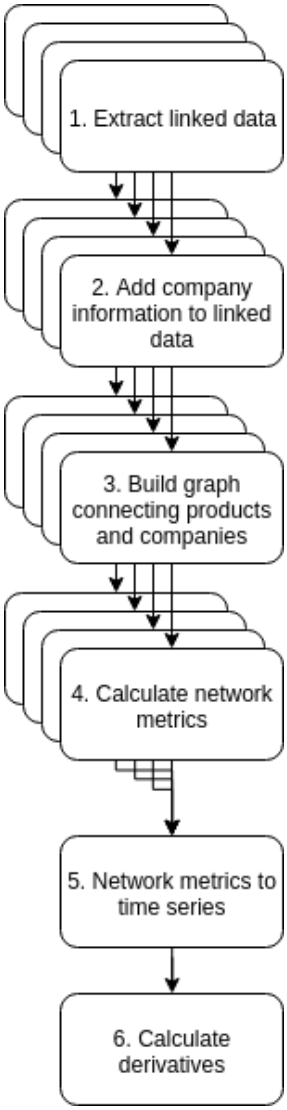
Extracting Linked Data

The whole pipeline begins with linked data from web archive files. Linked data in web sites is represented in several formats and vocabularies. The goal of this step is to take linked data and output it in one common syntax - N-triples. The triple extraction process generates a random ID for each extracted triple. After this step we have linked data from websites that is represented in a unified way, so processing it further is easy.

Adding Company Information to Linked Data

As an input for this step we have linked data triples extracted from Web archive files. These triples represent information from one web site and are not connected to other web pages (only in a very rare cases). However, the first part of a triple - the subject - defines the web page where it was mined from.

We want to analyze triples in the context of data driven credit scoring. This means that we need to know what company a triple is connected to. To do this, we create new links that show relationship between a triple and the company through the URL of the



1. Extract linked data from WARC files (uses Spark)
2. Add company information to linked data (uses Spark)
3. Construct bipartite graphs that link products and companies (uses Spark)
4. Calculate network metrics on bipartite graphs (uses Spark)
5. Join network metrics from multiple snapshots into time series values
6. Calculate derivatives on time series values (uses Spark)

Figure 6. Overview of pipeline. Multiple boxes shows that this is done for every snapshot. Last two steps combine data from multiple snapshots.

triple. The URL is connected to a company by using a list of companies and the URLs that are related to it. After this step the we have triples that are supplemented with links to companies. This gives us the ability to see triples in a bigger context than only the web page that they are from.

We continue with implementation details of every part of the pipeline.

Constructing Graphs from Linked Data

We want to analyze relationships between companies - to do this we need a way to connect two or more companies. Due to technical limitations of extraction of linked data there is no general ID that can be used to make sure if two objects are the same or different. What this means that if we ran the extraction on the same file twice, then the triple IDs that we get are all different. If we only connect triples to companies and build graphs from this information then we would get star graphs - graphs that have a company and the triples connected to it but no connections between companies.

In order to solve this issue we need to connect linked data objects that represent the same entity. In the scope of this thesis we chose to use product microdata, since it is well represented in the web crawl and it can be used to link companies. Product microdata also offers a good way to link two products that are the same together - SKU or stock keeping unit. Products, like many other microdata objects, also have microdata sameAs tags that is used to represent that one object is the same as another object. Both SKU and sameAs tags will be used to represent products that are the same.

Once we have linked products that are the same together, we can create graphs that represent relationships between companies through products. In the graph two companies that offer the same product are connected through this product, so we have a more informative graph than before.

Network Metrics of Graph

Once the graphs are built we can continue with the actual feature extraction process. In this part of the pipeline we will perform the first step of feature extraction by calculating different network metrics on the graph created in previous step.

Network metrics give us a statistical overview of the graph. They describe companies based on their relationships to products and other companies. More advanced metrics also help us determine which companies are more important based on the connections in the graph.

Time Series

Network metrics on one graph tell only part of the story. In order to see how relationships between companies change over time, we need to calculate network metrics on many web crawls. One way to represent changes in time is to store the values in time series - data ordered in time. Time series show us how data changes over time.

After this step we will have time series values that describe how network metrics change over web crawls.

Feature Extraction

Last part of the pipeline is to use a script provided by Inforegister OÜ to calculate derivatives of time series. The derivatives are used, not in the scope of this thesis, to train credit scoring machine learning models. The script uses Apache Spark to speed up the computation. It is written in Python that acts as a wrapper for R - essentially it uses Python to call R functions and Apache Spark to parallelize the computation.

This script is used to increase the number of extracted features. It increases the number of features 10 times for every network metric time series by calculating new time series specific features. The features calculated by the script range from basic simple statistics, such as minimum value, maximum value, average value, to more complex statistics that describe time series - peak, season, entropy.

4.2 Extracting Linked Data

Linked data extraction is based on the work of Khalil Ur Rehman that is described in Microdata Deduplication with Spark [RK16]. First step of Rehman's thesis is extracting microdata from Web ARChive (WARC) files (used to store results of web crawling). We made his job more Spark-like: less writing to disk, more work in memory. For example, we wanted to use WARC files directly in Spark and for that we used a WARC Utilities[6] connector. The tool is part of WARC utilities, provided by SURFsara [7].

We also improved exception handling to increase the number of extracted triples. Another notable change is the addition of tools to fix input and to guarantee correct output. Some HTML and XML files are broken (they do not contain closing tags or they have unescaped quotation marks) and due to this less triples are extracted from them. We used SAXParser[8] to validate and correct HTML and XML files. All in all, these fixes increased the amount of extracted triples by 49%.

One of the most used extractors for mining linked data from websites is Apache Any23 - "Anything to Triples". The tool has been used in several articles about using linked data, including [Meu17] and [PBB14]. It is a Java library that provides methods for extracting linked data from HTML strings. All most popular vocabularies and RDF representations are supported. Any23 can extract all linked data as long as it is correctly represented, it does not check the correctness of surrounding HTML [Meu17]. Any23 uses multiple popular libraries, such as TagSoupParser[9] for reading and parsing HTML as it is found on the Internet (broken) and Apache Tika[10] for detecting the content type of files.

[6] WARC Utilities - `https://github.com/norvigaward/warcutils`

[7] SURFsara - `https://userinfo.surfsara.nl/`

[8] Java API for XML -`https://docs.oracle.com/javase/tutorial/jaxp/sax/index.html`

[9] TagSoupParser - `http://vrici.lojban.org/~cowan/XML/tagsoup/tagsoup.pdf`

[10] Apache Tika - `https://tika.apache.org/`

Linked data is extracted using Any23 version 1.1, however, it is planned to upgrade to version 2 as it supports more extractors. Version 1.1 has 26 extractors, 2.1 supports 32. We carried out tests comparing the two versions and saw that version 2 extracts more triples from the same file. Using an older version is not a blocking issue as for this thesis only one extractor is needed - "html-microdata" extractor. An issue with Any23 is that it can output triples that are not to standard. For example Neo4j did not accept triples extracted by Any23 as it included broken triples. We used Eclipse RDF4J[11] Rio triple parser to make sure that all extracted triples are correct and to standard.

HTML from web pages is often broken - missing closing tags, unescaped strings in object names, etc. It is possible to use different tools to fix HTML and increase the the number of extracted triples. One such tool is SAXParser that reads in HTML content and tries to fix broken tags. Any23 also extracts triples that are not according to standards that other solutions require. This means that oftentimes it is not possible to use extracted triples and preprocessing is required. We use Eclipse RDF4J to make sure that all extracted triples conform to stronger standards. RDF4J's Rio triple parser will take as input all triples and notifies when a triple is not according to the standard.

In the following example we use *@prefix* to shorten the triples to make the demonstration easier to read and fit into one line. This shortening was not used in our solution, but it is often used when working with triples. One defined is *catalog:* <*http://www.somesite.com/catalog/*>, and it is used in the example as *catalog:product*. This is resolved as <http://www.somesite.com/catalog/product>. Example of a triple outputted in this step that shows a product mined from somesite.com/catalog/product with SKU value "product123", represented as triple with ID *_:productNode*:

```
@prefix catalog: <http://www.somesite.com/catalog/> .
@prefix microdata: <http://www.w3.org/1999/xhtml/microdata#>
@prefix syntax: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

catalog:product microdata:item _:productNode .
_:productNode syntax:type <http://schema.org/Product> .
_:productNode <http://schema.org/Product/sku> "product123" .
```

4.3 Adding Company Information to Linked Data

Next step in the pipeline is connecting Estonian companies to extracted triples - interlinking. For this a list connecting URLs to company codes, provided by Inforegister OÜ, is used. It is worth noting that an URL contains scheme (http, https, ftp, etc), server name, domain name and path, while query and fragment are optional.

```
scheme://www.server.domain.com/path0/path1/../pathN?query=search#row=1
```

[11] http://rdf4j.org/

The issue in matching URLs is that some companies have their own domain, while others use a hosting provider. The URLs for companies that use hosting providers are long and similar to others from the same hosting provider: companies that use Hot[12] all have the same host (and domain) in their URL but differ in path while companies that use Edicy[13] (site-name.edicypages.com) differ in server name. This means that matching company URL to a URL from WARC is not as simple as it might seem at first.

URLs from WARC files are often long and contain information that is redundant in matching against companies, such as the query and fragment parts of the URL. They often contain paths that are longer then needed. When company site is hosted in a hosting site, such as Edicy, then server name is also important. The main challenge in this part is avoiding false positives: making sure that "pihlakas.ee" and "kas.ee" are not determined to be the same site or "pood.ee" and "pood.ross.ee" or "kukke.edicypages.com" and "kadriine.edicypages.com". But at the same time we must say that "rossgate.ross.ee" and "pood.ross.ee" are from the same company.

In linking triples to companies every triple that has an URL in the first position (subject) is checked against the list of company codes and URLs. Scheme (http and https) and "www" are removed from the URLs. At first level of matching we checked if URL from RDF is inside company URL. There is also a second level matching that checks domain match, because the first level creates a lot of false positives. There was also an idea to match both server and domain but this would ignore bigger sites that span multiple servers and have URLs such as "shop.domain.com" or "booking.domain.com". There are multiple triples that have the same URL in subject, to reduce the number of triples we discard duplicates and return only distinct (unique) triple company pairs. Our proposed solution offers good performance and accuracy. To further increase the accuracy, it would be needed to manually define all exceptions.

For example, in matching "domain.com" and "server.domain.com" the first level requires only partial match: server.domain.com/path. Second level requires full match: domain.com. Example matching "main.com" and "server.domain.com". First level is a match as string "domain" includes string "main": server.domain.com/path. However, second level does not match: domain.com.

This part requires simple line-by-line text processing. We chose to use Spark to speed up the process in this step as the solution is easily dividable into small subtasks that can be independently processed. Since we are working with text processing, we chose Python as it provides great text processing tools. As this step requires both Spark and Python, it is written PySpark, Python's connector to use Spark.

After this step a new triple is created for every triple that is related to a company. An example of the output at this step showing how triple of "_:productNode" is connected to company with code "ee-12345678":

[12] Currently offline, once popular Estonian hosting and web mail site

[13] `http://www.edicy.com/`

```
@prefix catalog: <http://www.somesite.com/catalog/> .
@prefix microdata: <http://www.w3.org/1999/xhtml/microdata#>
@prefix media: <http://graph.ir.ee/media/> .
@prefix orgs: <https://graph.ir.ee/organizations/>.

catalog:product microdata:item _:productNode .
catalog:product media:usedBy orgs:ee-12345678 .
```

4.4 Constructing Bipartite Graphs of Products and Companies

After previous step we have data about companies and products. A good way to present this information is through a bipartite graph - a graph that can be divided into two (companies and products) so that an edge connects a vertex from one group (companies) to a vertex in another group (product). In this step the data is combined and filtered so that only connected companies and products remain.

Several specialized RDF engines were considered for this step. However, there were multiple issues in getting the solutions built on Spark running on local machine and even more issues were encountered trying to run the solutions in the cluster. Neo4j was easier to set up and had good support for triples. However, RDF import methods of Neo4j required triples that matched the strong RDF standards. This problem was solved by using triple fixing solutions in the triple extraction part. Another issue with Neo4j was the Spark connector - it does support loading data from Neo4j to Spark and vice versa but but it is time-consuming. In this step we decided to use graphframes in order to keep data conversion costs down - there is no conversion cost since input and output of graphframes graphs are native Spark applications.

As no RDF engine is used this is done in Python using PySpark and Spark SQL. Essentially the step consists of creating three new dataframes: one of companies, another of all microdata#items and one of all products.

Dataframe about companies uses data from previous step to connect RDFs with an URL to a company code. RDFs with URLs that do not have a company code are not used. Dataframe about microdata#item tags contains information about what nodeID is connected to what URL. Product dataframe contains information about all products in the input set. It contains information about product, such as name, description and sameAs tag.

Connecting these three dataframes results in input to the bipartite graph. First column contains company codes and second column contains product references. An example of data at this point, that describes how company "ee-12345678" is connected to "_:productNode":

```
<https://graph.ir.ee/organizations/ee-12345678> _:productNode
```

Dataframe about company codes and product references is used to construct a bipartite

graph using graphframes. Firstly, the two columns are joined into one to get the vertices of the graph. Next, edges between from companies to products are built. Edges are represented in the dataframe shown above, first column is source and second column is destination. We created another edge from product to company because several graph algorithms require undirected edges, but graphframes supports only directed edges. In the end of this step a dataframe that represents the graph is built. Example row from the graph, showing edge "minedFrom" from that connects company "ee-12345678" to product "_:productNode":

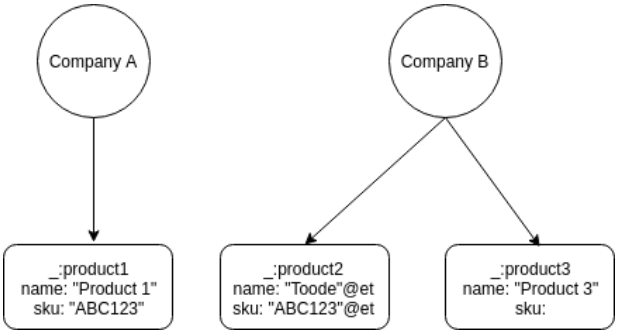
```
+-----------------------------------------------+-------------+------------+
|source                                         |destination  |relationship|
+-----------------------------------------------+-------------+------------+
|<https://graph.ir.ee/organizations/ee-12345678>|_:productNode|minedFrom   |
```

Product Matching Using SKU

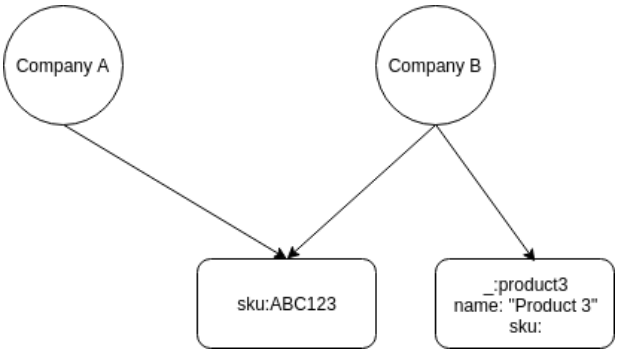
Triple extraction output represents each product using a randomly generated ID. This means that products that are the same are not represented using the same ID. Due to this, the graph we are building, product-company graph, is full of stars - companies and their products - but no company is connected to another company. The goal of this subtask is to create a better understanding of relationships between companies by linking products that are the same together - entity linking. Products are linked using their SKU and sameAs values. This step is done as a part of the bipartite graph building step.

Entity linking of products is demonstrated in figure 7. Figure 7a, illustrates a situation with two companies and three products. Products 1 and 2 share SKU but not name, product 3 does not have an SKU and the name is different than that of the other products. The products are then matched based on SKU and the result is shown on figure 7b. Products 1 and 2 are represented as one product but product 3 is a separate entity. It is also visible that during matching the SKU was cleaned.

Tables 1, 2, 3 and 4 show the challenges that arise in entity linking using microdata. These tables show partial microdata for the exact same product from four different websites. Tables are constructed mostly form schema.org/Product microdata, but many times it is supplemented by other schema.org elements and even OGP metadata. There exists a big discrepancy in the microdata that different sites offer: products that are physically the same are not similarly represented. Out of the 4 products from different sites, none share the name. Every name does contain "MacBook Air", but it could stand for many different MacBook Air models. Another possible matching item - Stock Keeping Unit (SKU) - is present in 3 tables, and matches in tables 1 and 4, while it has multiple values in table 3. Only tables 1 and 2 have Brand value (missing brand fields can be extracted from titles and descriptions, according to Petrovski et al. [PBB14]). Another filed that could be used for entity linking is the description. Three of the four



(a) Companies that share a product, before SKU matching



(b) Companies that share a product, after SKU matching

Figure 7. Product Matching Using SKU

products have description in microdata and one has it in OGP metadata. However, the descriptions for two items are in Estonian and two descriptions are in English. These examples show that there is no ideal way to perform entity linking but it suggests that Stock Keeping Unit can be used as it is one of the most well defined property available.

One issue that arises here is that some products use multiple SKUs. One example of this is shown in Table 3, where multiple SKUs are defined for one product: "53211", "Boonuskontole 8,36", "13", "Kuumakse alates: 31,66". In such case only the first SKU is used as, in this and similar cases, the first SKU was the correct one. However, a method for detecting the correct SKU would be favorable. Another issue encountered here is that sometimes a product has one SKU field that contains multiple SKU values, for example ’"EON3430AOX / EHA6041XOK"@et-ee’. In these cases different SKUs were often separated using "/". An approach that provided the best result was to split the string from " / " (notice the white spaces) and then use the last value. This does not split SKUs that already contain a slash, such as "MQD32ZE/A", demonstrated in table 1.

In this step another dataframe is built that contains all products that have an SKU.

Table 1. Microdata about MacBook Air 2017 from Euronics

Name	Sülearvuti Apple MacBook Air (2017) / 128 GB, ENG, MQD32ZE/A
SKU	MQD32ZE/A
Price	9.5999E2 [from offer]
Currency	EUR [from offer]
Brand	Apple [from brand]
Product URL	www.euronics.ee/t/80489/tahvelarvutid-ja-arvutid/sulearvuti-apple-macbook-air-(2017)-128-gb-eng/mqd32ze-a
Description	Mis iganes ülesanne sind ees ootab, viienda põlvkonna Intel Core i5 protsessor koos Intel HD Graphics 6000 graafikaprotsessoriga saavad sellega hakkama. Kõik töötab ülikiiresti – alates fotode töötlemisest kuni veebi sirvimiseni.

Table 2. Microdata about MacBook Air 2017 from Õunaturg

Name	MacBook Air
SKU	-
Price	650
Currency	EUR
Brand	Apple
Product URL	-
Description	Apple MacBook Air (2017) / 128 GB, SWE Tootega tuleb kaasa kõik, mis oli algselt pakendis! +10 kuuline garantii euronicsi poolt.Töötab ilma ühegi veata! Täpsema info saamiseks palun helistage: [hidden] või kirjutage : [hidden] Saadan üle Eesti, kulud katan mina! Hind on lõplik ning ei muutu :)

Table 3. Microdata about MacBook Air 2017 from iShop

Name	Apple MacBook Air 13ï5 1.8GHz 8GB 128GB SSD MQD32
SKU	Multiple values: 53211, Boonuskontole: 8,36, 13 , Kuumakse alates: 31,66
Price	928.996
Currency	EUR
Brand	-
Product URL	http://ishop.ee/et/macbook-air/2407-apple-macbook-air-13-i5-18ghz-8gb-128gb-ssd-mqd32.html
Description	1.8GHz dual-core Intel Core i5 processor, Turbo Boost up to 2.9GHz 8GB 1600MHz LPDDR3 memory 128GB SSD storage Intel HD Graphics 6000 Accessory Kit [from OGP metadata - the page description]

Table 4. Microdata about MacBook Air 2017 from iDream

Name	MacBook Air 13 (2017)
SKU	MQD32ZE/A
Price	879
Currency	EUR [from offer]
Brand	-
Product URL	-
Description	Mid 2017, 1.8GHz dual-core Intel Core i5 protsessor, 8GB of 1600MHz LPDDR3 operatiivmälu, 128GB flash mälu andmete salvestamiseks, 13Ï440x900 läikiv LED ekraan, Intel HD Graphics 6000 integreeritud graafikakaart, Integreeritud FaceTime kaamera, Thunderbolt 2 väljund, Kaks USB 3.0 pesa, SDXC kaardipesa, Aku kestvus kuni 12h, Integreeritud kõlarid ja mikrofon, 802.11ac Wi-Fi (IEEE 802.11a/b/g/n), Bluetooth 4.0, Apple 45W MagSafe 2 Power Adapter, Mac OS X 10.12, Laius 32,5 cm, Kõrgus 0,3 kuni 1,7 cm, Sügavus 22,7 cm, Kaal 1,35 kg, Komplektis: 13-inch MacBook Air, 45W MagSafe 2 power adapter, Dokumentatsioon

SKU values are first cleaned: the language code, "@et-ee", "@et", "@fi", etc is removed. Additionally, quotation marks are stripped from the beginning and the end. A new triple of product code, sameAs relationship, and product's cleaned SKU is created. The SKU is represented with "sku:" prefix, such as "sku:product123". In addition to that, all previous sameAs triples that were already included in the data are used in this step.

The sameAS dataframe created in this step is used when creating bipartite graphs. When a product has sameAs tag, then its value used in bipartite graphs. Due to this, products with same SKU are represented using the same vertex in bipartite graph.

4.5 Calculating Network Metrics on Bipartite Graphs

In this step network metrics are calculated on the previously created bipartite graph.

Graphframes, framework that was used to build bipartite graphs, already has a few network metrics algorithms implemented. These include simpler algorithms, such as degree, and more advanced ones, such as PageRank [PBMW99]. Using these algorithms on the graph is straightforward.

However, the number of available graph algorithms, that can be used in the context of this thesis, is limited. For these situations graphframes supports creating custom algorithms. To demonstrate that it is possible, and to evaluate how difficult it is, one custom algorithm was implemented.

We implemented Average Nearest Neighbour Degree (AAND) graph algorithm that compares nodes that have the same degree. When AAND is small but the nearest neighbour degree is high, then this node is part of a more complex neighborhood than other nodes with the same degree. In our case the degree of a company node shows how many products this company offers - higher degree means more products. Nearest neighbour degree shows how well connected a company's products are - when company's nearest neighbour degree is one then we have a star graph (one node in the middle, products connected to the node but products are not connected to other companies). AAND can be used to compare companies that offer similar number of products. When AAND is small but nearest neighbour degree is big, then this means that the products of a company are connected to more companies than other companies that offer the same number of products.

Implementation of AAND has three steps. First, degrees are calculated for every node. Secondly, average neighbour degrees are calculated for every node. In the last step all vertices with the same degree are grouped together and the average degree of their neighbours is calculated.

The algorithm is implemented using graphframes' *AggregateMessages* method, shown on listing 4. *AggregateMessages* works by sending and receiving messages in the graph. The user has to define for every vertex what is the message it wants to send to its neighbours (*sentToDst*), what is the message it wants to send to vertices that send a message to it (*sendToSrc*) and what to do with the messages it receives. Lines 6-8 of the

example calculate the degree with already defined method. Lines 11 and 12 define what messages should be sent to the sender (source) and destination. In the implementation of this algorithm nothing needs to be sent to the sender (source). Line 12 defines that every vertex sends its degree to all the vertices it is connected to. Lines 15-18 show how *aggregateMessages* is called on graph that contains already calculated degrees to calculate nearest neighbour degree. Line 16 defines that vertex should call Pyspark SQL's average function on the messages it receives and that this is called "nearest neighbour degree" in output dataframe. Lines 21-27 calculate average nearest neighbour degree - firstly nodes are grouped together based on degree, then average nearest neighbour degree is calculated for every group.

```
1  from pyspark.sql import functions as F
2  import graphframes
3  from graphframes.lib import AggregateMessages as AM
4
5  # find the degree of nodes
6  g = graphframes.GraphFrame(nodes, edges)
7  degree_dataframe = g.degrees
8  degree_graph = graphframes.GraphFrame(degree_dataframe, edges)
9
10 # define messages that are sent to source and destination
11 msgToSrc = None
12 msgToDst = AM.src["degree"]
13
14 # calculate Nearest Neigbour Degree
15 nnd = degree_graph.aggregateMessages(
16     F.avg(AM.msg).alias("nearest neighbour degree"),
17     sendToSrc=msgToSrc,
18     sendToDst=msgToDst)
19
20 # calculate Average Nearest Neigbour Degree
21 annd = nnd
22   .groupBy(nnd.degree).avg("nearest neighbour degree")
23     .select(
24       F.col("id"),
25       F.col("nearest neighbour degree"),
26       F.col("avg(nearest neighbour degree)").alias("ANND")
27     )
```

Listing 4. Average nearest neighbour degree in Graphframes

In this step we extract four metrics from the graph: degree, PageRank, nearest neighbour degree and average nearest neighbour degree. When all metrics are built we can convert the results data into time series.

4.6 Building Time Series of Network Metrics

We want to see how the values of network metrics change over time as this helps us see how relationships between companies develop over time. The network metrics, calculated on each snapshot[14] of the Estonian web, are joined together to from time series that shows changes in network metrics over snapshots.

At this step, network metrics from each snapshot are joined together. The result is a dataframe that has company IDs as row indexes, column names are metric name followed by the snapshot number. For example, degree value calculated on first snapshot is "degree01", on second snapshot it is "degree02" and so forth. Example of the dataframe at this point is visible in table 5.

Table 5. Network metrics as time series

ID	degree01	degree02	pagerank01	pagerank02
Company 1	12	2	0.02	0.01
Company 2	310	310	0.35	0.36

This is the only part of the pipeline that is running on local system and does not use Spark. Using Spark at this step is not needed as the data is small and Spark would create runtime and maintenance overhead. Expected dataframe size for this step is a 40 000 rows and a few hundred columns, something Pandas and other popular Python frameworks can easily handle.

Currently we have network metrics from only one snapshot but for the next task we need more. At this step we generate synthetic data using Python's (pseudo) random number generator to simulate a situation where we have data from 52 snapshots. Every time series is filled with random numbers so that they would contain 52 values in total, to represent one year.

4.7 Calculating Derivatives on Time Series

Credit scoring machine learning models need more input than only the values of network metrics, since more inputs help models see more relationships in data. In previous steps we already joined network metrics together to form time series. To increase the number of inputs, and to provide time series based derivatives, for machine learning models feature extraction is performed in this step. This is done by calculating derivatives of network metrics time series.

Derivatives are calculated using a script by Jüri Kuusik [KK17]. The algorithm creates derivatives that range from simpler statistics, such as mean, median, maximum, minimum, to more advanced derivatives, such as statistics that describes time series

[14]Snapshot - Information crawled from websites at some point in time

parameters (peak, season, entropy). The derivatives are calculated for every time series and in total 460 new derivatives per time series are created on one year's worth of network metrics. In the context of this thesis, this means that 1 840 derivatives are created for every company (4 metrics times 460 derivatives).

Table 6. Example of derivatives calculated on the network metrics time series

ID	Company 1	...	Company 35 000
SEASON_degree	20.42	...	1.30
QREG_LONG_ACL_LAG_1_degree	0.003	...	0.01
...	...	...	...
QREG_LONG_ACL_LAG_48_degree	0.82	...	0.84
...	...	...	...
SEASON_PageRank	0.01	...	0.12
QREG_LONG_ACL_LAG_1_PageRank	-0.54	...	0.12
...	...	...	...
QREG_LONG_ACL_LAG_48_PageRank	0.04	...	0.76
...	...	...	...

An example of result after this step is shown in table 5. The table is transposed to fit it into this page, in real results the companies are in rows and derivatives in columns. In the table we show a few examples of what derivatives are calculated: we demonstrate season and quadratic regression. Derivatives are calculated for every company and on every time series: table shows degree and PageRank.

4.8 Discussion

All the challenges of feature extraction, mentioned in Chapter 2.4, were addressed when designing this pipeline. The three main challenges were scalability, stability and working with linked data. Since the whole pipeline is designed with scalability in mind, then this issue can be crossed off. The issue of stability remains as the output of PageRank can (slightly) change based on the order of nodes in the calculation. What is an even bigger threat is that dropping a bridge[15] can drastically change some network metrics.

Since we are working with linked data we have to keep in mind that the extracted features are not independent and identically distributed (IID). The two questions raised by Tang et al. [TAL14], "how to exploit relations among data instances" and "how to take advantage of these relations for feature selection", are addressed. Connecting companies to products is fully built on exploiting the relations between data instances and most of the network metrics are calculated on relations extracted from linked data.

[15]Graph theory: an edge whose removal increases the number of connected components.

5 Experimental Results

In this chapter we describe the experimental testing that we performed in order to validate the proposed pipeline. We also analyze the results of the graph created in the pipeline. At first we give an overview of the cluster where the implementation was run and where we performed computational experiments. We then describe the input data in Dataset Description section. Next, we give an overview of results of triple extraction, product entity linking and linking companies to triples. We describe the manual verification results of product entity linking and creating company links. Based on these results we describe issues with input data and provided company-url list. In Scaling Verification we investigate scaling and parallel efficiency of the first two parts of the pipeline to see how they would behave with bigger input data. In Scaling Up Graph Processing we describe what happens with bigger data in the network metrics calculation part of the pipeline. Lastly, we describe how the created pipeline could be improved in future work.

Measurable requirements for the solution that were described in the introduction and are verified in this chapter are:

1. Total run time of the solution must be less than one week.
2. The solution must be scalable.
3. The solution must support graphs that are 100 times bigger than the one initially constructed.
4. The solution must support using a script introduced in Chapter 4 to calculate derivatives on network metrics time series.

This chapter does not verify the last requirement about using the script, as it was verified in Chapter 4.

5.1 Cluster

The whole pipeline was ran in Inforegister's Cloudera cluster. The cluster consists of 3 identical machines.

Hardware

Every machine has 256 GB of RAM. Each machine has two Intel Xeon E5-2630 v4 @ 2.20GHz CPUs, both have 10 cores. Each machine has 900 GB of storage. HDFS provides 150 TB of storage, usable by all machines. The machines are connected over a 10 Gigabit Ethernet network.

Software

Cloudera[16] version running on the machines is Cloudera Express 5.10.0. This also defines default Linux, Spark, Python and Java versions. Every machine is running on Linux 14.04. Cluster has Apache Spark 1.6.0 built with Scala 2.10.5. Python version is 2.7.6 and Java 1.7.0 but more modern versions are also installed, such as Python 3 and Java 1.8.

Yarn

Apache Spark jobs are submitted through Yarn resource management and scheduling framework. At the moment the configuration gives Apache Spark 120 total cores and 300 GB of RAM.

5.2 Dataset Description

Inforegister OÜ is crawling the Estonian web using NetarchiveSuite[17]. In this thesis we used their latest full crawl as an input for our pipeline. This crawl was ran in 17-24 August 2017 and it contains 10 519 WARC files, in total 4.5 terabytes. The web pages were crawled from a seed list of more interesting Estonian web pages, crawler was following links from them to discover new web pages. WARC files contain 88 895 983 WARC records - results of crawling for a web page. Out of those we have 1 827 511 records that are not interesting in the scope of this thesis - DNS information about crawled pages.

5.3 Linked Data Extraction

In the scope of this thesis we extracted only one type of linked data: microdata. In microdata extraction we only checked for text based WARC records. For that we checked the content-type (what the web page says about the content it serves) of WARC record and used only the ones that contain "http", "xml", json", "text". This filter ignored 40 334 028 WARC records - images, videos, compressed files, binary files, etc. In total microdata extraction was attempted from 42 743 137 web sites. Out of these 2 369 823 sites (5.5% of all sites) had microdata and 148 475 284 triples were extracted. This is a 49% increase compared to initial microdata extraction script presented in [RK16]. Microdata extraction failed with exception or error from 961 397 WARC records: 643 077 due to ExtractionException, 313 856 due to RuntimeException, 4 458 due to UnsupportedCharsetException and 6 due to StackOverflowError. ExtractionException happened when Any23 was attempting to parse triples, the message shown was "Error

[16]Cloudera, enterprise cloud software for Hadoop technologies - `https://www.cloudera.com/`

[17]NetarchiveSuite web crawling tool - `https://sbforge.org/display/NAS/NetarchiveSuite`

while processing on subject". This exception is thrown due to a malformed URL and is caused by input files with content that is not according to standard. RuntimeException mostly happened when Any23 was trying to identify file content type using Apache Tika. It was caused by a "invalid char between encapsulated token end delimiter" when Apache Tika was attempting to detect if input was a comma separated value file. UnsupportedCharsetException was mostly caused by input files with unsupported encoding and came from TagSoupParser component of Any23. StackOverflowError happened when the process ran out of memory due to a recursive call. One cause for the Error was when Any23 was trying to create a tree structure of the contents of a WARC record but the tree became too big to be processed.

Table 7. Statistics about triples

Description	Count
Triples	148 475 284
Product triples	1 178 312
Product SKU triples not empty	33 412
Product connected to companies	714 004
Companies connected to products	1 230
Web pages in triples	104 428 510
Unique pages in triples	1 049 303
Web pages connected to companies	682 769

Statistics about extracted triples is presented in table 7. In total we extracted 148 million "html-microdata" triples from the latest crawl. 682 769 web pages (0.65% of all, 65% of unique web pages in triples) were connected to 3 174 unique companies (8.17% of all companies). There were 1 178 312 product related triples (0.78% of all triples). In total 1 230 companies were connected to 714 004 products. The numbers in tables contain duplicates because some websites were crawled multiple times and some triples are represented on multiple web pages. Analysis of SKU matches revealed that one triple is represented in 2 to 5 times.

We used the extracted microdata to build product-company graphs.

5.4 Product Entity Linking

Due to the extraction logic each product was represented using a separate microdata object. To link the microdata objects that represent the same product we performed product entity linking using SKU values. This helps create better quality graphs as companies are connected through products.

Number of unique products with SKU that were connected to a company is 13 699. Table 8 describes the number of companies a product is connected to, grouped

by connected company count - essentially it says that 13 420 products are connected to only one company, 279 products are connected to more than one company and only 33 products are connected to three companies.

Table 8. Frequency table of number of companies a product is connected to

Companies	Count of Products
1	13 420
2	245
3	33

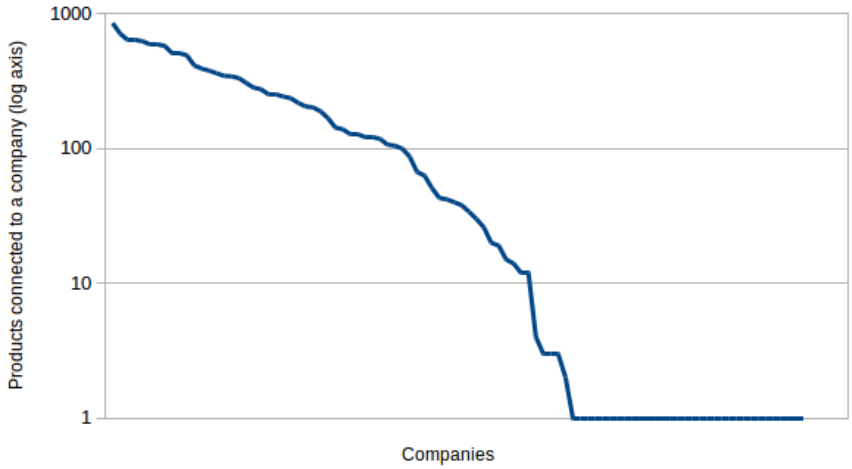


Figure 8. Number of products with SKU per company

Ninety four companies are connected to products that have SKU value, this is shown in figure 8. Biggest company is connected to 852 products. Forty companies are connected to 100 or more products while 32 companies are connected to only one product. This means that 22 companies are connected to 2 to 99 products. Twelve companies offer 51% of the total number of matched products.

A more concrete example of the graph is shown in figure 9 where two companies are connected through multiple products. Company 10730372 is Autoextra OÜ (autoextra.ee), company 11034918 is Ghozt OÜ (carsec.ee), smaller company with one product is 11485207 - Audiofookus OÜ (audiofookus.ee). The subgraph contains vertices that are maximum 3 steps from "autoextra.ee". It contains 282 products out of which 123 are connected to both "autoextra.ee" and "carsec.ee". This graph demonstrates what was expected in this step. We managed to connect two companies through the products they

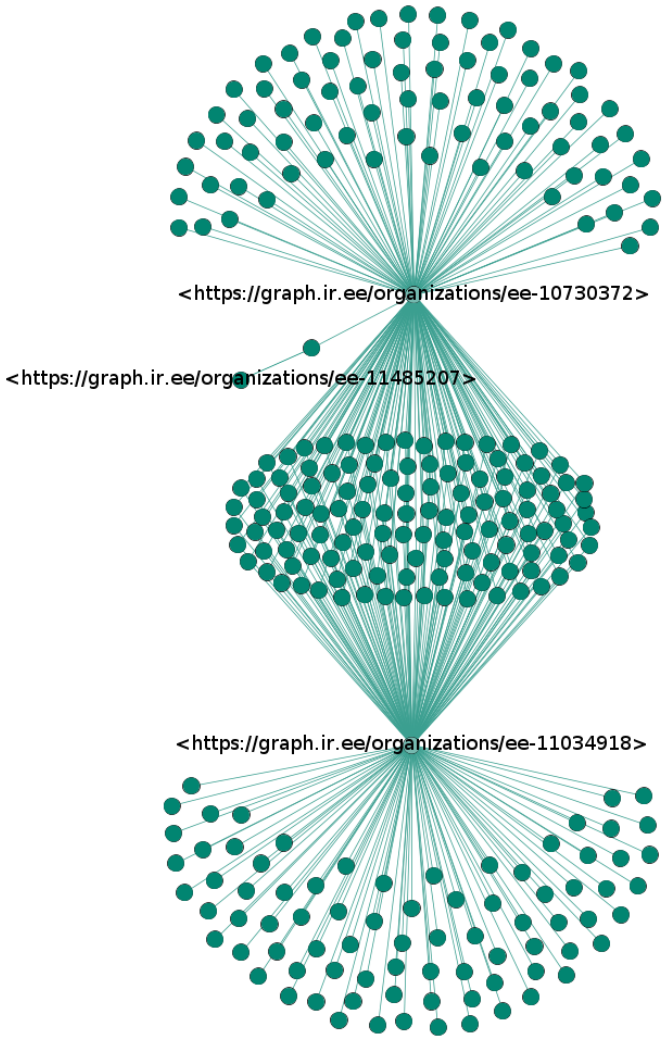


Figure 9. Subgraph showing two companies that offer many same products

offer. However, the number of products connected to more than one company is too low. Future work is needed to increase this number.

5.5 Validation of Connecting Companies and Products

To check the accuracy of connecting companies to triples we created three tests. In these tests we manually compared subsets of results, since it would take too long to look through all the results, and there was no good way to use automatic tests. First, we

manually evaluated 100 matched triples to assess the accuracy of creating company-triple links. Second, we manually checked 100 URLs that were not connected to a triple to see why they were not connected. We wanted to know if this was due to our developed solution or if it happened because they were not in the list of Estonian companies and their websites. Third, we compared the number of triples connected to a company to total number of triples that were not connected to a company.

The first step of evaluation consisted of checking the URL in subject of triple to the URL of the company it was connected to. 100 triples were selected in the following way: first, we selected 50 output files[18] and from each of them we selected one random triple. The process was then repeated for another 50 files, where we selected a random complex URL in order to assess the accuracy of the solution for web sites that use a hosting provider or web sites that span multiple server. In this context a complex URL is an URL that has both server and domain, for example "www.shop.site.ee". We picked another random file if a complex URL was not found in 10 000 tries. All random selections were done using Python's (pseudo) random methods.

For both simple and complex URLs our solution had an accuracy of 100%. This is expected because the algorithm used strong matches (required full match, not partial) and had two ways to test if URLs are the same. The first test contained 5 complex URLs and no duplicates. Complex URL test set contained 34 unique companies.

Secondly, we checked URLs of triples that were not connected to a company. For that the process was very similar to what we had above: select 50 random triples from 50 random files that were not connected to a company. We then checked the URL against our company list.

The results of analyzing triples that are not connected to a company are shown in figure 10. Most of the checked URLs, 27 out of the 50 random URLs (54%), belong to foreign companies - no direct relationship to Estonia. One foreign company was a Latvian web page of an Estonian web shop (kriso.lv). In total, there were 22 web pages that are connected to Estonia but were not linked to an Estonian company (this also includes one duplicate company). Out of these companies 20 (40% of all, 91% of Estonian companies) were not represented in the list of Estonian companies and their URLs. Two simple URLs that were in company list were not connected to company code. There were 6 companies that were in the list but the concrete domain URL was not in the list. The analysis shows that company matching can be improved: we discovered URLs would not be matched by our solution - for example www.ema.edu.ee, https://sites.google.com/site/eestiveemoto and blogspot.com.ee. This means that the URL matching should be improved in future work. This analysis has provided additional sites to be added to the company code list, which will in turn help connect more companies to triples.

Third test consisted of finding out how many triples were not connected to companies. In total 682 769 URLs out of 1 049 303 unique triple objects were connected to a

[18] At this point the pipeline created one output file for every input file - we have 10 519 files.

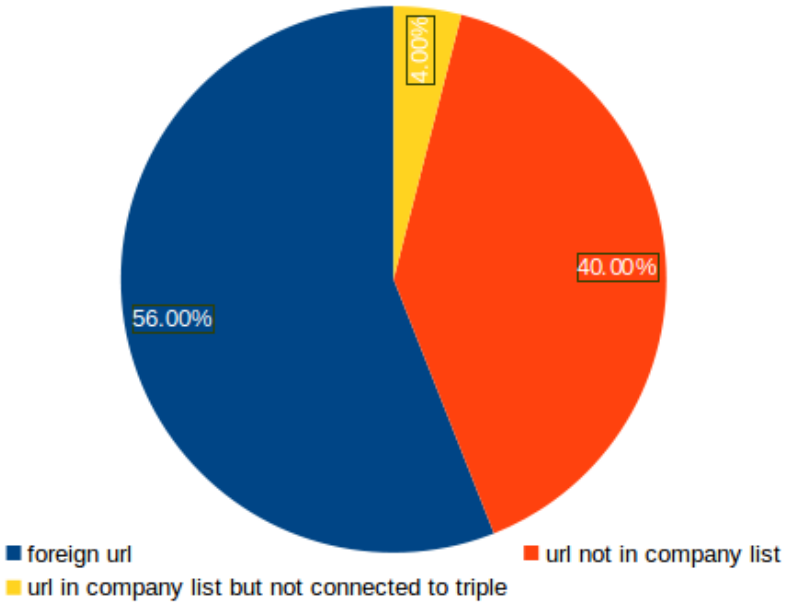


Figure 10. Analysis of triples not connected to a company

company, this means that we managed to connect 65% of triples. Analysis above showed that 46% of triples that were not matched belong to Estonian companies. This means that if matching is improved and company list is updated, we have the potential to increase the number of triples connected to company by 25%, since we could match approximately 168 600 (46% of URLs not matched) additional URLs to companies.

These three tests show that connecting companies to products has a high accuracy but it can always be improved. We saw that there were some products that were not connected due to our implementation, but most of the improvement could be achieved by fixing the input data. We showed that it is possible to increase the number of companies connected to products by up to 25% if the list of company URLs is improved.

5.6 Validation of Product Entity Linking

To check the accuracy of product entity linking via SKU we manually checked products that were presented in figure 9. This was done manually as there was no good way to do it using automatic tests. We also checked all entity linked products that were connected to more than one company. We investigated if the product is the same by checking the contents of URLs related to the product.

For example, one product that the two companies shown in figure 9 share is SKU "AP900" - Liteon AP900 Drive by Wire Cruise Control and Speed Limiter. Carsec

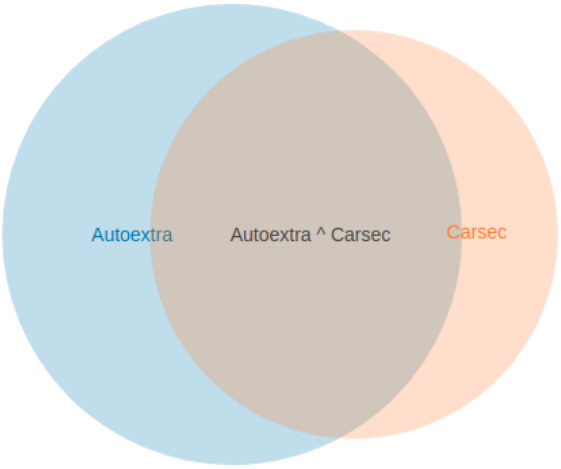


Figure 11. Venn diagram of 20 leaves of Autoextra and Carsec from figure 9

has the product under the name "AP900 analog" and autoextra offers it as "AP900 püsikiirushoidja kmpl" (AP900 cruise control set). It is clearly the same product when comparing name, description and image. To validate product entity linking we manually checked 25 products (20% of shared products) that were connected to both companies. Results are illustrated in figure 11. We found that 24 of those match on both sites and both have stopped offering one product that they stocked in 2017. We also checked 20 products out of the 158 that were connected only to one company (degree 1). We found that carsec offers 11 of them, autoextra offers 14. Two products were not available on either site. Eight products were actually offered by both web sites. The cause why the products that were offered by both sites but were not connected in the graphs is the crawling quality - these product pages were not represented in the WARC files.

A wider validation of product matching was also carried out. We found that 122 out of 279 products, that we said were the same and were connected to more than one company, had integers as SKU. This is mentioned because with these SKUs there are bound to be collisions, for example SKU 1001 is found from two sites: biotrend.ee where it is a facial cleaner and ross.ee, where it represents iron nails. SKU 111151 connects ilutooted.ee - a cosmetic shop - and hortulus.ee - a website selling garden and forest tools. It is easy to deduct from these and other product "matches" that SKUs with integer values are not useful. This means that there are approximately 165 SKUs left that connect two companies. Out of those 165 only five were not represented in the graph above: "kasutatud11", "must", "valge", "KVAV4444", "SK-2779". What we can conclude here is that the SKUs are not always unique. There is no good standard in use at this time but results show that the more complex an SKU, the better it is.

There were several examples demonstrated earlier. SKU "EHA6041XOK" mentioned in Chapter 4.4 did not make into the the final graph as one provider, alfashop, was not

included in the list connecting companies to URLs. Chapter 4.4 also mentioned products from sites that were not crawled: Euronics and ishop, and from sites not in company list: ounaturg and idream.

We have shown that the this solution can be used to link companies through products. We have also identified what needs to be fixed to increase the number of connected companies. To do this we need to improve the list of company URLs so that more companies are connected to products. We also found that using integers as SKUs is not a good idea because links created using these SKUs are of low quality. Based on the low number of companies that are connected through products we can say that this result is not directly usable as input to machine learning models. In order to use the results we need to greatly increase the number of companies that are connected to more than one product. However, these results can be used to complement other data about companies.

Next we investigate and describe how to improve the input data to improve linking products and companies.

5.7 Issues with Input Data

In connecting companies to triples we used a list provided by Inforegister OÜ. Unfortunately that list contained only Estonian companies and many Estonian company websites were not represented. By investigating the results we have been able to connect multiple websites to their respective companies. This means that future results will be of higher quality and it will also help Inforegister in other parts of their business. Initial data contained 677 product sameAs tags but none were connected to a company since the websites that they were from were not in company URL list.

Main reason behind the small number of connections is crawling quality. Table 9 shows top 10 domains according to number of triples extracted. It is visible that five of them are not part of the Estonian web (all dot com domains). All dot ee domains can be connected to company codes except the last two that were not on the company URL list.

Table 10 shows top 10 most visited Estonian websites according to Freqmedia[19]. First column shows if the website has schema.org triples, second column shows how many triples were extracted. Triples were extracted from only two web sites on the list. Six sites do not contain any schema.org triples. Two sites that have triples but from which none were extracted, on24.ee and perekool.ee, were not represented in the crawl. From this it can be concluded that many websites do not have triples even though they would benefit from them. Kava.ee could use triples to present information about movies and TV-shows, Buduaar could use triples for the products that are offered on their marketplace.

These two tables show that improving crawling would improve the result. Previous

[19]Estonian most popular websites on week 11 of 2018 - `https://metrix.station.ee/?subm=1&act=&cat=&selweek=2018-03-12`

Table 9. Top domains by extracted html-microdata triples

Site	Number of Triples
microsoft.com	4 809 903
youtube.com	4 492 249
google.com	2 579 560
piletilevi.ee	2 472 101
astri.ee	1 803 918
skype.com	1 197 703
delfi.ee	1 109 298
opera.com	1 108 533
crystalmix.eu	953 234
toonekurg.ee	915 643

Table 10. Estonian most visited sites on week 11 of 2018, according to Freqmedia

Companies	Has triples	Triples Extracted
ekool.eu	no	0
okidoki.ee	yes	41 063
upload.ee	no	0
buduaar.ee	no	0
kava.ee	no	0
on24.ee	yes	0
piletilevi.ee	yes	2 472 101
perekool.ee	yes	0
kalkulaator.ee	no	0
cv.ee	no*	0

At the time of the crawl (August 2017) cv.ee contained no triples. Currently (May 2018), the page has schema.org JobOffer triples.

data from Inforegister OÜ shows that the coverage of initial input data is approximately 48% of all Estonian URLs. Coverage of web shops is between 0.70% and 30%. Improving the coverage of web shops would surely increase the number of product triples extracted. There were some web sites that were represented 2 to 5 times, as mentioned earlier in Chapter 5.2. By reducing the number of duplicates and increasing the number of crawled web sites it is possible to increase the accuracy of the solution. This thesis provides a list of websites that were not connected to companies and websites that were not crawled at all.

Another issue encountered was with connecting triple URLs to companies. There are websites that are hard to match to a company. One example would be autoextra.ee.klient.veebimajutus.ee that should belong to autoextra.ee. However, this was matched to veebimajutus.ee, as we only checked the domain (last part of the url before country code). A way to fix this problem would be to design a more complex algorithm with a list of most popular hosting sites, so that they could be handled differently. Another way to fix this would be to improve the crawling because the autoextra.ee.klient.veebimajutus.ee should be autoextra.ee.

5.8 Scaling Verification

We carried out performance measurements to validate the first requirement. Implemented pipeline was executed on one whole web crawl data set. The results are shown in table 11. Total runtime was 2.3 days which is under 7 days that was defined as the first requirement for this pipeline. For the first part of the pipeline we present two times. The first one is run time whose results are analyzed in this chapter. The other, with the asterisk, is

Table 11. Run times of parts of the pipeline

Part of Pipeline	Run Time
Extract linked data from WARC files	41.9 h 20.2h without HTML/triple fix*
Connect companies to triples	13.0 h
Bipartite graphs that link products and companies	0.13 h
Network metrics of bipartite graphs	0.07 h
Network metrics to time series values	< 0.01 h per time series
Derivatives of network metrics	0.02 h
Total	55.16 hours - **2.3 days**

linked data extraction that did not have HTML fixing and output triple fixing (that were described in Chapter 4.2). It is two times faster but outputs less triples. The table shows that the two first tasks are the most time consuming - they are the bottlenecks of this pipeline. To investigate the performance of the application and to be able to estimate how it behaves in future when the size of data changes, we ran additional performance tests on first two parts of the pipeline to calculate their parallel speedup and efficiency.

Speedup [Sco11] measures how much application run time decreases when we add additional computing resources. Speedup is calculated using the formula shown in equation 1. In the formula p is number of executors, T(n,1) is run time with one executor and T(n,p) is run time with p executors. In ideal situation S(p) = p, meaning that increasing the number of cores x times will decrease runtime x times, but this rarely happens since adding extra executors increases communication overhead.

$$S(p) = \frac{T(n,1)}{T(n,p)} \tag{1}$$

Parallel efficiency [Sco11] compares achieved speedup and ideal speedup. Formula for calculating parallel efficiency is shown in equation 2. Efficiency is speedup divided by the number of executors. Perfect speedup is 1 but this happens rarely.

$$E(p) = \frac{T(n,1)}{p * T(n,p)} = \frac{S(p)}{p} \tag{2}$$

The tests were ran on three different input sizes: 25 GB, 50 GB and 100 GB - we selected these three sizes because 50 GB represents 1% of the input size but the run times on these dataset sizes are long enough to give meaningful information about performance. Shorter run times would be distorted by Spark startup and job scheduling overhead. Another reason why smaller datasets were used was because calculating parallel speedup and efficiency required computing the runtime values on a single core and measuring such values take very long time on big datasets. The datasets were formed by selecting

WARC files from the crawl that was used as input to pipeline, we made sure to select files from different parts of the crawl to keep the variety of websites. We elected to run the tests on 1, 3 and 6 executors (executor - separate processes with its own CPU cores), every executor had one CPU core. One executor provides a baseline for speedup. Maximum was six executors as linked data extraction with HTML and triple fixing is so memory heavy that we can only use six executors at a time. Using more executors would exhaust all available RAM in YARN and crash the application. We used three executors to see what happens when we increase the number of executors 3 times, compared to one executor, and what happens when we double the number of executors (from 3 to 6). With every executor-dataset size we ran three tests and took the average run time.

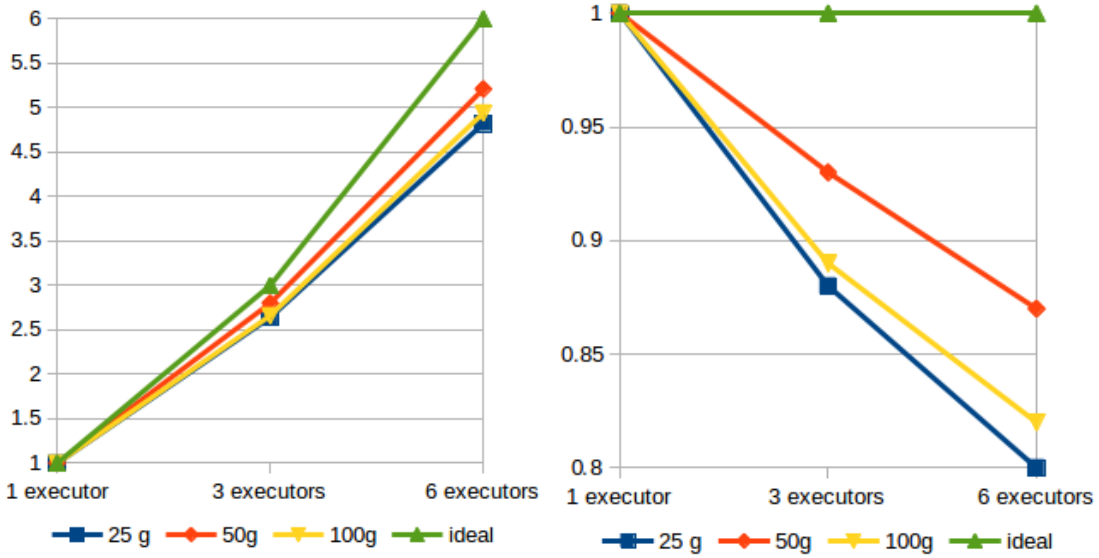
Table 12. Microdata extraction run times in seconds for 25, 50 and 100 GB input sizes

	100 GB	50 GB	25 GB
6 CPU cores	2 723 s	1 101 s	510 s
3 CPU cores	5 047 s	2 049 s	932 s
1 CPU core	13 367 s	5 733 s	2 457 s

Results of the tests on the first part of the pipeline are shown in table 12. Speedup is shown in figure 12a and parallel efficiency is shown in figure 12b. Extraction's efficiency is between 80% and 87% when 6 executors are used. Efficiency of 80% shows that 80% of the computing power is well utilized but 20% is wasted waiting behind other processes and different kinds of overhead (scheduling, distributing jobs, serializing the results). There is no limit for good parallel efficiency but 80% is acceptable. Efficiency is the worst on smallest dataset size, efficiency is best with 50 GB dataset size. However, more tests are needed to pinpoint the peak point - when speedup plateaus and adding new executors does not improve run time. These tests also highlight the need to reduce the memory consumption of the extractor, in order to use more CPU cores and speed up the process of extraction. Memory consumption could be reduced by using smaller input files, currently, most WARC files are 1 GB.

For measuring the efficiency of linking triples to companies we made a few changes to experiment design that we used with triple extraction. In this part we use the triples extracted in previous tests from 25, 50 and 100 GB of WARC files - 543 427, 1 138 064 and 2 954 490 triples. We also used more cores as this process was not that memory heavy and we did not have to worry about using all the memory of the cluster. We used 1, 2, 4, 8, 16 and 32 cores, all multiples of two to see how much execution time changes when we keep doubling the number of available cores.

Results of tests on second part of the pipeline are shown in table 13. Speedup is shown in figure 13a and parallel efficiency is shown in figure 13b. We see that the smallest dataset was the least efficient, since there was most overhead. 50 GB dataset is again the most efficient. Overall, efficiency with small number of cores is high but it



(a) Speedup of Microdata Extraction.

(b) Parallel Efficiency of Microdata Extraction

Figure 12. Speedup and parallel efficiency of microdata extraction

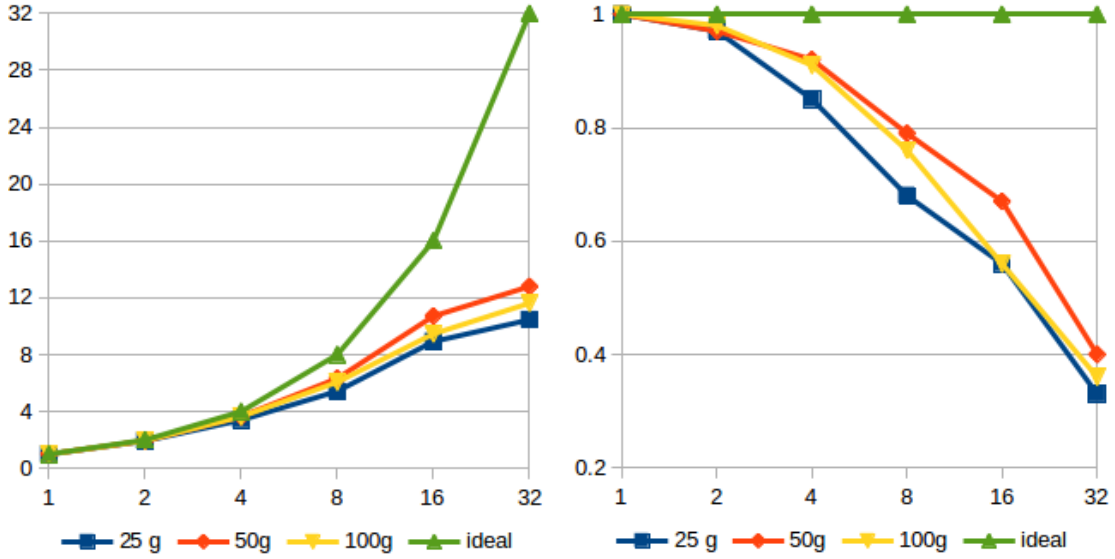
Table 13. Connecting triples to companies run times for microdata triples extracted from 25, 50 and 100 GB input sizes

	100 GB	50 GB	25 GB
32 CPU cores	1 002 s	412 s	249 s
16 CPU cores	1 231 s	492 s	292 s
8 CPU cores	1 918 s	832 s	460 s
4 CPU cores	3 197 s	1 349 s	766 s
2 CPU cores	5 960 s	2 719 s	1 351 s
1 CPU core	11 645 s	5 269 s	2 607 s

drops quickly. With 32 cores the efficiency is below 40% for all input sizes. This low efficiency with more cores is caused by higher communication overhead, since Spark needs to constantly send and receive results from the executors. In this part we only return distinct triples, in order to reduce the number of duplicates. When other parts of the pipeline are embarrassingly parallel[20], then Spark's *.distinct()* requires data exchange between executors, that takes more time with more executors. Another reason for low parallel efficiency is the number of files: our test datasets contained only 34, 78 and 194 input files. Using 32 executors with 34 input files is not very efficient, as there will be about one input file per executor. Low number of files per executor means that if all

[20]Embarrassingly parallel - really easy to divide into tasks that can be calculated independently

executors work through their assigned inputs fast, but there are files that take longer time, then they will be the bottlenecks.



(a) Speedup of linking triples to companies (b) Parallel efficiency: triples to companies

Figure 13. Speedup and parallel efficiency of linking triples to companies

Low parallel efficiency means that even though 32 cores gives a speedup of about 11.6 times, compared to only using 1 core, it is more viable to use 16 cores that offer speed up of 9.7 times. Using 16 cores instead of 32 would leave 16 cores for other processes. We could decide that duplicates are this part is okay, that would mean that one URL-company link is represented hundreds of times, but this would hurt other parts of the pipeline. However, the result with bigger number of cores only applies with these input sizes and the results might be different with bigger inputs.

With these tests we show that our solution is scalable and can handle bigger input data sizes, therefore fulfilling the second requirement set for the pipeline: "The solution must be scalable". The tests did not reveal mistakes in algorithm design that would stop us from using more cores or bigger dataset sizes. We did find, however, that linked data extraction could be improved by using smaller WARC files - this way the extraction would use less memory.

5.9 Scaling Up Graph Processing

One requirements of this thesis was that it should also work with 100 times bigger graphs. This requirement was set to see how the solution would scale and handle situations when we have more products connected to companies. This would happen when we improve

the pipeline or when we use bigger web crawl files. To demonstrate that our solution fills this requirement we generated random triples, connected to random companies, such that we would have a bigger graph than the one that was extracted from web crawl. The graph we built from triples of web crawl had a little under 15 000 vertices.

One hundred times bigger graph contains one and a half million vertices. We wanted to make the graph as life-like as possible and generated data that would be similar to real world data. The generated data contains 35 000 companies (list connecting companies and URLs contains approximately the same amount of companies) and one and a half million product nodes (we extracted 1.2 million product triples from web crawl). We generated 750 000 SKUs to simulate a situation where a lot of products are connected via SKU. Products were connected to companies such that 10 percent of the companies would represent 50 % of products - in this way there are many companies with a lot of products, there are smaller companies that are connected to less products, and there are companies that are not connected to a product at all. Since the number of SKUs is half the number of products then, on average, a product is represented two times.

Table 14. Scaling up graph processing test results

	15 000 vertices	150 000	1 500 000
Calculating network metrics	4.1 minutes	4.5 min	5.6 min

The results are shown in table 14. Calculating network metrics on 100 times bigger graph took 5.6 minutes. It does not take 100 times longer, this is most likely due to the fact, that the graphs are still pretty small and Spark can easily handle them. The runtime is proof that the solution can handle bigger graphs. Most time consuming part of the graph was calculating PageRank that took more time than other metrics combined. This does show that, even though we can easily handle 100 times bigger graph, the algorithms used to calculate the network metrics are not complex. With more complex algorithms the results would most likely be different, this should be investigated in future work. The tests were carried out with generated data that was similar to data from Estonian web crawl. To see how the graph algorithm handles other graph types we should use another crawl, such as Common Crawl [Fou] as input and build graphs based on it. Common Crawl's [Fou] April 2018 crawl contains 54 TB of WARC files.

With this we have shown that our solution fills the third requirement that was set on the pipeline : "The solution must support graphs that are 100 times bigger than the one initially constructed." More tests are still needed to see the run time on graphs with different structure.

5.10 Improving the Pipeline

The pipeline can be improved further in order to extract more features and to decrease runtime of the pipeline. First step of the pipeline could benefit from removing duplicates in crawling output. This can be done by using a script that checks for the full URL of each WARC record and only keeps unique records. One possibility to speed up other parts of the pipeline is to further reduce the number of triples that need to be processed by removing triples that contain identical information. This can be done by using a script presented by Rehman [RK16].

We could also increase the number of triples extracted in the pipeline. In Chapter 2.1 we described how, according to WDC, linked data should be found from 39% of the websites, but we extracted it from 5.5% of web sites. Even if the extraction would be without exceptions, and we would get triples from each web site whose processing gave an exception, we would get linked data from only 7.8% of the websites in triples. Assuming Estonian web has the same proportion of linked data than WDC crawl, then we could, in theory, extract triples from 16.7 million websites. Assuming the distribution of triples of WDC web sites and Estonian web sites is similar, then the potential would be one billion triples (a 4x increase). In our solution we only used html-microdata extractor. We also tried to run the test with all extractors - then runtime of the script was 18 days (using only html-microdata extractor took 40 hours) but the increase in triples was only 20% - number of triples with all extractors is 202 292 879. In order to increase the number of extracted triples we need to improve crawling process, but improving the script would also benefit - updating Java version and adding additional tools to reduce the number of exceptions encountered would help here.

Some URL-company links that were missing from initial file were already provided in this thesis to help increase the number of links in second step of the pipeline. It can be expanded by analyzing the URLs that were not connected to any company in order to detect URLs that belong to Estonian companies but are not in the list.

Improving product entity linking would give us more connections between companies and products. The number of links that we achieved using only SKUs was not good. Chapter 3 gave an overview of other ways to link products. All good solutions categorized the products before matching, it simplifies entity linking as it groups similar products together. Another way to increase product matching results is to extract information from product name and description using natural language processing methods.

Network metrics part of the pipeline can be improved by adding additional metrics. This thesis has shown how to use predefined metrics and how to build custom metrics. The examples will help to implement new metrics, such as centrality and betweenness.

We are only using product microdata but our input contains multiple other triple types that can be also turned into a graph. We could create graphs that represent companies and job offers, companies and locations, companies and events and many more.

It is possible to increase the number of derivatives calculated. Currently presented

solution calculates 1 840 derivatives for every company for a years worth of data. These derivatives are calculated for each company alone. The number of derivatives can be increased by calculating derivatives between companies. It can also be increased by using moving window method: looking at parts of the data at a time, for example the first 6 weeks, then second to seventh week and so on.

6 Conclusion

Usage of linked data on the web is growing fast and websites who use it will benefit due to their data being more understandable to search engines [ACORH18]. Features extracted from linked data can be used for machine learning, for example to calculate the credit score of companies. However, there are some challenges with extracting features from linked data, such as scalability, stability and working with linked data [TAL14]. In this thesis we have developed a solution that solves these challenges and we have shown how features can be extracted from linked data. The presented solution is built in a scalable way to process web scale data that is measured in terabytes. Feature extraction is built on using the relationships that linked data provides.

The pipeline presented in this thesis begins with extracting features from Internet archive files. Linked data extracted from these files is processed, such that microdata representation of products is connected to companies. Result is a graph connecting product microdata to companies. On this graph several network metrics are calculated. Network metrics from different web crawls are stored in time series that show changes in graphs over time. Features are extracted from these time series. Later, not part of this thesis, the extracted features are used as input to machine learning models that calculate company credit scores.

The pipeline is mostly built in Apache Spark. Spark gives the ability to easily use multiple machines and has exceptional performance when the problem can be divided into parts that can be processed independently. Spark is not required in parts of the pipeline where data sizes are smaller, as using it has an overhead. We took this into account and a part of the final solution uses standard one machine solutions.

The built pipeline solved all four requirements that were set for it. The run time of the pipeline was 2.3 days, which is lower than the 7 day limit that was set on it. The solution is scalable: adding additional resources to the computation reduces the run time with good efficiency. We showed that our solution was fast even with a graph that was 100 times bigger than the one that we constructed from one web crawl. Finally, our pipeline used a script provided to extract features from time series data. Thus, we have fulfilled all the initial requirements of this work. However, there are still potential upgrades that can improve this solution in future work.

We analyzed the graphs that were created and found that the number of products that connected two or more companies was too low to make the result directly usable as input to machine learning models. We expected to extract linked data from 39% of the domains but only got it from 5.5%. Because of this the pipeline and input data needs to be improved in future work to produce more links. However, the results can be used to complement other data about companies.

As a future work, the run time of the pipeline can be improved by reducing duplicates in data. Since the input is web data then it naturally contains duplicates. The same information is represented on multiple web pages and some web pages occur multiple

times in web crawling results. This means that there are multiple extracted triples that contain the same information. There already exist deduplication techniques to handle this.

Product entity linking can be improved in future work by using advanced techniques. It is possible to use natural language processing methods to extract information from product name and description. This information can be used to find links between products that do not have a SKU or have ambiguous SKU.

In this thesis we extracted 1 840 features per company but in future work this can be increased by calculating derivatives against other companies. Another way to increase the number of extracted features is by building additional graphs. We only used product microdata but there are many others: job offer, location, event and person. In future work, we plan to increase the number of features extracted by using the moving window methods: focusing on smaller parts of the input at a time.

References

[ABMP08] Ben Adida, Mark Birbeck, Shane McCarron, and Steven Pemberton. Rdfa in xhtml: Syntax and processing. *Recommendation, W3C*, 7, 2008. https://www.w3.org/TR/rdfa-syntax/.

[ACORH18] Jamshaid Ashraf, Richard Cyganiak, Sean O ' Riain, and Maja Hadzic. Open ebusiness ontology usage: Investigating community implementation of goodrelations. 04 2018.

[Agr] Aileen Agricola. Qualia media customers experience greater roi with identity graph analysi. Accessed April 2018. https://neo4j.com/news/qualia-media-customers-experience-greater-roi-identity-graph-analysis/.

[AHKK17] Ibrahim Abdelaziz, Razen Harbi, Zuhair Khayyat, and Panos Kalnis. A survey and experimental comparison of distributed sparql engines for very large rdf data. *Proc. VLDB Endow.*, 10(13):2049–2060, September 2017. https://doi.org/10.14778/3151106.3151109.

[AXL+15] Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomer Kaftan, Michael J Franklin, Ali Ghodsi, et al. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1383–1394. ACM, 2015.

[B+08] Dhruba Borthakur et al. Hdfs architecture guide. *Hadoop Apache Project*, 53, 2008.

[BHAR07] Chris Bizer, Tom Heath, Danny Ayers, and Yves Raimond. Interlinking open data on the web. In *Demonstrations track, 4th european semantic web conference, innsbruck, austria*, 2007.

[BHBL09] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. https://eprints.soton.ac.uk/271285/, 2009.

[BM04] Dave Beckett and Brian McBride. Rdf/xml syntax specification (revised). *W3C recommendation*, 10(2.3), 2004.

[BMP17] Chiristian Bizer, Robert Meusel, and Anna Primpeli. Web data commons - rdfa, microdata, and microformat data sets, 2017. http://webdatacommons.org/structureddata/index.html.

[C+14] World Wide Web Consortium et al. Html microdata. 2014. https://www.w3.org/TR/microdata/.

[C+18] World Wide Web Consortium et al. Json-ld 1.0: a json-based serialization for linked data. 2018.

[CJ18] Richard Cyganiak and Anja Jentzsch. Linking open data cloud diagram, 2011. *URL* http://lod-cloud.net, 2018.

[CKG+11] Weiwei Cheng, Gjergji Kasneci, Thore Graepel, David Stern, and Ralf Herbrich. Automated feature generation from structured knowledge. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 1395–1404, New York, NY, USA, 2011. ACM. http://doi.acm.org/10.1145/2063576.2063779.

[CS14] Gavin Carothers and Andy Seaborne. Rdf 1.1 n-triples: A line-based syntax for an rdf graph. *World Wide Web Consortium. http://www. w3. org/TR/n-triples/. Accessed April 2018*, 24, 2014.

[DJL+16] Ankur Dave, Alekh Jindal, Li Erran Li, Reynold Xin, Joseph Gonzalez, and Matei Zaharia. Graphframes: an integrated api for mixing graph and relational queries. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*, page 2. ACM, 2016.

[Fou] Common Crawl Foundation. Common crawl.

[GLG+12] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: distributed graph-parallel computation on natural graphs. In *OSDI*, volume 12, page 2, 2012.

[GPL+06] Rayid Ghani, Katharina Probst, Yan Liu, Marko Krema, and Andrew Fano. Text mining for product attribute extraction. *SIGKDD Explor. Newsl.*, 8(1):41–48, June 2006. http://doi.acm.org/10.1145/1147234.1147241.

[Gra] Graphframes. Graphframes overview. https://graphframes.github.io/.

[Had09] Apache Hadoop. Hadoop, 2009.

[HAK+16] Razen Harbi, Ibrahim Abdelaziz, Panos Kalnis, Nikos Mamoulis, Yasser Ebrahim, and Majed Sahli. Accelerating sparql queries by exploiting hash-based locality and adaptive partitioning. *The VLDB Journal*, 25(3):355–380, 2016.

[HBF09] Olaf Hartig, Christian Bizer, and Johann-Christoph Freytag. Executing sparql queries over the web of linked data. In Abraham Bernstein, David R.

Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, editors, *The Semantic Web - ISWC 2009*, pages 293–309, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[KAA+13] Zuhair Khayyat, Karim Awara, Amani Alonazi, Hani Jamjoom, Dan Williams, and Panos Kalnis. Mizan: a system for dynamic load balancing in large-scale graph processing. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 169–182. ACM, 2013.

[KGAF11] Anitha Kannan, Inmar E. Givoni, Rakesh Agrawal, and Ariel Fuxman. Matching unstructured product offers to structured product specifications. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 404–412, New York, NY, USA, 2011. ACM. http://doi.acm.org/10.1145/2020408.2020474.

[KK17] Peep Küngas and Jüri Kuusik. Business credit scoring of estonian organizations with xgboost, 12 2017.

[KSI+08] Gjergji Kasneci, Fabian M Suchanek, Georgiana Ifrim, Maya Ramanath, and Gerhard Weikum. Naga: Searching and ranking knowledge. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 953–962. IEEE, 2008.

[Lan01] Douglas Laney. 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group, February 2001. https://www.bibsonomy.org/bibtex/263868097d6e1998de3d88fcbb7670ca6/sb3000.

[LS99] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Recommendation 22 February 1999 REC-rdf-syntax-19990222, W3C, Cambridge, MA, 1999. http://www.w3.org/TR/REC-rdf-syntax/.

[MAB+10] Grzegorz Malewicz, Matthew H Austern, Aart JC 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*, pages 135–146. ACM, 2010.

[MBY+16] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.

[Meu17] Robert Meusel. Web-scale profiling of semantic annotations in html pages, 2017. http://ub-madoc.bib.uni-mannheim.de/41884/.

[MP12] Peter Mika and Tim Potter. Metadata statistics for a large web corpus. *LDOW*, 937, 2012. Accessed 14 Apr. 2018., http://events.linkeddata.org/ldow2012/papers/ldow2012-inv-paper-1.pdf.

[MV16] Rajesh Mahule and Om Prakash Vyas. Leveraging linked open data information extraction for data mining applications. *Turkish Journal of Electrical Engineering & Computer Sciences*, 24(6):4874–4884, 2016.

[Neo] Inc Neo4j. Neo4j graph platform. https://neo4j.com/.

[PBB14] Petar Petrovski, Volha Bryl, and Christian Bizer. Integrating product data from websites offering microdata markup. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, pages 1299–1304, New York, NY, USA, 2014. ACM. http://doi.acm.org/10.1145/2567948.2579704.

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[PPPH16] Ristoski Petar, Petrovski Petar, Mika Peter, and Paulheim Heiko. A machine learning approach for product matching and categorization. *Semantic Web journa*, 2016. http://www.semantic-web-journal.net/system/files/swj1470.pdf.

[PTK+14] Nikolaos Papailiou, Dimitrios Tsoumakos, Ioannis Konstantinou, Panagiotis Karras, and Nectarios Koziris. H 2 rdf+: an efficient data management system for big rdf graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 909–912. ACM, 2014.

[RK16] Khalil Ur Rehman and Peep Küngas. Microdata deduplication with spark, 2016. https://comserv.cs.ut.ee/home/files/Rehman+softwareengineering+2016.pdf?study=ATILoputoo&reference=C809E8E183E0B571901F97306B4F8478EF73127A.

[SBP14] Max Schmachtenberg, Christian Bizer, and Heiko Paulheim. State of the lod cloud 2014. *University of Mannheim, Data and Web Science Group. August*, 30, 2014.

[Sco11] L. Ridgway Scott. Scientific parallel computing,a short course, 2011. Accessed April 2018, slides based on L. R. Scott, T. W. Clark, and B. Bagheri. Scientific Parlallel Computing. Princeton University Press, 2005. http://people.cs.uchicago.edu/~ridg/newpsc/cp.pdf.

[SLZ09] François Scharffe, Yanbin Liu, and Chuguang Zhou. Rdf-ai: an architecture for rdf datasets matching, fusion and interlink. In *Proc. IJCAI 2009 workshop on Identity, reference, and knowledge representation (IR-KR), Pasadena (CA US)*, 2009.

[Spa16] Apache Spark. Apache spark: Lightning-fast cluster computing. *URL* http://spark.apache.org, 2016.

[SW13] Semih Salihoglu and Jennifer Widom. Gps: a graph processing system. In *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, page 22. ACM, 2013.

[TAL14] Jiliang Tang, Salem Alelyani, and Huang Liu. *Data Classification: Algorithms and Applications*. CRC Press, 2014.

[vBBR+15] Ronald van Bezu, Sjoerd Borst, Rick Rijkse, Jim Verhagen, Damir Vandic, and Flavius Frasincar. Multi-component similarity method for web product duplicate detection. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, pages 761–768, New York, NY, USA, 2015. ACM. http://doi.acm.org/10.1145/2695664.2695818.

[VMD+13] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 5:1–5:16, New York, NY, USA, 2013. ACM. http://doi.acm.org/10.1145/2523616.2523633.

[VvDF12] Damir Vandic, Jan-Willem van Dam, and Flavius Frasincar. Faceted product search powered by the semantic web. *Decision Support Systems*, 53(3):425 – 437, 2012. http://www.sciencedirect.com/science/article/pii/S0167923612000681.

[XGFS13] 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*, page 2. ACM, 2013.

[ZCD+12] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

Appendix

I. Acronyms

AAND Average Nearest Neighbour Degree. 38

HDFS Hadoop Distributed File System. 19

IID independent and identically distributed. 21, 28, 41

OGP Open Graph protocol. 13

RDD Resilient Distributed Dataset. 19

RDF Resource Description Framework. 13

RDFa Resource Description Framework in Attributes. 13

TB terabyte. 10

WARC Web ARChive. 31

WDC Web Data Commons. 23

YARN Yet Another Resource Negotiator. 19

II. Glossary

degree Number of edges connected to a vertex. 21, 38

entity disambiguation See entity linking. 22

entity linking The process of detecting similar objects and using one representation for it. For example using one object to represent a product that two websites offer. 22

interlinking Adding new links to data, for example adding relationships between websites and companies. 22

PageRank A graph algorithm that determines the importance of a vertex based on the vertices (and their importance) that connect to it. 38

Pyspark Apache Spark connector for Python. 19

snapshot Information crawled from websites at some point in time. 40

time series Sequence of values ordered over time. 21

III. Implementation Code

Metadata extraction

Source code for extraction of metadata from web archive files, based on solution by Rehman[RK16], can be found from the following GitHub repository:

`https://github.com/MadisKarli/microdeduplication`

Pipeline

Source code for every other part of the pipeline, including scripts to generate data, run tests used in this thesis and scripts to convert Graphframes output to GraphML format, can be found from the following GitHub repository :

`https://github.com/MadisKarli/masterthesis`

IV. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Madis-Karli Koppel**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1 reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2 make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

   of my thesis

   Large Scale Feature Extraction from Linked Web Data

   supervised by Pelle Jakovits and Peep Küngas

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 21.05.2018