

RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT BONN

MASTER THESIS IN COMPUTER SCIENCE

Distributed RDF Clustering Framework

Author:

Tina BOROUKHIAN

First Examiner:

Prof. Dr. Jens LEHMANN

Second Examiner:

Dr. Damien GRAUX

Supervisor:

Dr. Hajira JABEEN

Submitted: September 30, 2018

Declaration of Authorship

I declare that the work presented here is original and the result of my own investigations. Formulations and ideas taken from other sources are cited as such. It has not been submitted, either in part or whole, for a degree at this or any other university.

Location, Date

Signature

Acknowledge

I would like to express my sincere gratitude to my supervisor Dr. Hajira Jabeen for the continuous support of my master study and research, for her patience, motivation, enthusiasm, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis. I am immensely grateful to Prof. Dr. Jens Lehmann for letting me pursue master thesis at the Smart Data Analytics (SDA) group in the University of Bonn (at the Department of Computer Science) and for giving me an opportunity to work on Big Data technologies. Without his support and motivation, this work would not have been possible. Special thanks to the member of my committee Dr. Damien Graux. I would like to thank Mr. Gëzim Sejdiu for his technical supports. Finally, I must express my very profound gratitude to my parents and to my husband for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.

Distributed RDF Clustering Framework

Tina Boroukhian

Friedrich-Wilhelms-Universität Bonn

September 30, 2018

Abstract

In this thesis, we propose a framework, Distributed RDF Clustering Framework, which is useful for RDF graph mining and adopted for clustering and partitioning of big RDF datasets. Clustering is a knowledge discovery method in which data are divided into groups in a way that objects in each group share more similarity than with objects in other groups. There are difficulties for applying traditional clustering techniques to big data due to some challenges, including high complexity and computational cost, and inconsistency in data flow. Here, we study the various clustering algorithms including Borderflow clustering algorithm, Link-based clustering algorithm and PowerIteration clustering algorithm. Our main aim is to show the comparison of these clustering algorithms and find out which algorithm will be most suitable for clustering of big RDF datasets. The clustering algorithms of our framework are implemented in Scala. Scala is a powerful language supported by Spark. Apache Spark is a big data framework which offers a reliable method for general-purpose processing of big datasets. BorderFlow and Link-based are two local graph clustering algorithms that allow overlapping clusters. Here, we address the problem of improving the runtime of BorderFlow algorithm. We introduce FH-BF clustering algorithm which is a new partitioning clustering algorithm improving the BorderFlow clustering approach. FH-BF is much more faster than BorderFlow; in particular, it generates clusters without overlaps. Similarity measures are core components used by clustering algorithms to cluster similar data points into the same clusters. We also analyze and compare the influence of different similarity measures on these clustering algorithms. We use feature model semantic similarity measures (such as Jaccard, Ratio Model, Rodriguez-Egenhofer and Batet) for calculating the similarity between resources in RDF datasets. We observe that the candidate semantic similarity measures are not working well for big datasets. Hence, we define a pseudo similarity measure, it approximates the Jaccard similarity for highly overlapping communities can exhibit more external than internal connections; moreover, it is working well for big datasets. Here, we successfully implement PowerIteration on big datasets extracted from DBpedia and BSBM databases and apply this pseudo similarity measure. We also implement Silhouette algorithm on large datasets, as a method of interpretation and validation of consistency within clusters of data. We find

that, by our experiments, PowerIteration clustering and Silhouette evaluation are scalable and efficient algorithms on Spark. On the other hand, by a practical scenario for BorthorFlow and Link-based, we observe that they are iterative recursive algorithms; hence, they cannot be performed in parallel by using the Spark methods. This motivates us to present a new algorithm based on Link-based clustering method suitable for clustering of big datasets. Furthermore, some various datasets which have obvious clusters as input are prepared from DBpedia to test the quality of the other candidate clustering algorithms.

Contents

Acknowledgement	iii
1. Introduction	1
1.1. Motivation	1
1.2. Challenges	3
1.3. Contributions	4
1.4. Thesis Outline	6
2. Background and Technical Details	7
2.1. Semantic Web	7
2.2. Resource Description Framework (RDF)	8
2.3. RDF Database	11
2.3.1. DBpedia	11
2.3.2. BSBM	12
2.4. Big Data	12
2.4.1. Apache Spark	13
2.4.2. SANSa	16
2.4.3. GraphX	17
2.5. Clustering	18
2.5.1. Power Iteration Clustering	19
2.5.2. BorderFlow Clustering Algorithm	21
2.5.3. FH-BF Clustering Algorithm	23
2.5.4. Link-based Clustering algorithm	24
2.5.5. Similarity measures	26
2.6. Silhouettes: a method for evaluating of clustering algorithms	26
3. Related Works and Contributions	29
3.1. Clustering of RDF data	29
3.2. Similarity Measure	31
4. Methodology	33
4.1. Problem Description	33
4.2. Design Goals	33

Contents

4.3. Datasets Inspection	35
4.4. Converting RDF data to Graph data	35
4.5. Similarity Measures	36
4.5.1. Challenges in Similarity Measures and its Solution	39
4.6. Techniques to Implement the RDF data clustering Algorithms	43
4.6.1. PowerIteration	44
4.7. Link-based Clustering of RDF Datasets	44
4.8. BorderFlow and FH-BF Clustering	45
4.9. Techniques to Implement Silhouette Evaluation	47
4.10. Storage Unit	48
5. Evaluation	49
5.1. Cluster configuration	49
5.2. Datsets	50
5.3. Comparing Clustering Algorithms by Performing on Special DBpedia Datasets	51
5.4. Evaluation Results for Similarity functions	58
5.5. Evaluation Results for Illustrating Scalability	59
6. Conclusions and Future Work	61
A. Appendix	63
References	64

List of Figures

2.1. The Semantic Web Technology Stack	8
2.2. Triples in different forms, all of them equivalent	9
2.3. Example of a RDF graph representation of set of triples describing a property. (Source by: https://www.researchgate.net/)	10
2.4. The diagram illustrates the relationships between DBpedia Ontol- ogy, it's parts, DBpedia, and the world it describes	12
2.5. 20 Free Big Data Sources Everyone Should Know	13
2.6. The Apache Spark- the full stack	14
2.7. Distributed and partitioned RDD	15
2.8. SANSA Layers	16
2.9. SANSA Stack	17
2.10. Clustering result and the embedding provided by v^t for the 3Circles dataset. (Sources: by Frank Lin and William W. Cohen [37])	19
4.1. Design Goal Diagram	34
4.2. A graph (Karate Club of Zachary) with thirty four nodes and two communities (Sources: by Evans, T.S., Lambiotte, R. [19])	39
5.1. Mountainbike - Motor bout	53
5.2. Mountainbike - Motor bout	53
5.3. Mountainbike - Motor bout	53
5.4. Mountainbike - Motor bout	54
5.5. Daughter - in - law - Son - in - law	54
5.6. Daughter - in - law - Son - in - law	54
5.7. Daughter - in - law - Son - in - law	54
5.8. Daughter - in - law - Son - in - law	55
5.9. Taxi-Taxi Driver	55
5.10. Taxi-Taxi Driver	55
5.11. Taxi-Taxi Driver	55
5.12. Taxi-Taxi Driver Jaccard	56
5.13. Toothbrush-Toothpaste	56
5.14. Toothbrush-Toothpaste	56
5.15. Toothbrush-Toothpaste	56

List of Figures

5.16. Toothbrush-Toothpaste	57
5.17. HairStylist-TaxiDriver	57
5.18. HairStylist-TaxiDriver	57
5.19. HairStylist-TaxiDriver	57
5.20. HairStylist-TaxiDriver	58
5.21. Comparing some similarity measures	58
5.22. Evaluation algorithm for PowerIteration Clustering	59

List of Algorithms

1.	The PIC algorithm	20
----	-----------------------------	----

1. Introduction

1.1. Motivation

The amount of structured data available on the Web has been increasing rapidly over the last few years. Plain texts and multimedia information such as graphics, audio or video have become a major part within the web's information traffic. Traditional search engines have the limits to understanding the semantics of the data. The Semantic Web¹ is an extension of the current web which refers to a Web of data that is readable and processable by machines. The Semantic Web² standardized by W3C aims at providing a common framework that allows data to be shared and analyzed across applications. The Resource Description Framework (RDF) is one major component of this vision and represents the universal graph-based data model for publishing and exchanging data in the Semantic Web.

The RDF data model expresses statements about resources in the form of subject-predicate-object triples. The subject denotes a resource; the predicate expresses a property (of the subject) or a relationship (between subject and object); the object is either a resource or literal.

The most popular use case for RDF on the Web is Linked Data [7], it has a large collection of different knowledge bases, which are represented in RDF. DBpedia is one of the popular knowledge base [5] consists of 1,445,000 persons, 735,000 places and 241,000 organizations. Objective of DBpedia is to extract structured content from wikipedia and store it into a single integrated dataset.

Since the RDF is increasingly adopted to model data, managing large volumes of such datasets becomes essential. In addition, the increasing size of these RDF datasets implies more and more often the use of distributed context.

The aim of this research is to propose a framework, called Distributed RDF Clustering Framework, which is adopted for efficiently storing and partitioning of big data with focusing on RDF graph technology.

In this literature, data are difficult to capture, store, manage, and analyse using traditional data management tools. The one main challenge in developing a

¹<https://www.w3.org/2001/sw/>

²<http://www.scientificamerican.com/article/the-semantic-web/>

1. Introduction

distributed RDF system is how to split up the data across multiple servers. The clustering is one of the important data mining issue especially for big data analysis, where large volume data should be grouped. Here, we consider a fundamental task of clustering, which assigns similar individuals into the same group, where similarity depends on the features available for each individual as well as the function applied on them and discuss clustering techniques suitable for RDF data. Over the last years, a large number of approaches have been developed to achieve the goal of clustering graphs with high accuracy. Specifically we consider here three clustering algorithms, PowerIteration, BorderFlow and Link-based, for clustering of RDF data.

Generally, conventional clustering algorithms can be divided into two categories: hierarchical clustering such as Link-based, and partitional clustering such as PowerIteration. Hierarchical clustering algorithms produce a hierarchy of nested clusters, whereas partitional clustering algorithms directly output a one-level clustering solution.

PowerIteration (PIC) is a fast spectral clustering technique, (PIC) [37] is not only simple (it only requires a matrix vector multiplication process), but is also scalable in terms of time complexity, $\mathcal{O}(n)$. PIC is an efficient clustering method which can be used in the graph based semi-supervised learning community.

BorderFlow is a local graph clustering algorithms that allow overlapping clusters. It uses solely local information for clustering and achieves a soft clustering of the input graph. We have chosen BorderFlow because it is parameter free and has already been used successfully in diverse applications including clustering protein-protein interaction (PPI) data [45] and query clustering for benchmarking.

Link-based algorithm has been selected among those available in literature for the capability of reconciling the principles of overlapping communities and hierarchy as aspects of the same phenomenon.

BorderFlow and Link-based are two local graph clustering algorithms that allow overlapping clusters. By applying a simple hardening approach to clustering results of BorederFlow, clustered RDF data without overlaps are created. Here, we address the problem of improving the runtime of the algorithm. We introduce a new partitioning clustering algorithm which is based on the BorderFlow approach, the so-called FH-BF, which is more faster than BorderFlow.

To get the weighted graph from an RDF dataset we apply a similarity measure to the algorithm. A review, or even a listing of all the convenient similarity measures is impossible. Here, we use the feature model semantic similarity measures such as Jaccard, Ratio Model, Rodriguez-Egenhofer and Batet which are suitable for calculating the similarity between resources in RDF datasets.

The quantitative evaluation of these algorithms was carried out by using the

Silhouette evaluation algorithm [53]. It refers to a method of interpretation and validation of consistency within clusters of data.

The clustering algorithms of our framework are implemented in Scala. Scala is a powerful language supported by Spark. Scala is functional, statically typed and object oriented language. To process the big datasets, parallel processing systems like Spark and Hadoop are needed. Apache Spark is a big data framework which have been designed and matured over the years. Apache Spark offers a proven and reliable method for general-purpose processing of big datasets. It is a state-of-the-art distributed in-memory analytic framework allows iterative workflow but does not support semantic technology standards. Spark has been benchmarked against Hadoop's MapReduce and ran 100 times faster³.

In this thesis, our research method consists of writing an implementation of an RDF data clustering framework. The algorithms are evaluated using Silhouette evaluation algorithm and running on DBpedia and BSBM datasets. Elements of efficiency and quality can be tested by running on datasets with obvious structure. Also, scalability can be tested by running on big datasets for PowerIteration algorithm and Silhouette evaluation algorithm.

In our implementation for PowerIteration, we apply a pseudo similarity measure. This is because the candidate semantic similarity measures are not working well for big datasets. This pseudo similarity approximates the Jaccard similarity for highly overlapping communities can exhibit more external than internal connections.

Our practical experiments for PowerIteration clustering and Silhouette evaluation show that they are scalable and efficient algorithms based on Spark, and they are working well on big datasets. On the other hand, one problem that then arises is the scalability of BorderFlow and Link-based approaches. Our observation shows that these clustering algorithms are iterative recursive, and cannot be performed in parallel by using the Spark methods. Hence, several directions for further improvements should be offered, among of them a new clustering algorithm based on Link-based algorithm is presented.

Additionally, some datasets are prepared from DBpedia to test the quality of the candidate clustering algorithms manually. Some results are established about performance, quality, and accuracy of the clustering algorithms.

1.2. Challenges

The amount of data in the web is growing day by day. Data is growing because of use of internet, smart phone and social network. A growing amount of data from

³<http://spark.apache.org>

1. Introduction

diverse domains is being converted into RDF as demonstrated by the growth of the Linking Open Data Cloud. With this conversion come a significant number of complex applications which rely on large amounts of data in RDF to perform demanding tasks. Our goal is to handle massive RDF datasets.

One way to overcome the challenges that are raised with big data is to have big data clustered in a compact format that is still an informative version of the entire data. In fact, clustering is a knowledge discovery tool for analyzing big data. The goal of graph clustering is to divide the nodes of the graph into several subsets while minimising the number of links with endpoints in different subsets. Thus, by partitioning RDF data using graph clustering methods, one increases the chance that highly interconnected nodes are placed on the same server. One major challenge in using clustering algorithms is scalability of such algorithms. Notice that traditional clustering techniques cannot cope with this huge amount of data because of their high complexity and computational cost. In order to exploit this large amount of data, efficient clustering algorithms are needed.

In general, effectively and efficiently distributing data structures in potentially complex machine learning approaches are two major challenges in this layer. The graph like representation and multimodal nature of RDF makes the mining techniques more challenging.

In this research project we create a framework (Distributed RDF Clustering Framework) adopting for implementing and analysing some different clustering algorithms required for processing Big Data with RDF graph technology. Some strategies for big data clustering are also presented and discussed.

The two main challenges in developing a distributed RDF system are

- how to split up the data across multiple servers,
- how to deploy clustering techniques to big data and get the results in a reasonable time.

To achieve the goal of processing large datasets, we leverage existing big data frameworks, like Apache Spark [24]. It is designed for analytic tasks where a large portion of the dataset must be read and processed to produce the output.

When we work with big datasets, we use Spark to parallelize the computations to make the process for big data faster.

1.3. Contributions

This research work is an attempt to create a framework (Distributed RDF Clustering Framework) established for implementing and analysing some different clus-

tering algorithms required for processing large RDF datasets. Our framework composes of some clustering methods: PowerIteration, BorderFlow, FH-BF and finally Link-based algorithms. Here we focus on RDF data graphs. To get the weighted graph from an RDF dataset we apply a similarity measure to the algorithm. We use the feature model semantic similarity measures such as Jaccard, Ratio Model, Rodriguez-Egenhofer and Batet which are suitable for calculating the similarity between resources in RDF datasets. Moreover, we apply Silhouette algorithm to evaluate the efficiency of the candidate clustering algorithms. During this thesis, the following are proposed:

1. Implementing PowerIteration, BorderFlow (for directed and undirected graphs) and Link-Based algorithms.
2. Designing and implementing FH-BF clustering that is a new clustering algorithm based on BorderFlow.
3. Implementing Hardening approach and Silhouette evaluation algorithm.
4. Designing a new clustering algorithm based on Link-based algorithm.
5. Comparing the quality, efficiency and scalability of candidate clustering algorithms by performing on special datasets with different sizes.
6. Implementing four semantic similarity measures (Jaccard, Ratio Model, Rodriguez-Egenhofer and Batet).
7. Analysing and comparing the influence of these similarity measures on the candidate clustering algorithms.
8. Introducing and implementing a pseudo similarity measure (Tina).
9. Extracting datasets with obvious clusters from DBpedia.

These experiments lead us to point several observations:

1. BorderFlow and Link-based are useful for RDF clustering but they are not working well in parallel by using the Spark methods. Because they are recursive processes.
2. The new algorithm FH-BF is much more faster than BorderFlow (Hard and Soft).
3. The candidate similarity measures (Jaccard, Ratio Model, Rodriguez-Egenhofer and Batet) are suitable for RDF clustering but they are not working well for big datasets whenever we use the Spark methods.

1. Introduction

4. The new pseudo similarity (Tina) is faster than other similarities and works well on large datasets.
5. PowerIteration clustering and Silhouette evaluation algorithm are scalable and efficient algorithms in Spark.

1.4. Thesis Outline

This thesis is organized as follows. Chapter 2 provides some background on Semantic Web, RDF, Big Data, DBpedia, clustering algorithms (such as PowerIteration, BorderFlow and Link-based), computational framework Apache Spark and GraphX, Silhouette evaluation, etc. It is also described the related work and the motivation for our work in Chapter 3. Chapter 4 is associated with the various methodologies that are implemented to obtain the results. The evaluation and results of our thesis are explained in Section 5. Finally, we conclude our work in Chapter 6. It is also described future work that could be done to make more efficient, more experiments with different datasets.

2. Background and Technical Details

In order to understand better the domain of this project this chapter will present the background concepts and technical details used in this work.

2.1. Semantic Web

The Semantic Web¹ is an extension of the Web that provides common formats for the interchange of data. It is a common language for recording how data relates to real world objects and allowing a person or a machine to start off in one database. Moreover, it provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries”.

The main purpose of the Semantic Web is to allow both machines and humans to treat and manage data that can be found in the Web. Plain texts and multimedia information such as graphics, audio or video have become a major part of the Web. But, how to find some useful information within this huge information space? Traditional search engines have the limits to understanding information content. For managing and processing data in Web, it is important that data pieces are available in a standard format.

The term “Semantic Web” also refers to W3C’s vision of the Web of linked data. Linked data is central to the concept of the Semantic Web. Linked data are empowered by technologies such as RDF, SPARQL, OWL, and SKOS. The common example of Linked data is DBpedia database that provides the data in RDF triples in various formats like Ntriples, Turtle, XML, JSON, etc. We are using DBpedia database to extract datasets.

Tim Berners-Lee² defines the Semantic Web as ”a web of data that can be processed directly and indirectly by machines”. It is based on machine-readable information and builds on XML technology’s capability to define customized tagging schemes and Resource Description Framework’s (RDF) flexible approach to

¹<https://www.w3.org/2001/sw/>

²<https://en.wikipedia.org/wiki/Semantic-Web#cite-note-3>

2. Background and Technical Details

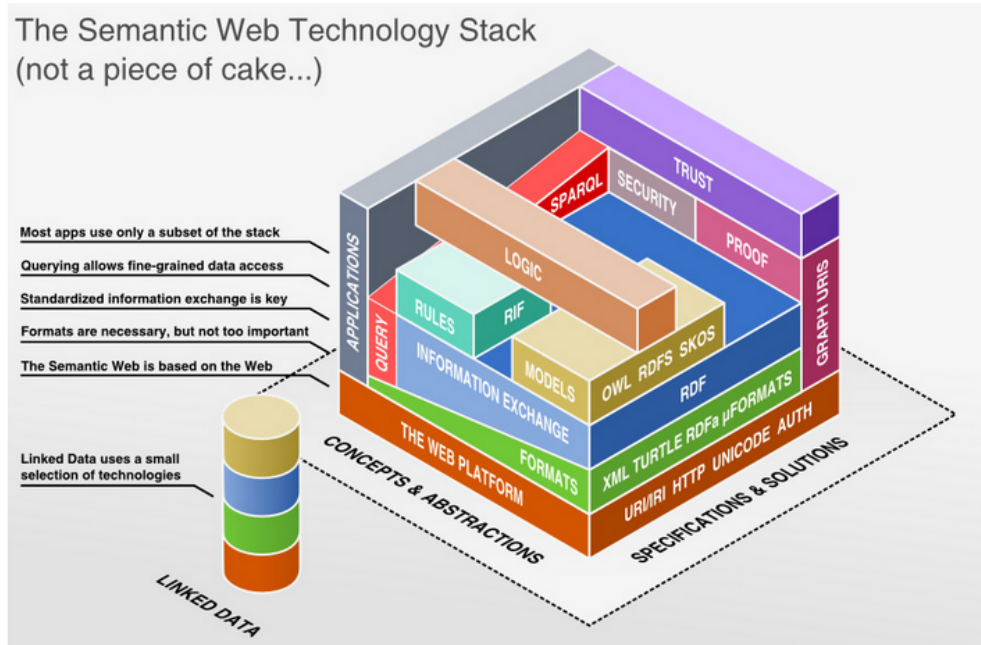


Figure 2.1.: The Semantic Web Technology Stack

representing data. The goal is to enable distributed information resources to be connected and to allow uniform access to make the best use of knowledge world-wide.

The Semantic Web should bring structure to the content of Web pages in which information is given a well defined meaning. Recently, different applications based on this Semantic Web vision such as knowledge management, information integration, community web portals, e-learning, multimedia retrieval, etc. have been designed. The Semantic Web relies on formal ontologies that provide shared conceptualizations of specific domains and on metadata defined according these ontologies.

In Figure 2.1, the Semantic Web Technology Stack is illustrated³.

2.2. Resource Description Framework (RDF)

The Resource Description Framework (RDF) defines a metadata data model intended for representing information about web resources. A resource can refer to any conceptual thing, such as a Web site, a person or a device. It is a general method for conceptual description or modeling of information that is implemented

³<https://de.wikipedia.org/wiki/Semantic>


Graphical form	
Triple	subject predicate object
Relational form	predicate(subject,object)
RDF/XML	<pre> <rdf:Description rdf:about="subject"> <ex:predicate> <rdf:Description rdf:about="object"/> </ex:predicate> </rdf:Description> </pre>
Turtle	subject ex:predicate object.

Figure 2.2.: Triples in different forms, all of them equivalent

in web resources, using a variety of syntax notations and data serialization formats. RDF provides a general, flexible method to decompose any knowledge into small pieces, called triples, with some rules about the semantics (meaning) of those pieces, each consisting of a subject, a predicate and an object. The subject denotes the resource, and the predicate denotes traits and expresses a relationship between the subject and the object. These triples define an RDF graph, in which classes, instances of such and values are represented by nodes and properties are represented by directed edges, see Figures 2.2 and 2.3.

For example, "Tina studies in Bonn University" can be stored as an RDF statement in a triple store and it describes the relationship between the subject, "Tina", and the object, "Bonn University". The predicate "studies" shows how the subject and the object are connected.

The RDF integrates a variety of applications from library catalogs and world-wide directories to syndication and aggregation of news, software, and content to personal collections of music, photos, and events using XML as an interchange syntax.

The RDF is a general-purpose language that provides a means for publishing both human-readable and machine-processable vocabularies. Vocabularies are the set of properties, or metadata elements, defined by resource description communities.

Developed under the guidance of the World Wide Web Consortium, RDF was designed to allow developers to build search engines that rely on the metadata and to allow Internet users to share Web site information more readily. This mechanism for describing resources is a major component in the W3C's Semantic Web activity. RDF uses the key concepts including Graph data model, URI-based vocabulary, Datatypes, Literals and XML serialization syntax.

2. Background and Technical Details

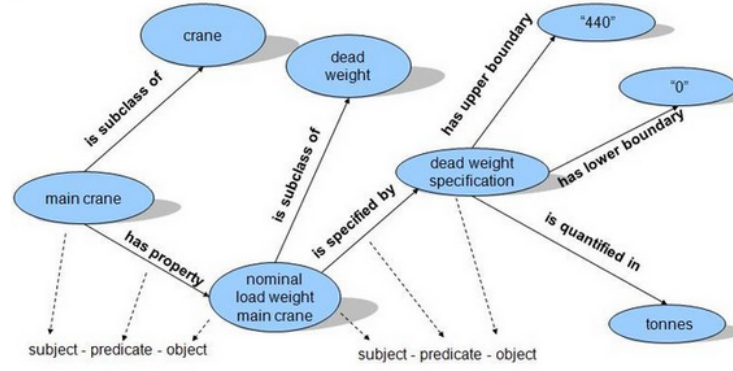


Figure 2.3.: Example of a RDF graph representation of set of triples describing a property. (Source by: <https://www.researchgate.net/>)

The RDF Data Model

The RDF data model is similar to classical conceptual modeling approaches such as entity–relationship or class diagrams. It provides a model for describing resources. Resources have properties (attributes or characteristics). RDF defines a resource as any object that is uniquely identifiable by an Uniform Resource Identifier (URI). The properties associated with resources are identified by property-types, and property-types have corresponding values. Property-types express the relationships of values associated with resources. A collection of these properties that refers to the same resource is called a description. At the core of RDF is a syntax-independent model for representing resources and their corresponding descriptions.

The RDF graph model is used as a basis for uniformly expressing information across different fields and domains. A collection of RDF statements intrinsically represents a labeled, directed multi-graph. This in theory makes an RDF data model better suited to certain kinds of knowledge representation than are other relational or ontological models.

RDF Data-Graph

RDF data sets are given as (subject-predicate-object) triples and typically are represented as directed edge-labeled graphs. To be more precise, based on [25], an *RDF data-graph* $G^d = (V^d, E^d, \ell^d)$ is defined as a tuple where:

- (a) $V^d = V_r \cup V_\ell$ is a finite set of resource (V_r) and literal (V_ℓ) vertices;

- (b) $\ell^d : V^d \times V^d \rightarrow L^d$ is an arc-labeling function returning, for a pair of nodes in V^d , an element in $L^d = L_R \cup L_A \cup \{type\}$, that is a finite set of relation (L_R) and attribute (L_A) labels;
- (c) $E^d \subseteq V^d \times V^d$ is a finite set of directed edges of type $\ell(v_1, v_2)$, where
 - (i) $\ell \in L_R$, if $v_1 \in V_r, v_2 \in V_r$;
 - (ii) $\ell \in L_A$, if $v_1 \in V_r, v_2 \in V_\ell$;
 - (iii) $\ell = \{type\}$, if $v_1 \in V_r, v_2 \in V_r$.

2.3. RDF Database

RDF databases are engines that standardize on the SPARQL query language. These databases require a query language more advanced than SQL so as to make possible the semantic querying of data to bring the world closer to the concept of the Semantic Web. SPARQL is not only efficient in semantic queries, but also in interfacing with the data. RDF databases can do set processing and at the same time do graph processing. In the following some RDF databases are described.

2.3.1. DBpedia

DBpedia is an open, free and exhaustive knowledge base improved and extended by a large global community. It provides a complementary service to Wikipedia⁴ by exposing knowledge. Data is published strictly in line with Linked Data principles using open standards (e.g., URIs, HTTP, HTML, RDF, and SPARQL) and open data licensing.

DBpedia, in fact, is a project that extracts structured, multilingual knowledge from Wikipedia and makes it freely available using Semantic Web and Linked Data standards. DBpedia uses the RDF as a flexible data model for representing extracted information and for publishing it on the Web. The DBpedia project was started in 2006 by the developers of University of Mannheim and University of Leipzig and has meanwhile attracted large interest in research and practice.

Figure 2.4 illustrates the relationships between DBpedia Ontology, its parts, DBpedia, and the world it describes.

DBpedia.org⁵ is a community effort to extract structured information from Wikipedia and to make this information available on the Web. DBpedia allows

⁴<https://www.wikipedia.org>

⁵<https://wiki.dbpedia.org/>

2. Background and Technical Details

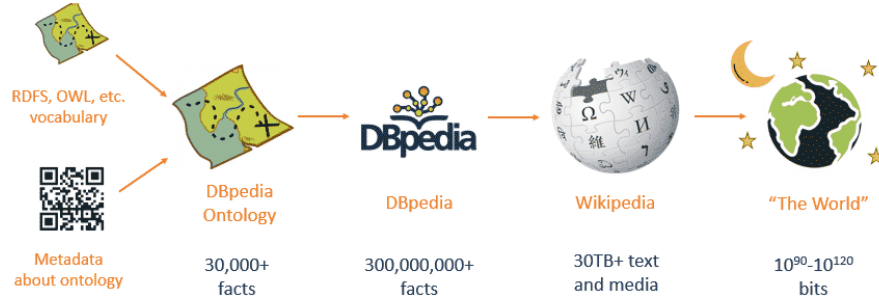


Figure 2.4.: The diagram illustrates the relationships between DBpedia Ontology, its parts, DBpedia, and the world it describes

you to link other datasets on the Web to Wikipedia data. In this thesis we have extracted some datasets from DBpedia.

2.3.2. BSBM

The SPARQL Query Language for RDF, SPARQL Update and the SPARQL Protocol for RDF are implemented by a growing number of storage systems and are used within enterprise and open web settings. As SPARQL is taken up by the community there is a growing need for benchmarks to compare the performance of storage systems that expose SPARQL endpoints via the SPARQL protocol. Such systems include native RDF stores, Named Graph stores, systems that map relational databases into RDF, and SPARQL wrappers around other kinds of data sources. The Berlin SPARQL Benchmark (BSBM)⁶ defines a suite of benchmarks for comparing the performance of these systems across architectures. The benchmark is built around an e-commerce use case in which a set of products is offered by different vendors and consumers have posted reviews about products. In our thesis, we have extracted some datasets from BSBM.

2.4. Big Data

With the growth of the web, the use of social networks, mobile, connected and communicating objects, information is growing faster every day. Big Data is referring to extremely large data sets that may be analysed computationally to reveal patterns, trends, and associations, especially relating to human behaviour and interactions. Big data can be characterized by three V's: volume (large amounts of

⁶<http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/>



Figure 2.5.: 20 Free Big Data Sources Everyone Should Know

data), variety (includes different types of data), and velocity (constantly accumulating new data) [62]. Data become big when their volume, velocity, or variety exceed the abilities of IT systems to store, analyse, and process them.

Big Data encompasses unstructured and structured data, however the main focus is on unstructured data [13]. Unstructured data are the data that either do not have a pre-defined data model or are not organized in a pre-defined manner. Structured data are relatively simple and easy to analyse, because usually the data reside in databases in the form of columns and rows. The challenge for scientists is to develop tools to transform unstructured data to structured ones. Big data requires a set of techniques and technologies with new forms of integration to reveal insights from datasets that are diverse, complex, and of a massive scale.

There are many applications of big data: business, technology, telecommunication, medicine, health care, and services, bioinformatics (genetics), science, e-commerce, finance, the Internet (information search, social networks), etc. Research on the effective usage of information and communication technologies for development (also known as ICT4D) suggests that big data technology can make important contributions but also present unique challenges to International development. In Figure 2.5, you can see some of the best free big data sources available today.

2.4.1. Apache Spark

Apache Spark⁷ is a unified analytics engine for large-scale data processing. Spark is used at a wide range of organizations to process large datasets. The main feature

⁷<https://spark.apache.org/news/>

2. Background and Technical Details

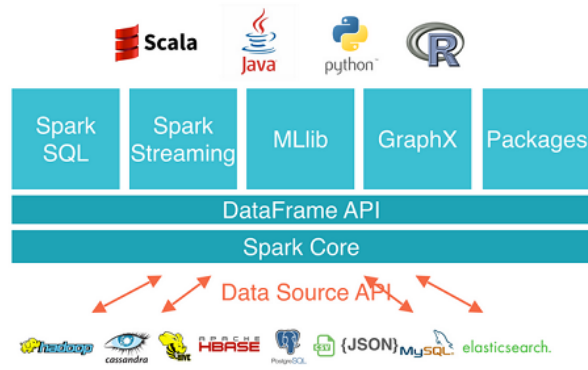


Figure 2.6.: The Apache Spark- the full stack

of Spark is its in-memory cluster computing that increases the processing speed of an application.

In fact, Apache Spark is an open-source cluster-computing framework that originally started by Matei Zaharia in 2009 at the University of California, Berkeley's AMPLab, and open sourced in 2010 under a BSD license. It provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Moreover, it is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing. Spark and its RDDs were developed in 2012 in response to limitations in the MapReduce cluster computing paradigm, which forces a particular linear dataflow structure on distributed programs: MapReduce programs read input data from disk, map a function across the data, reduce the results of the map, and store reduction results on disk. Spark's RDDs function as a working set for distributed programs that offers a (deliberately) restricted form of distributed shared memory [70]. Spark is one of the most active projects in the Apache Software Foundation and one of the most active open source big data projects.

Internet powerhouses such as Netflix, Yahoo, and eBay have deployed Spark at massive scale. It has quickly become the largest open source community in big data.

Apache Spark consists of Spark Core and a set of libraries. The core is the distributed execution engine and the foundation of the overall project. The Java, Scala, and Python APIs offer a platform for distributed ETL application development. Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming, see Figure 2.3. Spark SQL is a component on top of Spark Core that introduced a data abstraction called

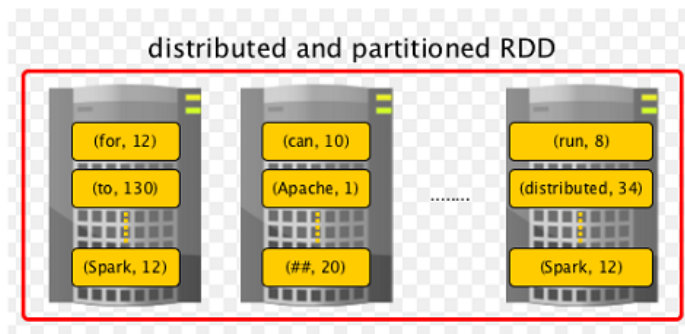


Figure 2.7.: Distributed and partitioned RDD

DataFrames, which provides support for structured and semi-structured data. Spark SQL provides a domain-specific language (DSL) to manipulate DataFrames in Scala, Java, or Python. It also provides SQL language support, with command-line interfaces and ODBC/JDBC server. Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, and S3. You can run Spark using its standalone cluster mode, on EC2, on Hadoop YARN, or on Apache Mesos.

You can see the full-stack of Apache Spark in Figure 2.6⁸.

Resilient Distributed Dataset (RDD)

Resilient Distributed Dataset (RDD) is the primary data abstraction in Apache Spark and the core of Spark. It is an immutable collection of objects which computes on the various nodes of the cluster. RDD is large collection of data or an array of reference of partitioned objects. Each and every datasets in RDD is logically partitioned across many servers so that they can be computed on different nodes of cluster. RDDs are fault tolerant i.e. self-recovered/recomputed in case of failure.

The features of RDDs:

1. Resilient, i.e. fault-tolerant with the help of RDD lineage graph and so able to recompute missing or damaged partitions due to node failures.
2. Distributed with data residing on multiple nodes in a cluster.

⁸<https://www.safaribooksonline.com/library/view/learning-spark/9781449359034/ch01.html/>

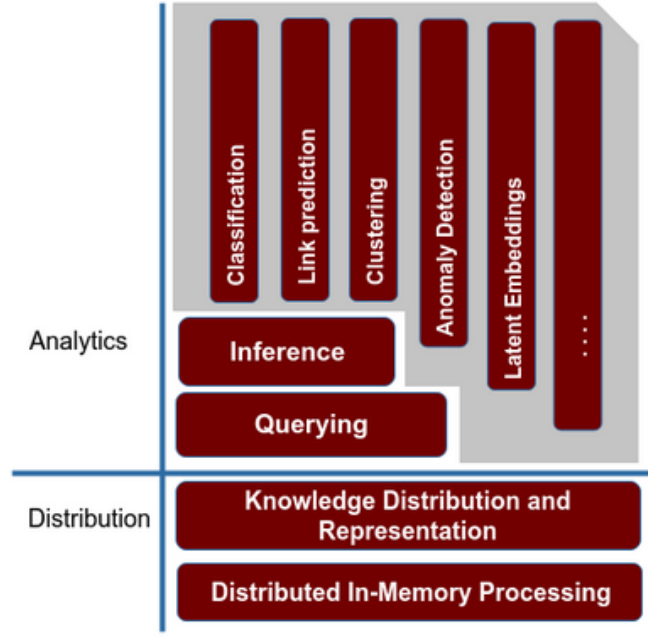


Figure 2.8.: SANSA Layers

3. Dataset is a collection of partitioned data with primitive values or values of values, e.g. tuples or other objects (that represent records of the data you work with).

Figure 2.7 illustrates the distributed and partitioned RDD⁹.

2.4.2. SANSA

SANSA [35] is an open-source structured data processing engine for performing distributed computation over large-scale RDF datasets¹⁰. It provides data distribution, scalability, and fault tolerance for manipulating large RDF datasets, and applying analytics on the data at scale by making use of cluster-based big data processing engines. SANSA is easily integrated with well-known open source systems both for data input and output (HDFS) and is build on top of Spark and Flink. SANSA layers is illustrated in Figure 2.8.

Our thesis work will be an addition for Semantic Analytics Stack(SANSA)¹¹, a distributed computing frameworks with the semantic technology stack. A layer

⁹<https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-rdd.html>

¹⁰sansa-stack.net/

¹¹<https://github.com/SANSA-Stack/SANSA-ML/tree/develop/sansa-ml-spark/src/test/scala/net/sansa-stack/ml/spark/clustering>

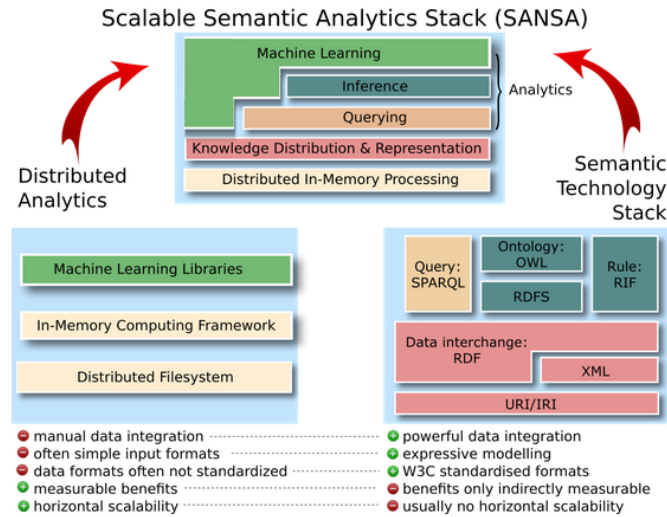


Figure 2.9.: SANSA Stack

named as "Clustering" in SANSA Stack will help to cluster the RDF data, see Figure 2.9.

2.4.3. GraphX

GraphX¹² is the graph processing layer on top of Spark that brings the power of Big Data processing to graphs. GraphX is the new Spark API for graphs such as Web-Graphs and Social Networks, and graph-parallel computation. GraphX extends the Spark RDD abstraction by introducing the Resilient Distributed Property Graph: a directed multigraph with properties attached to each vertex and edge. To support graph computation, GraphX exposes a set of fundamental operators e.g., subgraph, joinVertices, and mapReduceTriplets. In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks.

The goal of the GraphX project is to unify graph-parallel and data-parallel computation in one system with a single API. The GraphX API enables users to view data both as graphs and as collections (i.e., RDDs) without data movement or duplication. Moreover, it is able to optimize the execution of graph operations.

The property graph is a directed multigraph with user defined objects attached to each vertex and edge. A directed multigraph is a directed graph with potentially multiple parallel edges sharing the same source and destination vertex. The prop-

¹²<https://spark.apache.org/graphx/>

2. Background and Technical Details

erty graph is parameterized over the vertex (VD) and edge (ED) types. These are the types of the objects associated with each vertex and edge respectively. GraphX optimizes the representation of vertex and edge types when they are primitive data types (e.g., int, double, etc. . .) reducing the in memory footprint by storing them in specialized arrays.

2.5. Clustering

Clustering is an unsupervised learning task aiming at partitioning a collection of objects into subsets or clusters, so that those within each cluster are more closely related to one another than to the objects assigned to different clusters [1]. It is a main task of data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, information retrieval, and bioinformatics. Clusters can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. Clustering methods are based on the application of similarity (or density) measures, defined over a fixed set of attributes of the domain objects. An object is usually described by a fixed set of feature (attribute values) and the most common notion of similarity between the objects is expressed in terms of a distance function based on this set. The notion of a cluster cannot be precisely defined, which is one of the reasons why there are so many clustering algorithms, thus it varies significantly in its properties [18]. Typical cluster models include:

1. Connectivity models: for example, hierarchical clustering builds models based on distance connectivity.
2. Centroid models: for example, the k-means algorithm represents each cluster by a single mean vector.
3. Distribution models: clusters are modeled using statistical distributions, such as multivariate normal distributions used by the expectation-maximization algorithm.
4. Density models: for example, DBSCAN and OPTICS defines clusters as connected dense regions in the data space.
5. Graph-based models: a clique, that is, a subset of nodes in a graph such that every two nodes in the subset are connected by an edge can be considered as a prototypical form of cluster. Relaxations of the complete connectivity

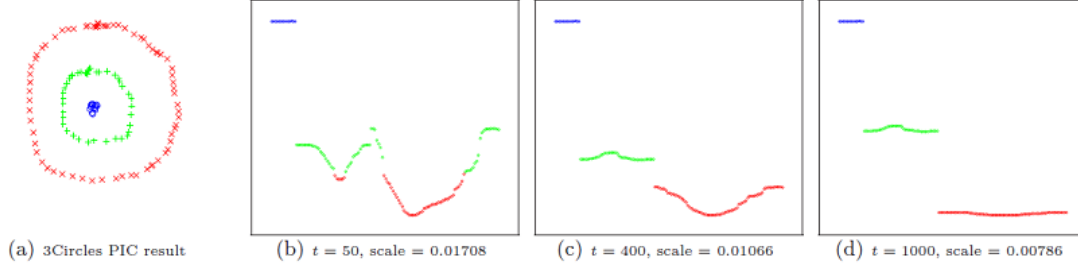


Figure 2.10.: Clustering result and the embedding provided by v^t for the 3Circles dataset. (Sources: by Frank Lin and William W. Cohen [37])

requirement (a fraction of the edges can be missing) are known as quasi-cliques, as in the HCS clustering algorithm.

Clusterings can be roughly distinguished as:

- **Strict partitioning clustering:** each object belongs to exactly one cluster;
- **Overlapping clustering:** objects may belong to more than one cluster;
- **Hierarchical clustering:** objects that belong to a child cluster also belong to the parent cluster.

Many real-world clustering problems involve data objects of multiple types that are related to each other, such as Web pages, search queries, and Web users in a Web search system. In such scenarios, traditional clustering methods may not work well. Additionally, there are difficulties for applying traditional clustering techniques to big data due to some challenges, including high complexity and computational cost, inconsistency in data flow and responding quickly to data velocity, that are raised with big data. In this thesis, we study and compare some data clustering algorithms which are suitable for RDF graphs with focusing on big datasets.

2.5.1. Power Iteration Clustering

Spectral clustering is one of the most popular clustering algorithms, however, it requires the use of computing eigenvectors, making it time consuming. To overcome this drawback, Lin and Cohen [37] proposed PowerIteration clustering (PIC) technique which is a simple and fast version of spectral clustering. It finds a very low-dimensional data embedding using truncated power iteration on a normalized pairwise similarity matrix of the data points, and this embedding turns out to

2. Background and Technical Details

Algorithm 1: The PIC algorithm

Data: A row-normalized affinity matrix W and the number of clusters k

Result: Clusters C_1, C_2, \dots, C_k

Pick an initial vector v^0 ;

while $|\delta^t - \delta^{t-1}| \simeq 0$ **do**

 Set $v^{t+1} \leftarrow \frac{Wv^t}{\|Wv^t\|_1}$ and $\delta^{t+1} \leftarrow |v^{t+1} - v^t|$;

 Increment t ;

end

Use k -means to cluster points on v^t

be an effective cluster indicator. Instead of finding the eigenvectors, PIC finds only one pseudo-eigenvector, which is a linear combination of the eigenvectors in linear time. PIC is a partitioning clustering algorithm which directly output a one-level clustering solution. PIC is not only simple, it only requires a matrix-vector multiplication process, but it is also scalable in terms of time complexity, $O(n)$.

Given a dataset $X = \{x_1, \dots, x_n\}$, a similarity function $s(x_i, x_j)$, an affinity matrix $A \in R^{n \times n}$ defined by $A_{ij} = s(x_i, x_j)$ and a diagonal matrix D defined by $d_{ii} = \sum_j A_{ij}$, we take a normalized affinity matrix W which is defined as $D^{-1}A$. One simple method for computing the largest eigenvector of a matrix is power iteration (PI). PI is an iterative method, which starts with an arbitrary vector $v_0 \neq 0$ and repeatedly performs the update $v_{t+1} = cWv_t$ where W is the normalized affinity matrix and c a normalizing constant to keep v_t from getting too large. PI stops for some small number of iterations t , after it has converged within clusters but before final convergence. This leads to an approximately piecewise constant vector, where the elements that are in the same cluster have similar values. Let us define the *velocity* at t to be the vector $\delta^t = v^t - v^{t-1}$ and define the *acceleration* at t to be the vector $\epsilon^t = \delta^t - \delta^{t-1}$. We pick a small threshold $\hat{\epsilon}$ and stop PI when $\|\epsilon^t\|_\infty \leq \hat{\epsilon}$. This is best illustrated by an example from [37]. Figure 2.1(a) shows a simple dataset-i.e., each x_i is a point in R^2 space, with $s(x_i, x_j)$ defined as $\exp(\frac{-\|x_i - x_j\|^2}{2\sigma^2})$.

In Figures 2.8(b) to 2.8(d) shows v^t at various values of t , each illustrated by plotting $v^t(i)$ for each x_i .

2.5.2. BorderFlow Clustering Algorithm

BorderFlow [44] is a novel local graph clustering algorithm for directed and undirected weighted graphs. It was designed initially for the computation of semantic classes out of large term-similarity graphs. For each node v , the algorithm begins with an initial cluster X containing only v . Then, it expands X iteratively by adding nodes from the direct neighborhood of X to X until X is node-maximal with respect to a function called the border flow ratio. The same procedure is repeated over all nodes. Notice that different nodes can lead to the same cluster, hence identical clusters should be collapsed to one cluster. Then the set of collapsed clusters and the mapping between each cluster and its nodes are returned as result. In this algorithm, the standard definition of clusters is that they have a maximal intra-cluster density and outer-cluster sparseness. When considering a graph as the description of a flow system, this definition of a cluster implies that a cluster X can be understood as a set of nodes such that the flow within X is maximal while the flow from X to the outside is minimal. To be more precise, let $G = (V, E, \omega)$ be a weighted directed graph with a set of vertices V , a set of edges E and a weight function ω , which assigns a positive weight $\omega(e) \in \mathbb{R}^+$ to each edge $e \in E$. Non-existing edges e are considered to be edges such that $\omega(e) = 0$. Let $X \subset V$ be a set of nodes. We denote by $i(X)$ the set of inner nodes, $b(X)$ of border nodes and $n(X)$ of direct neighbors of X . For two subsets X and Y of V , we define $\Omega(X, Y)$ as the total weight of the edges from X to Y (i.e., the flow between X and Y), i.e. $\Omega(X, Y) = \sum_{x \in X, y \in Y} \omega(xy)$. The border flow ratio $F(X)$ of $X \subseteq V$ is then defined as follows:

$$F(X) = \frac{\Omega(b(X), X)}{\Omega(b(X), V \setminus X)} = \frac{\Omega(b(X), X)}{\Omega(b(X), n(X))}. \quad (2.1)$$

The aim of BorderFlow is to compute non-trivial local maximums of $F(X)$. Accordingly, for each node $v \in V$, BorderFlow computes a set X of nodes that maximize the ratio $F(X)$ with respect to the following maximality criterion:

$$\begin{aligned} \forall X \subset V, F(X) \text{ maximal} &\Leftrightarrow \forall X' \subset V, \forall v \notin n(X), \\ X' = X + v &\Rightarrow F(X') < F(X). \end{aligned}$$

The computation of the cluster X for $v \in V$ begins with $X = \{v\}$. Then, X is expanded iteratively. Each of these iterations is carried out in two steps.

During step 1, the set $C(X)$ of candidates $u \in n(X)$ which maximize $F(X + u)$ is computed as follows: $C(X) := \arg \max_{u \in n(X)} F(X + u)$.

In step 2, BorderFlow picks the candidates $u \in C(X)$ which maximize the flow

2. Background and Technical Details

$\Omega(u, n(X))$. The final set of candidates $C_f(X)$ is thus

$$C_f(X) := \arg \max_{u \in C(X)} \Omega(u, n(X)). \quad (2.2)$$

The elements of $C_f(X)$ are added to X if and only if the condition $F(X \cup C_f(X)) \geq F(X)$ is satisfied. The insertion of nodes by the means of steps 1 and 2 is iterated until $C(X) = \emptyset$. X is then returned as the cluster for v . The clustering procedure is repeated over all nodes of V .

A heuristic for maximizing the border flow ratio

The implementation proposed above demands the simulation of the inclusion of each node in $n(X)$ into the cluster X for computing $F(X)$. Such an implementation can be time-consuming as nodes in PPI graphs can have a high number of neighbors. We can show that for a node $v \in n(X)$, maximizing $\Delta F(X, v) = F(X + v) - F(X)$ can be approximated by maximizing:

$$f(X, v) = \frac{\Omega(b(X), v)}{\Omega(v, V \setminus X)}. \quad (2.3)$$

By setting $C(X) := \arg \min_{u \in n(X)} 1/f(X, u)$, BorderFlow can be implemented efficiently.

Hardening

Since BorderFlow has its tendency to generate many overlapping clusters then hardening approach can be applied to BorderFlow's results. Hence, we perform the hardening algorithm to post-process BorderFlow's results. Let C_1, \dots, C_η be the clusters computed by BorderFlow. The hardening of the results of BorderFlow is as follows:

1. Discard all clusters C_j such that $\exists C_i : C_j \subset C_i$.
2. Order all remaining C_j into a list $L = \{\lambda_1, \dots, \lambda_m\}$ in descending order with respect to the number of seeds that led to their computation.
3. Discard all $\lambda_z \in L$ with $z > k$, with k being the smallest index such that the union of all λ_i with $i \leq k$ equals V .
4. Re-assign each v to the cluster C such that $\Omega(v, C)$ is maximal.
5. Return the new clusters.

Since the hardening approach can be applied to both the node-optimal and the heuristic version of the algorithm, we will distinguish the following four versions of BorderFlow in the remainder of this thesis: *OS* (Optimal, Soft), *OH* (Optimal, Hard), *HS* (Heuristic, Soft) and *HH* (Heuristic, Hard).

2.5.3. FH-BF Clustering Algorithm

A drawback of BorderFlow-soft is its tendency to generate many overlapping clusters and consuming lots of time to make the cluster for each node. Furthermore, in BorderFlow-hard we apply a hardening approach to post-process BorderFlow's results which takes more time than BorderFlow-soft. Here, we address the problem of improving the runtime of the algorithm. In this thesis, we introduce a new clustering algorithm which is based on BorderFlow, so-called FH-BF, which is much more faster than BorderFlow. Also it is a partitioning algorithm.

When considering a graph as the description of a flow system, this definition of a cluster implies that a cluster X can be understood as a set of nodes such that the flow within X is maximal while the flow from X to the outside is minimal. Let $G = (V, E, \omega)$ be a weighted graph with a set of vertices V , a set of edges E and a weighing function ω , which assigns a positive weight to each edge $e \in E$. For each $v \in V$, let $Y(v)$ be the set of direct neighbors of v . As we have explained before, for each node v , BorderFlow algorithm begins with an initial cluster X containing only v . Then, it expands X iteratively by adding nodes from the direct neighborhood of X to X until X is node-maximal with respect to the border flow ratio. The idea behind FH-BF is to start clustering with the node v so that the total weight of the edges from v to its neighbor set Y (i.e. the flow between $X = \{v\}$ to Y) is maximized. To be more precise, if we consider $\Omega(v, Y(v)) = \sum_{y \in Y(v)} \omega(v, y)$, where $Y(v)$ is the set of all border nodes of v , then $\Omega(v, Y(v)) = \max_{u \in V} \Omega(u, Y(u))$. The node that maximizes the total weight can generate the main cluster for undirected graphs. Therefore those nodes which are belong to this cluster can eliminate from the list of input nodes.

Let N_1, \dots, N_n be the neighbors of node $v \in V$ collected by Graphx and let C_1, \dots, C_η be the clusters computed by FH-BF. Then FH-BF algorithm is described as follows:

1. Compute the following total weight for each node $v \in V$,

$$\Omega(v) = \sum_{k=1}^n \text{Sim}(v, N_k).$$

2. Order all $v \in V$ into a list $L = \{v_1, \dots, v_m\}$ in descending order with respect

2. Background and Technical Details

to their total weights.

3. Use BorderFlow to cluster nodes v_1, \dots, v_m , respectively.
4. In each iteration j discard all vertices v_i such that $v_i \in C_j$ from list L .
5. Return clusters C_1, \dots, C_η .

By converting RDF data to weighted graphs we use this algorithm for clustering RDF data. In FH-BF algorithm like as BorderFlow, we assign a weight to each edge in graph by using of similarity measures. In this thesis, we have implemented FH-BF algorithm for undirected graphs and with the candidate similarity measures.

2.5.4. Link-based Clustering algorithm

Here, we explain a hierarchical link clustering method from [2] and [25] that classifies links into topologically related groups. Link information plays an important role in discovering knowledge from data. Relational data are those that have link information among the data items in addition to the classic attribute information for the data items. When we say Link-based clustering, we mean the clustering of relational data. In other words, links are the relations among the data items or objects. Link-based clustering (see [25] and [2]) is a hierarchical method that creates a hierarchical decomposition of the given set of data objects forming a dendrogram which is a tree that splits the database recursively into smaller subsets. The bottom up approach of hierarchical called the “agglomerative” approach, starts with each object forming a separate group. It successively merges the objects or groups according to some measure and this is done until all of the groups are merged into one, or until a termination condition holds. The definition of similarity between links adopted in [2] considers only the local network topology as available information, i.e., the neighborhood of a node is employed in the similarity evaluation. In particular, it is computed only the similarity between pairs of links that share a node k .

The similarity between two links sharing a node k can be evaluated with the Jaccard index. To be more precise, for any node i , we define the inclusive neighbors of a node i as:

$$n_+(i) = \{x : d(i, x) \leq 1\} \quad (2.4)$$

where $d(i, x)$ is the length of the shortest path between nodes i and x . The set simply contains the node itself and its neighbors. From this, the similarity S

between links can be given by, e.g., the Jaccard index:

$$S(e_{ik}, e_{jk}) = \frac{|n_+(i) \cap n_+(j)|}{|n_+(i) \cup n_+(j)|}. \quad (2.5)$$

Each link is initially assigned to a different cluster; then, the pair of links with the largest similarity are merged, until all links are members of the same community. Single-linkage hierarchical clustering builds a link dendrogram from the link density equation below. Cutting this dendrogram at some clustering threshold—for example the threshold with maximum partition density (see below) yields link communities. Hence, we can find a partition $\mathcal{P} = \{P_1, \dots, P_m\}$ of the set of links in the RDF graph which optimizes a link similarity criterion such as partition density. Starting from partition \mathcal{P} , a set \mathcal{C} of m node clusters is obtained, putting in each cluster C_i , $i \in \{1, \dots, m\}$, nodes incident to each link contained in partition P_i .

Partition density

For a network with M links and N nodes, $\mathcal{P} = \{P_1, \dots, P_C\}$ is a partition of the links into C subsets. Let $m_c = |P_c|$ be the number of links of the c -th cluster of \mathcal{P} , and n_c be the number of nodes induced by links in P_c . Note that $\sum_c m_c = M$ and $\sum_c n_c \geq N$ (assuming no unconnected nodes). The link density D_c of community P_c is

$$D_c = \frac{m_c - (n_c - 1)}{n_c(n_c - 1)/2 - (n_c - 1)}. \quad (2.6)$$

This is the number of links in P_c , normalized by the minimum and maximum numbers of links possible between those nodes, assuming they remain connected. (We assume $D_c = 0$ if $n_c = 2$.) The partition density, \mathcal{D} , is the average of D_c , weighted by the fraction of present links:

$$\mathcal{D} = \frac{2}{M} \sum_c m_c \frac{m_c - (n_c - 1)}{(n_c - 2)(n_c - 1)}. \quad (2.7)$$

Notice that the partition density is an objective function measuring the quality of a link partition in terms of links density inside communities, and it is evaluated at each level of the link dendrogram. It is then cut at the level identified by the maximum value of \mathcal{D} .

2.5.5. Similarity measures

The concept of similarity is fundamentally important in almost every scientific field. A *similarity measure* or *similarity function* is a real-valued function that quantifies the similarity between two objects. A review, or even a listing of all the uses of similarity is impossible. To get the weighted graph from an RDF dataset we apply different similarity measures to the algorithms including Jaccard Similarity, Batet Similarity, Rodriguez and Egenhofer Similarity, Tversky's Ratio Model Similarity. In Chapter 4, these similarities are discussed [29].

2.6. Silhouettes: a method for evaluating of clustering algorithms

Silhouette [53] refers to a method of interpretation and validation of consistency within clusters of data. The average silhouette width provides an evaluation of clustering validity, and might be used to select an appropriate number of clusters. The silhouette value is a measure of how similar an object is to its own cluster compared to other clusters.

Here, we have used Silhouette evaluation for estimating validity of the clusters computed by using the different clustering methods.

Take any node i in the data set, and denote by A the cluster to which it has been assigned and let C be any cluster which is different from A . We compute the average similarity of the node i with all other objects of A

$$a(i) = \text{average similarity of } i \text{ to all other nodes of } A.$$

Analogously, we compute the average similarity of i with all objects of C

$$s(i, C) = \text{average similarity of } i \text{ to all nodes of } C.$$

After computing $s(i, C)$ for all clusters $C \neq A$, we select the biggest of those numbers and denote it by $b(i) = \max_{C \neq A} s(i, C)$. It should be done for each similarity measure listed above. Let B be the cluster for which this maximum is attained (that is, $s(i, B) = b(i)$). This is like the second-best choice for node i . We now define a silhouette:

$$s(i) = \begin{cases} 1 - b(i)/a(i) & \text{if } a(i) > b(i) \\ 0 & \text{if } a(i) = b(i) \\ a(i)/b(i) - 1 & \text{if } a(i) < b(i) \end{cases}$$

From the above definition it is easy to see that $-1 \leq s(i) \leq 1$.

2.6. Silhouettes: a method for evaluating of clustering algorithms

If $s(i)$ is close to 1 then we can say that i is well-clustered. A different situation occurs when $s(i)$ is about zero. Then $a(i)$ and $b(i)$ are approximately equal, and hence it is not clear at all whether i should have been assigned to either A or B . The worst situation takes place when $s(i)$ is close to -1 . Then $b(i)$ is much larger than $a(i)$, so i lies on the average much closer to B than to A so we can almost conclude that this node has been misclassified.

3. Related Works and Contributions

We give here a brief of existing works in clustering of RDF large-scale datasets. Moreover, we present the significant research contributions in this field. There exist several dimensions of related work. We structure our discussion along the presentation of our contributions:

3.1. Clustering of RDF data

The Web as a global information space is developing from a Web of documents to a Web of data. With the growth of Semantic Web technologies, the considerable amount of RDF datasets being made available on the Web, hence Web clustering has an important role in Semantic Web area. First RDF clustering techniques presented in literature rely on traditional data-clustering methods [21]. Although there are two challenges for applying the traditional clustering mechanisms to Semantic Web resources:

1. Traditional clustering methods are based on instances not on a large interconnected graph as described by RDF; How do we extract a representation of an instance from such a RDF graph?
2. Having extracted the instances, how do you compute the distance between two such instances?

The survey [10] explains and compares various clustering techniques applied to web documents. Most of them are text-clustering algorithms, including an efficient Suffix-Tree Clustering (STC) [72]. STC has been improved and become a basis for Lingo [46], both used in Carrot framework [47] and its commercial successor Carrot Search. While these algorithms show good results on corpus of data of average size, they have certain limitations common for all of them.

A different approach for clustering RDF statements exploiting the ontological knowledge was proposed by Delteil et al. in [14]. However, it is not applicable for real Semantic Web data, often noisy, incomplete and inconsistent. Many graph clustering methods are solely based on the network structure [50]. Communities in networks often overlap such that nodes simultaneously belong to

3. Related Works and Contributions

several groups. Hence, one can reinvent communities as groups of links rather than nodes. Link communities exist [2] in many networks, including major biological networks such as protein–protein interaction and metabolic networks. Hence, a link-based approach is superior to node-based approaches. In [25] an analysis of existing RDF clustering methods for data-mining and management with focus on link-based methods has been conducted.

There exist some local graph clustering algorithms allow for clusters to share nodes. Such algorithms are called non-partitioning approaches [61]. Non-partitioning approaches can be applied for clustering of communities in networks. Ngonga Ngomo and Schumacher [44] presented a non-partitioning novel local graph clustering algorithm called BorderFlow, which was designed especially to compute a soft clustering of graphs. BorderFlow has been applied in several domains including concept location in software development [57] and query clustering for benchmarking [41].

Spectral clustering is one of the most popular clustering algorithms, however, it requires the use of computing eigenvectors, making it time consuming. To overcome this drawback, Lin and Cohen [37] proposed PowerIteration clustering (PIC) technique which is a simple and fast version of spectral clustering. PIC is a scalable and efficient clustering method which can be used in the graph-based semi-supervised learning community. Although the PowerIteration method approximates only one eigenvalue of a matrix, it remains useful for certain computational problems. For instance, Google uses it to calculate the PageRank of documents in their search engine, and Twitter uses it to show users recommendations of who to follow.

Here, we propose a framework, Distributed RDF Clustering Framework, which is useful for RDF graph mining and adopted for clustering and partitioning of RDF data. Our framework composed of three clustering methods: PowerIteration, the BorderFlow and finally Link-based algorithms. To achieve the goal of clustering of RDF graphs with high accuracy and link communities, we consider BorderFlow and Link-based algorithms. BorderFlow and Link-based are two local graph clustering algorithms that allow overlapping clusters.

Still, with the growth of the size of the dataset at hand, improving the runtime of graph clustering becomes an increasingly urgent problem.

Our observation shows that BorderFlow and Link-based algorithms are recursive, this means that each step of both algorithms depend on the outputs of the previous step. Hence, we cannot perform them in parallel by using the Spark methods. Hence, several directions for further improvements should be offered, among of them a new algorithm based on Link-based algorithm is presented.

3.2. Similarity Measure

The choice of distance measure is the most important factor for a clustering algorithm. In traditional clusterings we use of geometric distance measures, such as Euclidean or Manhattan distance. However, standard distance metrics do not apply to RDF datasets.

Montes-y-Gómez et. al developed a similarity measure for comparing two conceptual graphs [40]. Conceptual graphs are a data structure commonly used for natural language processing. Conceptual graphs and RDF graphs are structurally sufficiently similar hence this similarity metric is also appropriate for comparing RDF graphs. You can see [40] and [28] for details on applying this metric to RDF data.

There have been several efforts for developing an ontology based similarity measure for RDF data. Most of them are focussing on similarity between whole ontologies. Ontologies are widely adopted in the biomedical domain to characterize various resources. Semantic similarity measures are used to estimate the similarity of concepts defined in ontologies. They are used in a wide array of applications: to design information retrieval algorithms [65, 31], to suggest drug repositioning [27] and to cluster genes according to their molecular function [48]. Dieng and Hug [16] compared two ontologies using an approach based on conceptual graphs. Maedche and Zacharias developed a similarity measure for instance data, with emphasis on the ontological background information [38].

Because the growth of Semantic Web and Linked Data paradigms, semantic similarity measures have recently been the subject of considerable interest [7].

The feature-based approaches are also ontology-based semantic similarity measures which focus on the evaluation of concepts. They are semantic similarities in the context of distance measures (e.g. set-based measures). Tversky formulated a framework of semantic similarity, the feature model, from which a family of semantic measures can be derived [66]. The feature model requires the semantic objects to be represented as sets of features. Their similarity is therefore intuitively defined as a function of their common features, an approach commonly used to compare sets (e.g. Jaccard index). He defined a parameterized semantic similarity measure (the ratio model) which can be used to compare two semantic objects through their respective sets of features. The other feature-based measure is given by Rodriguez and Egenhofer [52]. Batet et al. assess taxonomic distance as the ratio between distinct and shared features [55].

Since RDF is the standard language for the representation of semantic information, it seems that the feature model semantic similarity measures proposed by Tversky can be suitable for RDF datasets. Each RDF dataset can be converted to

3. *Related Works and Contributions*

a RDF data-graph. This motivates us to define the modified versions of some semantic similarities (e.g. Jaccard similarity, Ratio Model similarity, Rodríguez and Egenhofer similarity and Batet similarity). In this thesis, we use these modified versions of semantic similarities for clustering of RDF datasets.

When we work with big datasets, we use GraphX library to parallelize the computations to make the process for big data faster. It is Apache Spark's API for graphs and graph-parallel computation. To apply these similarities, we need to find the union, intersection and subtraction of neighbouring sets of any two vertices. For implementing these similarities in Spark, we must apply Collect or Join Operator. These operators do not work well for big datasets. Hence, we define a pseudo similarity measure to avoid union, intersection and subtraction between two sets.

4. Methodology

4.1. Problem Description

The Resource Description Framework (RDF) is a universal graph-based data model for publishing and exchanging data in the Semantic Web. The growth of Semantic Web technologies has led to the publication of large volumes of data. To process these large amounts of data, it is necessary to use powerful tools for knowledge discovery. The main goal of this project is an attempt to create a framework (Distributed RDF Clustering Framework) which is adopted for implementing and analysing some different clustering algorithms required for processing big datasets with RDF graph technology. To process the big datasets, we use parallel processing systems like Spark and Hadoop.

Here, three examples for different clustering algorithms are considered, namely PowerIteration, BorderFlow and Linked-Based algorithms. They all take an RDF graph as input and return the list of triples for each of the different clusters. To get the weighted graph from an RDF dataset we apply a suitable similarity measure to the algorithm. Elements of efficiency and complexity can be tested by running on special datasets.

4.2. Design Goals

Our goal here is to propose a framework, Distributed RDF Clustering Framework for implementing and analysing some different clustering algorithms required for processing big RDF datasets

The Resource Description Framework (RDF) is a data model for representing information about web resources. A resource is any physical or conceptual thing, such as a Web site, a person or a device. RDF provides a method to decompose any knowledge into small pieces, called triples, with some rules about the semantics (meaning) of those pieces. These triples define an RDF graph. In a big RDF dataset there is maybe millions or billions of triples in a file.

Our framework can be divided into three major parts: PowerIteration clustering which is a simple and fast version of spectral clustering; BorderFlow, a

4. Methodology

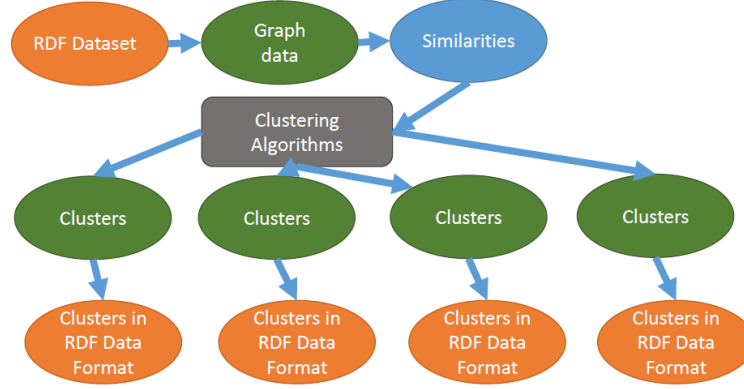


Figure 4.1.: Design Goal Diagram

general-purpose graph clustering algorithm which uses solely local information for clustering and achieves a soft clustering of the input graph and finally the Link-based clustering which is designed to cluster the edges of an RDF graph instead of its nodes. All these clustering algorithms have been implemented by different similarity measures including the Jaccard similarity measure, the Rodriguez and Egenhofer similarity measure, the Ratio similarity measure and the Batet similarity measure.

Our datasets have extracted from DBpedia and BSBM. Apache Spark helps to process a big dataset by distributing it across the cluster. In the following, you observe the main tasks of our framework.

Distributed RDF Clustering Framework

Tasks:

- 1- Extracting the RDF datasets from basese DBpedia and BSBM.
 - 2- Converting RDF data to graph data.
 - 3- Applying similarity measures to get the weighted graphs.
 - 4- Implementing clustering algorithms.
 - 5- Transforming the graph data in clusters to RDF data.
-

4.3. Datasets Inspection

The preparation of a dataset is a crucial and challenging task. Here, some RDF datasets are prepared from DBpedia and BSBM to test the quality, efficiency and scalability of the candidate clustering algorithms. A RDF dataset consists of triplets (subject, predicate, and object) in the form of URIs. The datasets have obvious clusters as input to evaluate the accuracy of clustering algorithms. The required datasets here are Taxi-Taxidriver with 661 triples, Toothpaste-Toothbrush with 104 triples, Hairstylist-Taxidriver with 516 triples, Daughter-in-law-Son-in-law with 203 triples and Mountainbike-Motorboat with 379 triples.

DBpedia is an open, free and comprehensive knowledge base constantly improved and extended by a large global community. It provides a complementary service to Wikipedia by exposing knowledge. Data is published strictly in line with "Linked Data" principles using open standards (e.g., URIs, HTTP, HTML, RDF, and SPARQL) and open data licensing. The selection of DBpedia over others is based on analyzing the patterns in the database.

BSBM is an open source and publicly available benchmarks which compares the performance of storage systems that expose SPARQL endpoints via the SPARQL protocol. It includes native RDF stores, Named Graph stores, systems that map relational databases into RDF, and SPARQL wrappers around other kinds of data sources. The Berlin SPARQL Benchmark (BSBM)¹ defines a suite of benchmarks for comparing the performance of systems across architectures.

Here, we have merged two files to get the required datasets. We have extracted two datasets from DBpedia by writing the query in <http://dbpedia.org/sparql> to obtain a dataset with shape of star to star graph (for instance Taxi and Taxidriver have merged with each other to obtain the Taxi-Taxidriver dataset). You can see the query which we have used for creating the Taxi dataset in Appendix A.

4.4. Converting RDF data to Graph data

The Resource Description Framework (RDF) specifies a language for defining data items and relationships, by using a graph representation, with the intent. RDF is a common acronym within the semantic web community because it forms one of the basic building blocks for forming the web of semantic data and defines a type of database the so-called a graph database.

An RDF dataset consists of triplets, i.e., subject, predicate, and object in the

¹<http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/>

4. Methodology

form of URIs. We need to convert the RDF data to Graph data to make it suitable for the algorithms. Graph data are numerical, therefore the volume of data is less than the RDF data.

Also it is easiest to illustrate RDF data in the form of a simple graph. For this purpose, we consider each triple as an edge. We define subject and object as the vertices of this edge. For example, representation of connection between Tooth-brush and Toothbrush as URI in Dbpedia is

```
(http://dbpedia.org/resource/Tooth_brush,  
http://dbpedia.org/ontology/wikiPageRedirects,  
http://dbpedia.org/resource/Toothbrush)
```

We convert

```
"http://dbpedia.org/resource/Tooth_brush"
```

to "0" and

```
"http://dbpedia.org/resource/Toothbrush"
```

to "1" and we consider (0,1) as an edge.

Another advantage for using Graph data is GraphX. We need to use GraphX library to parallelize the graph computations to make the process for big data faster. GraphX is Apache Spark's API for graphs and graph-parallel computation.

4.5. Similarity Measures

A similarity measure or similarity function is a real-valued function that quantifies the similarity between two objects in a dataset. In fact, a similarity function assigns a real number between 0 and 1 to the objects in a dataset. A zero value means that the objects are dissimilar completely whereas one indicates that the objects are identical practically. The selection of an appropriate similarity measure for a specific dataset is a challenging task. Is it possible to distinguish families of measures sharing specific properties? How can one identify the most appropriate measures according to particular criteria?

Here, we consider semantic similarities. They are used to estimate the similarity of concepts defined in ontologies and, hence, to assess the semantic proximity of the resources indexed by them. Among of them we take feature-based approaches. They are ontology-based semantic similarity measures which focus on the evaluation of concepts. Feature-based approaches are semantic similarities in the context of distance measures (e.g. set-based measures).

Tversky was the first to formulate a framework of semantic similarity, the feature model, from which a family of semantic measures can be derived [66]. The feature model requires the semantic objects to be represented as sets of features. Their similarity is therefore intuitively defined as a function of their common features, an approach commonly used to compare sets (e.g. Jaccard index). He defined a parameterized semantic similarity measure (the ratio model) which can be used to compare two semantic objects (u, v) through their respective sets of features U and V :

$$\text{Sim}_{RM}(u, v) = \frac{f(u \cap v)}{\alpha f(u \setminus v) + \beta f(v \setminus u) + f(u \cap v)} \quad (4.1)$$

with $\alpha, \beta \geq 0$. Note that, considering f as the cardinality of the set, setting Sim_{RM} with $\alpha = \beta = 1$ leads to the Jaccard index. Hence, a semantic object can be represented as a set of features and that commonalities and differences must be evaluated to assess the similarity. Therefore, the ratio model similarity is constrained in the set-based frame and it relies on an undefined function f , specifying how to capture the sets of features of compared objects.

These observation ensure that feature model semantic similarity measures proposed by Tversky can be suitable for RDF datasets. Notice that RDF is the standard language for the representation of semantic information: it encodes Web data as a labeled directed graph in which the nodes represent resources and links represent semantic relationships between them. Each RDF dataset can be converted to a RDF data-graph. This motivates us to define the modified versions of some semantic similarities (e.g. Jaccard similarity, Ratio Model similarity, Rodríguez and Egenhofer similarity and Batet similarity) that are explained below. In this thesis, we use these modified versions semantic similarities for clustering of RDF datasets.

Jaccard Similarity

For any two nodes u and v of a data set, the Jaccard similarity is defined as:

$$\text{Sim}_{Jaccard}(u, v) = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}, \quad (4.2)$$

where $N(u)$ is the subset of all neighbors of the node u and $|N(u)|$ the cardinality of $N(u)$ that counts how many elements are in $N(u)$.

Rodríguez and Egenhofer similarity

Another example of feature-based measure is given by Rodríguez and Egenhofer [52]. It can be modified as:

$$\text{Sim}_{RE}(u, v) = \frac{|N(u) \cap N(v)|}{\gamma \cdot |N(u) \setminus N(v)| + (1 - \gamma) \cdot |N(v) \setminus N(u)| + |N(u) \cap N(v)|} \quad (4.3)$$

with $\gamma \in [0, 1]$, a parameter that enables to tune measure symmetry. Again, we take $N(u)$ the set of all neighbors of u and let $\gamma = 0.8$.

Ratio Model

The ratio model proposed by Tversky can be modified as:

$$\text{Sim}_{RM}(u, v) = \frac{|N(u) \cap N(v)|}{\alpha |N(u) \setminus N(v)| + \beta |N(v) \setminus N(u)| + |N(u) \cap N(v)|} \quad (4.4)$$

with $\alpha, \beta \geq 0$.

Here, we take $|N(u)|$ the cardinality of the set $N(u)$ composed of all neighbors of u . Setting Sim_{RM} with $\alpha = \beta = 1$ leads to the Jaccard index. In this thesis, we set $\alpha = \beta = 0.5$.

Batet Similarity

Batet et al. assess taxonomic distance as the ratio between distinct and shared features [55]. Batet similarity can be changed as follows:

$$\text{Sim}_{Batet}(u, v) = \log_2(1 + \frac{|N(u) \setminus N(v)| + |N(v) \setminus N(u)|}{|N(u) \setminus N(v)| + |N(v) \setminus N(u)| + |N(u) \cap N(v)|}). \quad (4.5)$$

For example, consider the graph given by Figure 4.2 and take the two nodes (5) and (11) in that graph. Then

$$N(5) = \{1, 7, 11\}, \quad N(11) = \{1, 5, 6\},$$

$$N(5) \cap N(11) = \{1\}, \quad N(5) \cup N(11) = \{1, 5, 6, 7, 11\},$$

$$N(5) \setminus N(11) = \{7, 11\}, \quad N(11) \setminus N(5) = \{5, 6\}.$$

Now, we estimate the different similarities for (5, 11).

$$\text{Sim}_{Jaccard}(5, 11) = 1/5, \quad \text{Sim}_{RE}(5, 11) = 1/3,$$

$$\text{Sim}_{RM}(5, 11) = 1/3, \quad \text{Sim}_{Batet}(5, 11) = 0.848.$$

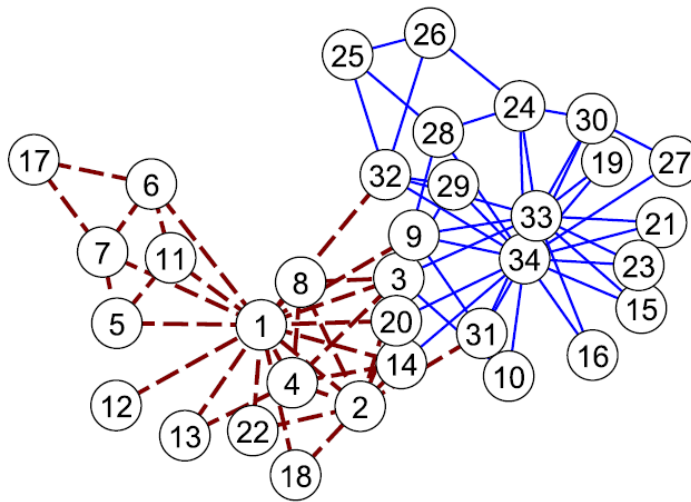


Figure 4.2.: A graph (Karate Club of Zachary) with thirty four nodes and two communities (Sources: by Evans, T.S., Lambiotte, R. [19])

But for the pair (11,26), all similarities are equal to 0, because these two nodes have not any common neighbor and they are in different communities.

4.5.1. Challenges in Similarity Measures and its Solution

In the previous section, we have discussed some suitable similarity techniques can be used for RDF datasets. These similarity measures take two finite sets as input and find the similarity between them. Here, we discuss some of the various challenges faced during the implementation of similarities in Spark and its solution.

Challenge 1

In our setting, a RDF dataset is converted to a data graph. A key step in many graph analytics tasks is aggregating information about the neighborhood of each vertex. For implementing the similarities we need to aggregate the neighbours of each vertex. Aggregating the neighbours can be implemented with some different Spark methods. These methods are Collecting Neighbors, Map Reduce Triplet, Aggregate Messages, GroupByKey and AggregateByKey function.

Collecting Neighbors

In some cases it may be easier to express computation by collecting neighboring vertices and their attributes at each vertex. This can be easily accomplished

4. Methodology

using the `collectNeighborIds` and the `collectNeighbors` operators. But this function can be quite costly as they duplicate information and require substantial communication.

Generally `collect` could be an expensive operation in spark. When a `collect` operation is issued on a RDD, the dataset is copied to the driver, i.e. the master node. A memory exception will be thrown if the dataset is too large to fit in memory.

Map Reduce Triplet

In earlier versions of GraphX neighborhood aggregation was accomplished using the `mapReduceTriplets` operator. The `mapReduceTriplets` operator takes a user defined map function which is applied to each triplet and can yield messages which are aggregated using the user defined reduce function. As the dataset size increases, the size of the RDD also increases, hence `mapReduceTriplets` operator does not work well on large RDDs.

Aggregate Messages

Since Collecting Neighbors and Map Reduce Triplet do not work well on large RDDs, one can use another way for aggregating the neighbours.

The `aggregateMessages` is the operator which applies a user defined `sendMsg` function to each edge triplet in the graph and then uses the `mergeMsg` function to aggregate those messages at their destination vertex.

The `aggregateMessages` operation performs optimally when the messages are constant sized.

GroupByKey

As name suggest `groupByKey` function¹ in Apache Spark just groups all values with respect to a single key. Unlike `reduceByKey` it doesn't perform any operation on final output. It just group the data and returns in a form of an iterator. It is a transformation operation which means its evaluation is lazy.

Now because in source RDD multiple keys can be there in any partition, this function require to shuffle all data with of a same key to a single partition unless your source RDD is already partitioned by key. And this shuffling makes this transformation as a wider transformation.

¹backtobasics.com/big-data/spark/apache-spark-groupby-example/

Looking at spark `groupByKey` function it takes key-value pair (K, V) as an input produces RDD with key and list of values.

Important Points:

1. Apache spark `groupByKey` is a transformation operation hence its evaluation is lazy.
2. It is a wide operation as it shuffles data from multiple partitions and create another RDD.
3. This operation is costly as it doesn't use combiner local to a partition to reduce the data transfer.
4. Not recommended to use when you need to do further aggregation on grouped data.
5. `groupByKey` always results in Hash-Partitioned RDDs.

AggregateByKey

There is a class aimed exclusively at working with key-value pairs, the so-called `PairRDDFunctions` class. When working data in the key-value format one of the most common operations to perform is grouping values by key. While there are many functions in the `PairRDDFunctions`² class, we focus on `aggregateByKey`. It is used to aggregate the values for each key and adds the potential to return a different value type. When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations. Like in `groupByKey`, the number of reduce tasks is configurable through an optional second argument.

The `aggregateByKey` operator can aggregate the `dstIds` in edges which has the same `srcId`. According to our code, to avoid duplicate information, between two efficient Spark operations, `aggregateByKey` and `aggregateMessages`, we have used the `aggregateByKey`.

In the following, you can see the `aggregateByKey` code in Scala.

²codingjunkie.net/spark-agr-by-key/

Listing 4.1: aggregateByKey operation

```
val initialSet = mutable.ArrayBuffer.empty[(VertexId)]

val addToSet = (s: mutable.ArrayBuffer[(VertexId)], v: (VertexId)) => s
    += v

val mergePartitionSets = (p1: mutable.ArrayBuffer[(VertexId)], p2:
    mutable.ArrayBuffer[(VertexId)]) => p1.++=(p2)

val y = x.aggregateByKey(initialSet)(addToSet, mergePartitionSets)
```

Challenge 2

As we have mentioned in the previous section, in our setting, we apply the semantic similarities Jaccard, Ratio Model, Rodríguez-Egenhofer and Batet for RDF datasets. Hence, we need to find the union, intersection and subtraction of neighbouring sets of any two vertices. For this purpose, we have two solutions:

1. Collect

Collect the RDD of vertices and their neighbours by the following code:

```
val y = x.collect()
```

We obtain a RDD x with VertexId as key and value as set of VertexId with the following form

```
Array[(VertexId, Set[VertexId])]
```

This solution is expensive for big data and does not work well on large RDDs.

2. Join

The other method is to join³ data originating from RDDs or external collections that have graphs. Joins in general are expensive since they require that corresponding keys from each RDD are located at the same partition so that they can

³<https://acadgild.com/blog/what-join-apache-spark>

be combined locally. If the RDDs do not have known partitioners, they will need to be shuffled so that both RDDs share a partitioner, and data with the same keys lives in the same partitions. If they have the same partitioner, the data may be colocated. In order to join the data, Spark needs it to be present on the same partition. The default process of join in apache Spark is called a shuffled Hash join. The shuffled Hash join ensures that data on each partition has the same keys by partitioning the second dataset with the same default partitioner as the first. Joining the large size RDD is expensive too and does not work for big dataset. Join the RDD of vertices and their neighbours by the following code:

```
val z = x.join(y).join(y)
```

We obtain a RDD with the following form

```
RDD[(VertexId2, ((VertexId1, (Set1[VertexId], VertexId2)), Set2[VertexId]))]
```

Solution

We provide a new pseudo similarity to avoid union, intersection and subtraction between two sets.

For any two nodes u and v of a data set, the pseudo similarity (Tina) is defined as:

$$\text{Sim}_T(u, v) = \frac{1}{|N(u)| + |N(v)|}, \quad (4.6)$$

where $N(u)$ is the subset of all neighbors of the node u and $|N(u)|$ the cardinality of $N(u)$ that counts how many elements are in $N(u)$. This pseudo similarity measure approximates the Jaccard similarity for highly overlapping communities can exhibit more external than internal connections. In particular, it works well for big data.

4.6. Techniques to Implement the RDF data clustering Algorithms

We have already defined four different clustering algorithm to find the clusters in RDF data. In this chapter, we deal with our implementation, implementation challenges and its Solution.

4.6.1. PowerIteration

The spark.mllib package supports PowerIteration Clustering (PIC) model. Spark.mllib includes an implementation of PIC using GraphX as its backend. It takes an RDD of (srcId: Long, dstId: Long, similarity: Double) tuples which are vertices of an edge and the similarity of these two vertices and outputs a model with clustering assignments. PIC implementation takes the following parameters:

1. k: number of clusters
2. maxIterations: maximum number of power iterations

```
def pic() = {  
    val pic = new PowerIterationClustering()  
        .setK(2)  
        .setMaxIterations(50)  
    pic  
}  
def model = pic.run(RDDofVerticesAndItsSimilarity)  
val modelAssignments = model.assignments
```

PowerIteration (PIC) is a scalable and efficient algorithm for clustering vertices of a graph.

4.7. Link-based Clustering of RDF Datasets

As you have seen before, Link-based clustering (see [25] and [2]) (that is the clustering of relational data) is a hierarchical method that creates a hierarchical decomposition of the data objects forming a dendrogram which is a tree that splits the database recursively into smaller subsets. First, we put each object in a separate group. It successively merges the objects or groups according to some measure and this is done until all of the groups are merged into one, or until a termination condition holds. Hence, in each level new communities come out from the previous level of communities. So, each step of algorithm depends on the outputs of the previous step. Spark framework is used to partition the data into multiple parts so that it can be processed simultaneously in a distributed way. On the other hand, since the algorithm is recursive and uses of an iterative recursive function, we cannot perform it in parallel by using the Spark methods. Therefore, we have just one solution:

Recursive for loop

While running a recursive evaluation loop for a given RDD, with each iteration the RDD lineage grows, which causes a StackOverflow error occur. Thus, this method does not work for big dataset. Notice that RDD is the main abstraction of spark, so we should implement the algorithm with RDDs. By these observations, it seems that some changes in the algorithm is needed.

New Algorithm

Link-based clustering algorithm requires multiple iterations to generate a resulting optimal model. In Spark, the main effort is to perform all different levels of the links dendrogram in parallel. Therefore, the merging of communities in each level should be done independently and simultaneously. For this purpose, we assign a similarity to each pair of edges. We aggregate all these similarities and then assign them to the levels of the links dendrogram as an ID which indicates the different levels. In each level, we merge communities based on their similarities.

To extract the best community structure, it is necessary to cut the dendrogram at a certain level and take the ID of that level as a threshold. Then we consider the communities obtained by merging the objects or groups in that level and all levels that their similarity are bigger than the threshold as our final communities.

Notice that in this new algorithm, due to assigning an ID (similarity ID) to each level, we use the `aggregateByKey` spark method, see Listing 5.1. Moreover, in each level we aggregate all communities which have the same similarity.

We denote by P_i , $i = 1, \dots, k$, the set of links optimizing our link similarity criterion. Then, starting from $\{P_1, \dots, P_k\}$, a set \mathcal{C} of k node clusters is obtained, putting in each cluster C_i , $i \in \{1, \dots, k\}$, nodes incident to each link contained in P_i .

Just the problem which occurs here is the overlapping between the groups of links. This means that it is possible $P_i \cap P_j \neq \emptyset$, for some $i, j \in \{1, \dots, k\}$.

4.8. BorderFlow and FH-BF Clustering

As we have discussed before BorderFlow [44] is a general-purpose local graph clustering algorithm designing for directed and undirected weighted graphs. It uses local information for clustering and achieves a soft clustering of the input graph. The idea behind BorderFlow is to maximize the flow from the border of each cluster to its inner nodes while minimizing the flow from the cluster to the nodes outside of the cluster.

4. Methodology

Here we focus on RDF data graphs and use BorderFlow for clustering directed and undirected RDF data. For this purpose, first we load the RDF data and convert it to a graph data. To assign a weight to each edge in the graph we use similarity measures.

We recall that the algorithm begins with an initial cluster X containing only one node v . Then, it expands X iteratively by adding nodes from the set $n(X)$ of direct neighbors of X to X until X is node-maximal with respect to a function F called the border flow ratio. The same procedure is repeated over all nodes. Notice that different nodes can lead to the same cluster, hence identical clusters should be collapsed to one cluster. Then the set of collapsed clusters and the mapping between each cluster and its nodes are returned as result. Since BorderFlow generates many overlapping clusters, we should use a simple hardening approach to post-process BorderFlow's results which increases the runtime of the algorithm. To improve the runtime of the algorithm, we have introduced a new clustering algorithm FH-BF which is based on BorderFlow-hard.

In both algorithms BorderFlow and FH-BF, due to make the maximum for the border flow ratio $F(X)$, we need to select elements from the neighboring set $n(X)$ of a cluster X iteratively and insert them in X and picks the candidates $u \in C(X)$ which maximise the flow $\Omega(u, n(X))$ to ensure that the inner flow within the cluster is maximised. As the case Link-based algorithm, we aggregate the neighboring set for each vertex ($\text{RDD}[(\text{VertexId}, \text{Set}[\text{VertexId}])]$). Therefore, the only solution to insert elements iteratively in the set X for finding the maximum for the border flow ratio $F(X)$ is to use "for loop" which is not a Spark method. On the other hand, for computing $F(X)$ and $C_f(X)$ we need to compute Ω . It means that we need to have the weight (similarity here) for each element between two sets in Ω , i.e. $b(X)$ and X or $b(X)$ and $n(X)$. Therefore we need to collect the similarities which does not work well on large RDDs. Also, after inserting all elements of $C_f(X)$ in X satisfying the condition $F(X \cup C_f(X)) \geq F(X)$, we should repeat all these steps for any new set X . This process is continue until the condition $F(X \cup C_f(X)) \geq F(X)$ does not hold. Thus, F is a recursive function and we have a recursive algorithm in both cases. On the other hand, we could not use RDD within another RDD transformation and actions can only be invoked by the driver, not inside of other transformation. Finally, for implementing BorderFlow we need to use "collect" and "nested for loop" method for the recursive function. Therefore, by these observations, BorderFlow and FH-BF are not recommended for big data in Spark.

4.9. Techniques to Implement Silhouette Evaluation

A clustering evaluation demands an independent and reliable measure for the assessment and comparison of clustering experiments and results. In this thesis, we implement Silhouette clustering evaluation and use it for all four different clustering algorithms.

Additionally, to providing information about the quality of classification of a single object, the silhouette value can be extended to evaluate the individual clusters and the entire clustering ¹.

In our implementation, Silhouette takes an RDD of (vertexId: Long, clusterId: Int) and output is Silhouette average with Double type.

According to Silhouette evaluation definition, we need to compute the average similarity of the node i with all other objects of A and compute the average similarity of i with all objects of C . Therefore, the implementation divided in two parts and for this purpose the creation of two groups based node i to have the similarities between i with all other objects of A and C is necessary. Create groups efficiently is the difficult part of this implementation. It could be implemented with the help of two different Spark methods. These methods are `groupByKey` and `aggregateByKey` function. Here, we are used the `aggregateByKey`.

GroupByKey

The `groupByKey` is the simplest method but it can cause out of disk problems as data is sent over the network and collected on the reduce workers. In fact, by using the `groupByKey`, all the key-value pairs are shuffled around, causing a lot of unnecessary data to being transferred over the network. When we want to group very large collections over low cardinality keys, `groupByKey` should be avoided.

AggregateByKey

`AggregateByKey` is the most efficient method for our purpose. It aggregate the data inside each partition, only one output for one key at each partition to send over network. The `aggregateByKey` function requires 3 parameters: This function takes 3 parameters as input:

1. initial value;
2. combiner logic;
3. sequence op logic.

¹<http://www.ims.uni-stuttgart.de/institut/mitarbeiter/schulte/theses/phd/algorithm.pdf>

4.10. Storage Unit

A storage system is needed to save the input and output files. Hadoop is ideal for storing large amounts of data and uses Hadoop Distributed File System (HDFS) as its storage system. HDFS lets you connect nodes contained within clusters over which data files are distributed. In this thesis, we use HDFS to store the RDF file. Then by using Spark RDD we store the intermediate results.

5. Evaluation

We have implemented and developed the similarity functions and clustering algorithms as illustrated in Chapter 4. In this chapter, we will compare some different methods of implementation for four techniques of clustering algorithms of RDF graphs and five ways for calculating the similarity measures. The quantitative evaluation of these algorithms will be carried out by using Silhouette evaluation which refers to a method of interpretation and validation of consistency within clusters of data. The metrics which are presented here are:

- comparing the quality, efficiency and scalability of some different techniques of clustering by performing on special datasets with different sizes;
- comparing the quality and scalability of some similarity measures;
- the number of clusters which produce by the candidate clustering algorithms;
- evaluating the clustering algorithms and similarity functions by Silhouette evaluation.

With this evaluation process, the effectiveness of the candidate clustering algorithms and similarity measures is also measured. Our implementation are tested in spark cluster¹ and its configuration details are discussed in the next section.

5.1. Cluster configuration

Our implementation of the candidate clustering algorithms and similarity functions are tested based on Spark and Scala in Spark cluster. We use a Spark Standalone mode with Spark version 2.3.1 and Scala with version 2.11.11. We also use a small cluster of Smart Data Analytics (SDA) which consists of 4 servers. These servers have a total of 256 cores, and each server has Xeon Intel CPUs at 2.3GHz, 256GB of RAM and 400GB of disks space, running Ubuntu 16.04.3 LTS (Xenial) and connected in a Gigabit Ethernet² network. Each Spark executor is assigned a memory of 250GB.

¹<https://en.wikipedia.org/wiki/Computer-cluster>

²<https://en.wikipedia.org/wiki/Gigabit-Ethernet>

5. Evaluation

Our experiments are performed on a Spark client mode with the configuration described in Table 5.1.

Parameter Name	Value
Driver Memory(GB)	4
Executors Memory(GB)	19
Executor Cores (GB)	5
Total Number of Cores	192

Table 5.1.: Spark configuration parameters in cluster mode.

5.2. Datasets

In this thesis, our research method consists of writing an implementation of an RDF data clustering framework. For comparing the quality and accuracy of these clustering methods, an experimental study is offered. This experimental study can also be improved by preparing suitable datasets. Here, the algorithms are running on DBpedia and BSBM datasets. Elements of quality, accuracy and complexity can be tested by running on special datasets. In fact, in our study, we have performed five different specific datasets extracted from DBpedia which have obvious clusters as input. These datasets have different sizes and apply to test and compare the quality and accuracy of the clustering algorithms. In Appendix A, you can see the query which we have used for creating these datasets. In Tables 5.2, 5.3 and 5.4, we describe the required datasets. We have used the large datasets

The Name of Dataset	The Number of Triples
Taxi-Taxidriver	661
Toothpaste- Toothbrush	104
Hairstylist-Taxidriver	516
Daughter- in-law-Son-in-law	203
Mountainbike-Motorboat	379

Table 5.2.: Description of Datasets.

5.3. Comparing Clustering Algorithms by Performing on Special DBpedia Datasets

(see Tables 5.3 and 5.4) to illustrate the scalability of PIC and Silhouette evaluation algorithms. On the other hand, as we have said before, we have introduced a pseudo similarity(Tina) because the candidate similarities defined in Section 4.5 do not work well for big datasets. Furthermore, to compare the candidate similarities with each other and with pseudo similarity, we have selected a dataset from DBpedia: "DBpedia-2016-05 with 30,793 triples".

The Table 5.2 consists of two columns: the name of the database and the number of RDF triples. The Tables 5.3 and 5.4 consist of three columns: dataset categorization, the number of RDF triples and the size of the desired files. This categorization is simply for our convenience so that we can distinguish the datasets easily.

Name	The Number of Triples	Dataset Size
DBpedia-2016-04	30,793	4.2 MB
Small DBpedia	24,670,594	3.7GB
Medium DBpedia	117,544,395	16.6GB

Table 5.3.: Description of DBpedia Datasets.

Name	The Number of Triples	Dataset Size
BSBM-2GB	8,289,484	2.15GB
BSBM-20GB	81,980,472	21.47GB
BSBM-50GB	204,730,067	53.78GB

Table 5.4.: Description of BSBM Datasets.

5.3. Comparing Clustering Algorithms by Performing on Special DBpedia Datasets

In this section, we present our experimental setup and results. It can be done by applying our clustering algorithms on special DBpedia datasets listed in the previous section. The quality and accuracy for four different clustering algorithms with four different similarity functions are compared. The candidate clustering algorithms are Borderflow(Soft), Borderflow(Hard), FH-BF, Linked-based(Silvia)

5. Evaluation

and PowerIteration. Furthermore, we are compared Borderflow(Soft) and Borderflow(Hard) for two cases, the directed and undirected graphs.

In Figures 5.2 - 5.20, we have illustrated the evaluation diagrams³ for the candidate algorithms and similarity measures. These diagrams illustrate that, in the undirected case, Borderflow(Soft), Borderflow(Hard) and FH-BF work better than the rest candidate algorithms. Moreover, our experiment shows that Jaacard, Rodrigue-Egenhofer and Ratio Model similarities in cluster quality are approximately equivalent but they are better than Batet.

For the directed case, the quality of Borderflow(Soft) is better than Borderflow(Hard). But, as you have seen before, a drawback of the BorderFlow(Soft) is its tendency to generate many overlapping clusters. On the other hand, by increasing the size of dataset, the execution time is taken by the clustering algorithms also increases. PowerIteration and FH-BF are taken much less time than the other clustering algorithms.

We recall that the required datasets are Taxi-Taxidriver, Toothpaste-Toothbrush, Hairstylist-Taxidriver, Daughter-in-law-Son-in-law and Mountainbike-Motorboat.

Another metric to evaluate the clustering algorithms is the number of clusters. In PIC, we need to determine the number of clusters as an input parameter. Determining the number of clusters in a data set, is a frequent problem in data clustering, and is a distinct issue from the process of actually solving the clustering problem. The correct choice of the number of clusters (k) is often ambiguous, with interpretations depending on the shape and scale of the distribution of points in a data set and the desired clustering resolution of the user. In addition, increasing k without penalty will always reduce the amount of error in the resulting clustering, to the extreme case of zero error if each data point is considered its own cluster (i.e., when k equals the number of data points). Intuitively then, the optimal choice of k will strike a balance between maximum compression of the data using a single cluster, and maximum accuracy by assigning each data point to its own cluster.

In other clustering algorithms (including BorderFlow, FH-BF and Link-based) the number of clusters are determined by these algorithms. Here, according to the structure of our datasets which have obvious clusters as input and extracted from DBpedia, we expect to have a few number of clusters. Hence, we could say that the algorithms with a few number of clusters have better quality and accuracy. Also, we could determine the number of clusters in PIC based on BorderFlow, FH-BF and Link-based clustering algorithms's results.

Partitioning algorithms including Borderflow(Hard), FH-BF and PowerItera-

³In these diagrams, evaluation have done by Silhouette evaluation algorithm, as a method of interpretation and validation of consistency within clusters of data.

5.3. Comparing Clustering Algorithms by Performing on Special DBpedia Datasets

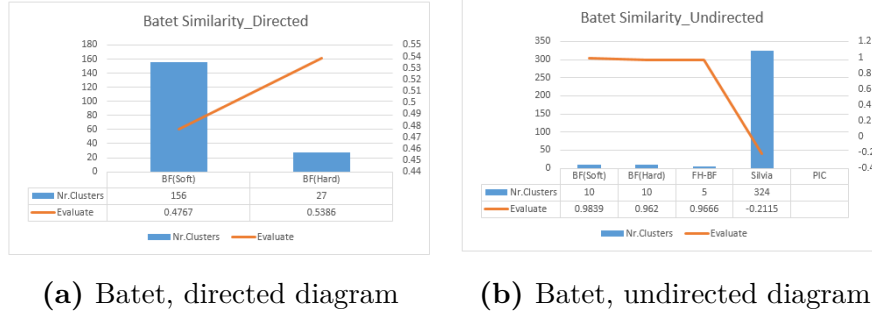


Figure 5.1.: Mountainbike - Motor bout, Batet similarity

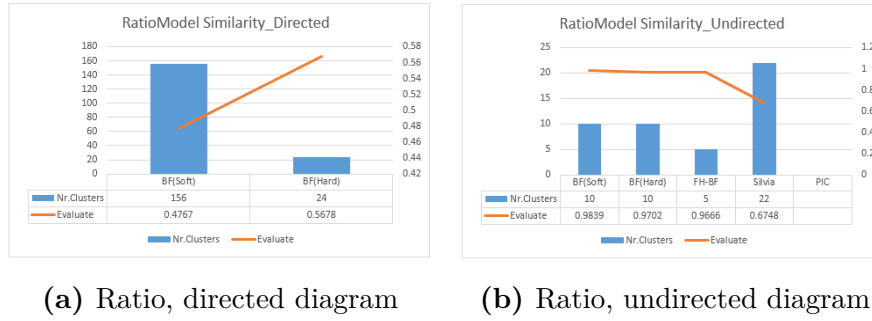


Figure 5.2.: Mountainbike - Motor bout, Ratio similarity

tion make a fewer number of clusters than the algorithms generate overlapping clusters such as BorderFlow(Soft) and Link-based. Between the two algorithms Borderflow(Hard) and FH-BF, FH-BF makes a fewer clusters than BorderFlow.

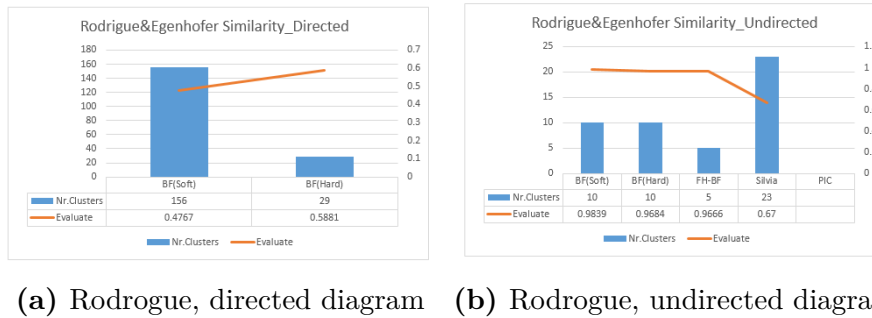
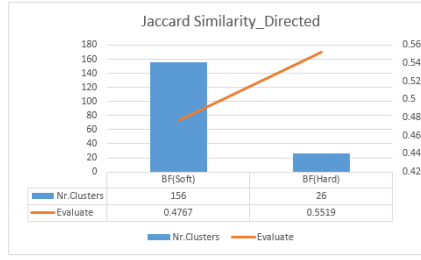
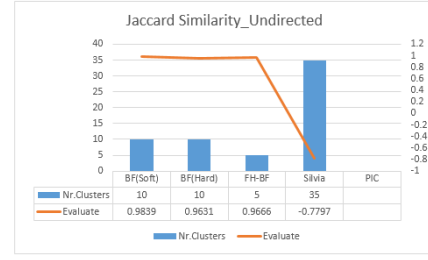


Figure 5.3.: Mountainbike - Motor bout, Rodroque similarity

5. Evaluation

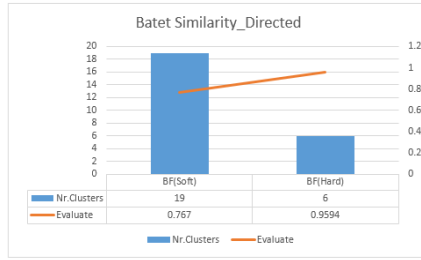


(a) Jaccard, directed diagram

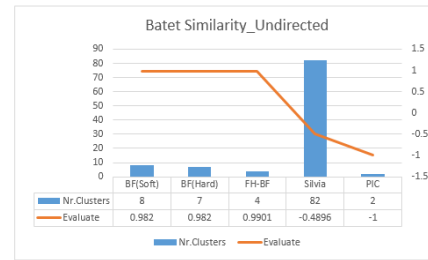


(b) Jaccard, undirected diagram

Figure 5.4.: Mountainbike - Motor bout, Jaccard similarity

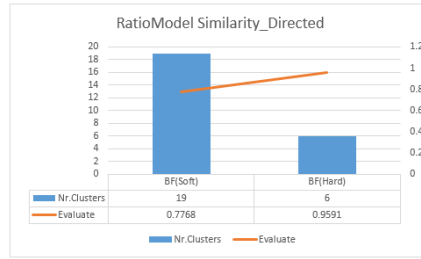


(a) Batet, directed diagram

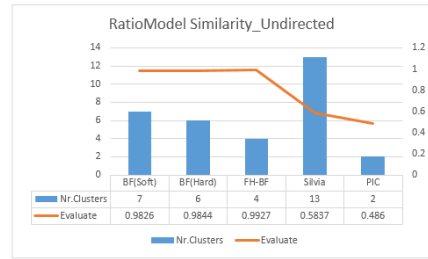


(b) Batet, undirected diagram

Figure 5.5.: "Daughter - in - law - Son - in - law", Batet similarity

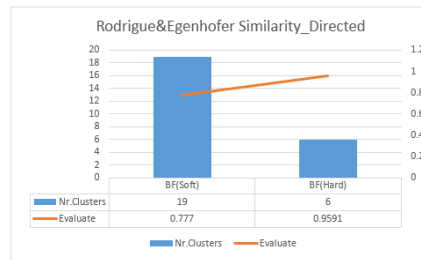


(a) Ratio, directed diagram

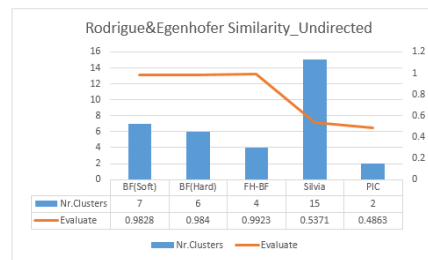


(b) Ratio, undirected diagram

Figure 5.6.: "Daughter - in - law - Son - in - law", Ratio similarity



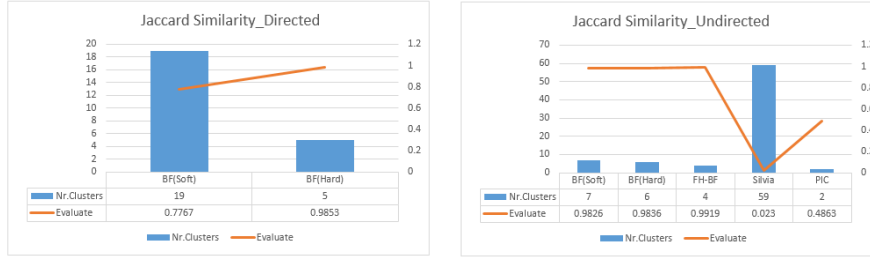
(a) Rodrigue, directed diagram



(b) Rodrigue, undirected diagram

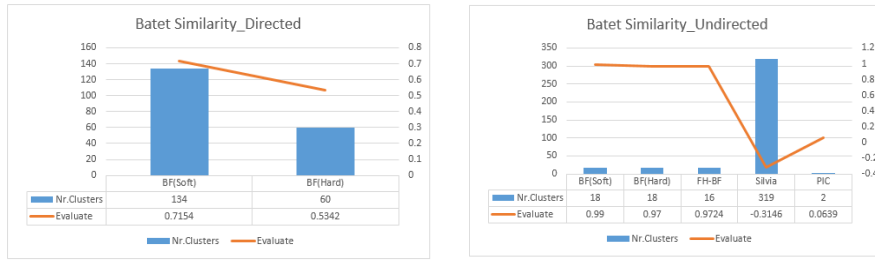
Figure 5.7.: "Daughter - in - law - Son - in - law", Rodrigue similarity

5.3. Comparing Clustering Algorithms by Performing on Special DBpedia Datasets



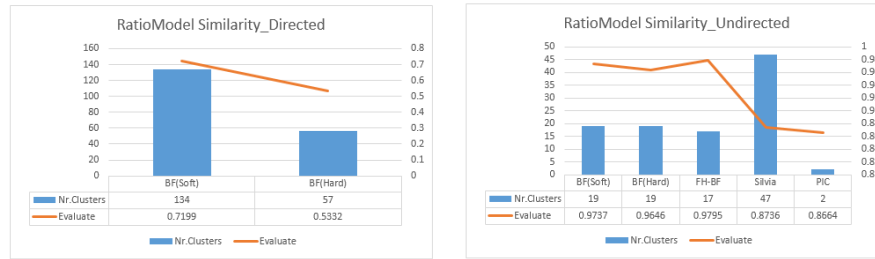
(a) Jaccard, directed diagram (b) Jaccard, undirected diagram

Figure 5.8.: "Daughter - in - law - Son - in - law", Jaccard similarity



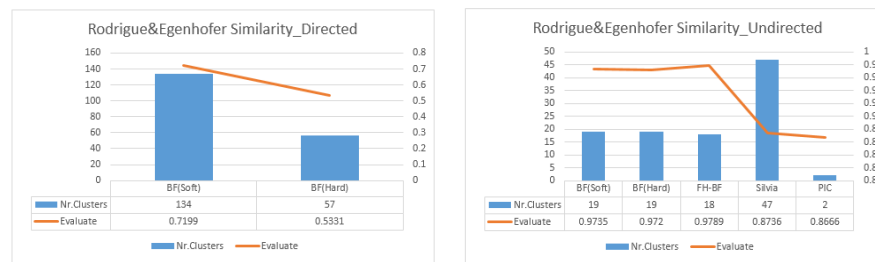
(a) Batet, directed diagram (b) Batet, undirected diagram

Figure 5.9.: "Taxi-Taxi Driver", Batet similarity



(a) Ratio, directed diagram (b) Ratio, undirected diagram

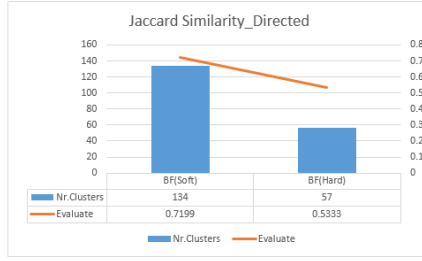
Figure 5.10.: "Taxi-Taxi Driver", Ratio similarity



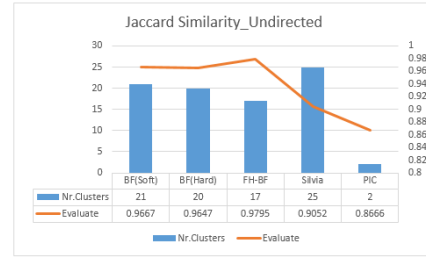
(a) Rodrigue, directed diagram (b) Rodrigue, undirected diagram

Figure 5.11.: "Taxi-Taxi Driver", Rodrigue similarity

5. Evaluation

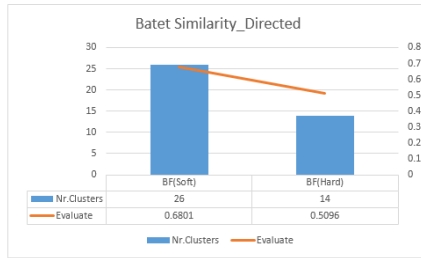


(a) Jaccard, directed diagram

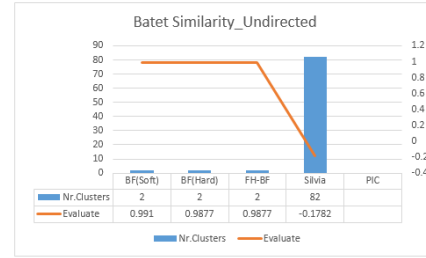


(b) Jaccard, undirected diagram

Figure 5.12.: "Taxi-Taxi Driver"- Jaccard Similarity

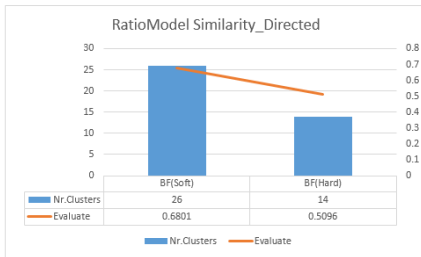


(a) Batet, directed diagram

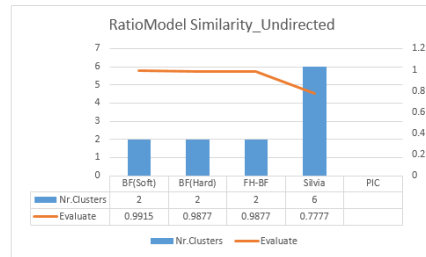


(b) Batet, undirected diagram

Figure 5.13.: "Toothbrush-Toothpaste", Batet similarity

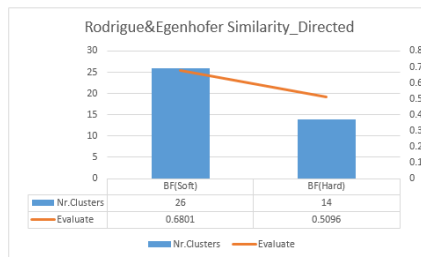


(a) Ratio, directed diagram

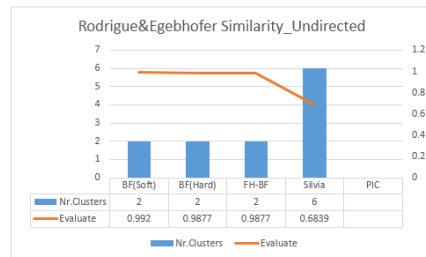


(b) Ratio, undirected diagram

Figure 5.14.: "Toothbrush-Toothpaste", Ratio similarity



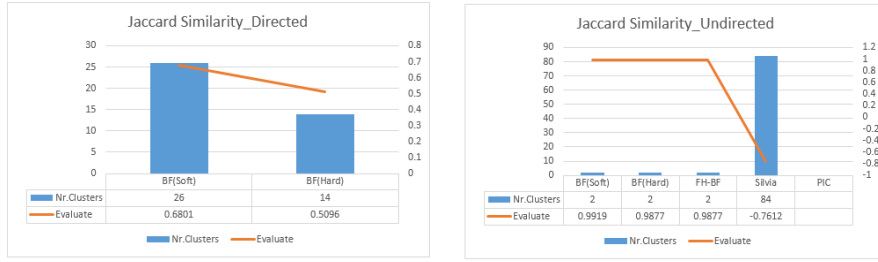
(a) Rodrigue, directed diagram



(b) Rodrigue, undirected diagram

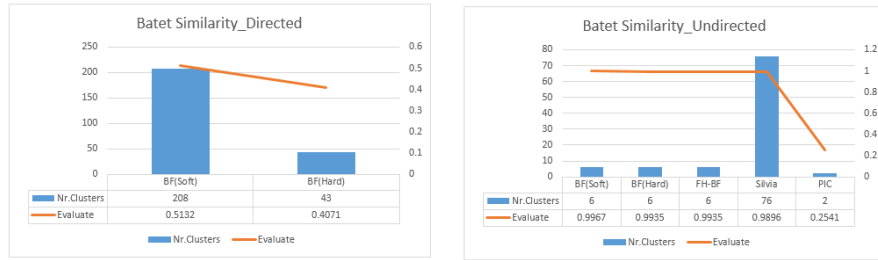
Figure 5.15.: "Toothbrush-Toothpaste", Rodrigue similarity

5.3. Comparing Clustering Algorithms by Performing on Special DBpedia Datasets



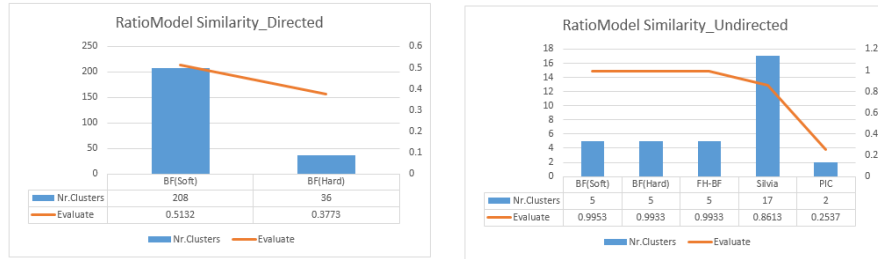
(a) Jaccard, directed diagram (b) Jaccard, undirected diagram

Figure 5.16.: "Toothbrush-Toothpaste", Jaccard similarity



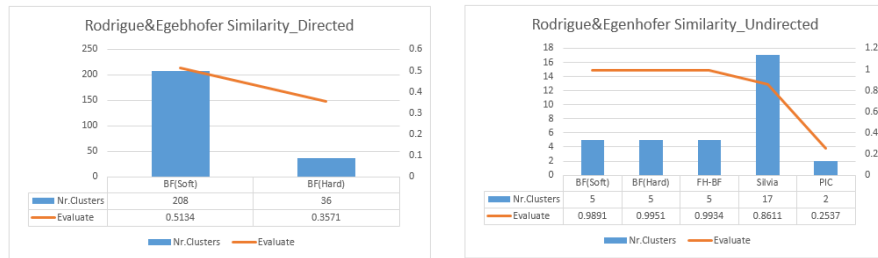
(a) Batet, directed diagram (b) Batet, undirected diagram

Figure 5.17.: "HairStylist-TaxiDriver", Batet similarity



(a) Ratio, directed diagram (b) Ratio, undirected diagram

Figure 5.18.: "HairStylist-TaxiDriver", Ratio similarity



(a) Rodrigue, directed diagram (b) Rodrigue, undirected diagram

Figure 5.19.: "HairStylist-TaxiDriver", Rodrigue similarity

5. Evaluation

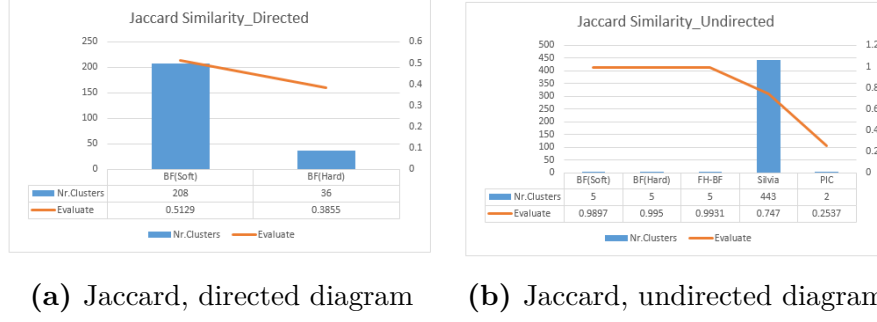


Figure 5.20.: "HairStylist-TaxiDriver", Jaccard similarity

5.4. Evaluation Results for Similarity functions

In this section, we present our results on small DBpedia dataset. The description for this dataset is, name: DBpedia-2016-04, number of triples: 30,793 and size: 4,2 MB. The execution time which is one of the crucial parameters to monitor the performance and evaluation, and to monitor the quality and accuracy between different similarity functions, are compared. The clustering algorithm that is selected for this purpose is PowerIteration. By Figure 5.21, Jaccard similarity makes

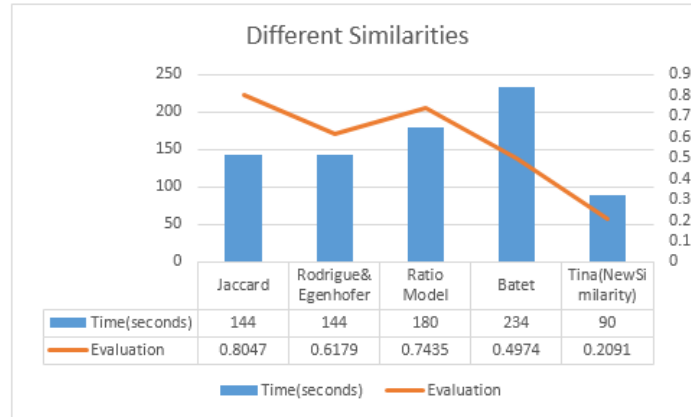


Figure 5.21.: Comparing some similarity measures

the best clusters among the other similarities. On the other hand, New Pseudo Similarity(Tina) given by (4.6) is the fastest similarity with the worst quality. But it works well for large datasets, in particular, for highly overlapping communities can exhibit more external than internal connections.

Implementing Jaccard, Rodrigue-Egenhofer, Ratio Model and Batet similarities are failed on the large datasets. Among of the candidate similarity measures, only New Pseudo Similarity(Tina) is scalable.

5.5. Evaluation Results for Illustrating Scalability

In the following diagram, we present our results on some datasets with different sizes (between 2 GB to 50GB) extracted from DBpedia and BSBM to explore the scalability of PowerIteration clustering (PIC) with (Tina)New Similarity function (4.6) and Silhouette evaluation code.

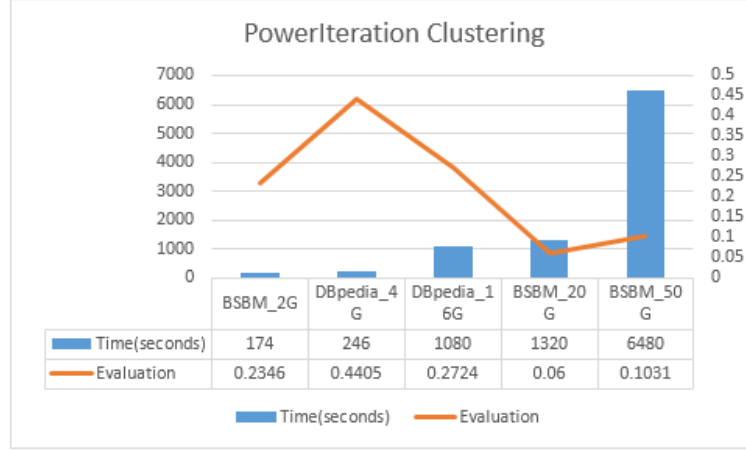


Figure 5.22.: Evaluation algorithm for PowerIteration Clustering

We observe that by increasing the size of dataset, the execution time in PowerIteration clustering with New Pseudo Similarity(Tina) and Silhouette evaluation is also increased. But the execution time according to the size of dataset is small. It illustrates how PowerIteration clustering and Silhouette evaluation are scalable algorithms. Notice that the execution time on 23 GB and 53 GB BSBM datasets for PIC and Silhouette algorithm is 20 minutes and less than 2 hours, respectively.

6. Conclusions and Future Work

During this thesis, we investigated some various research axes in the field of effectively and efficiently distributing data structures. This research project was an attempt to create a framework (Distributed RDF Clustering Framework) adopted for implementing and analysing some different clustering algorithms required for processing big RDF datasets.

In our thesis, we implemented the candidate clustering algorithms (PowerIteration, BorderFlow and Link-based) with five different similarities and used Silhouette evaluation algorithm to evaluate their efficiency. However, the scalability of the candidate clustering algorithms is a fundamental task. We successfully clustered the RDF data and tested the scalability of our algorithms by running on big datasets. We observed that PowerIteration clustering with Tina(pseudo Similarity) performs well on large RDF datasets as well as Silhouette evaluation algorithm. It will be exciting to see to optimise the other algorithms and make them scalable. Hence, we provided a primary plan of a new clustering algorithm based on Link-based(Silvia). Also we defined Tina(pseudo Similarity) which is a scalable pseudo similarity function to perform the clustering algorithms on large datasets. In addition, some datasets were prepared from DBpedia to test the quality of the other candidate clustering algorithms. Some conclusions were extracted here that belong to performance, quality, and accuracy of the clustering algorithms.

Actually, we notably:

1. Clustered the data in a scalable manner in the RDF data.
2. Performed PowerIteration (PIC) with Tina(New Pseudo Similarity) & Silhouette Evaluation on large RDF datasets.
3. Observed that New Pseudo Similarity is faster than other similarities.
4. Designed FH-BF clustering and observed that it is faster than BorderFlow(Hard & Soft) but with the almost same quality.
5. Implemented four clustering algorithms (PIC, BorderFlow, FH-BF and Link-based) with some distinct semantic similarity measures in RDF data.

6. Conclusions and Future Work

Our study, raises a number of opportunities for future work. Studying the scalability of the efficient clustering algorithms would constitute an interesting step for providing a new dimension in the preparation a distributed framework. This experimental study can also be improved by designing new algorithms in a scalable manner. Our thesis work can also be extended by finding the other similarities with better quality. Moreover, we will try to find more efficient clustering algorithms for big datasets. The following list presents some perspectives for further developments:

1. Designing and implementing a new clustering algorithm based on Link-based(Silvia) clustering approach.
2. Finding other scalable similarities.
3. Finding other efficient and parallelizable clustering algorithms.

Our thesis work will be an addition for Semantic Analytics Stack(SANSA)¹, a distributed computing frameworks with the semantic technology stack. A layer named as "Clustering" in SANSA Stack will help to cluster the RDF data.

¹<https://github.com/SANSA-Stack/SANSA-ML/tree/develop/sansa-ml-spark/src/main/scala/net/sansa-stack/ml/spark/clustering>

A. Appendix

We have created five special datasets by using DBpedia. The datasets we have used here are Taxi-Taxidriver with 661 triples, Toothpaste-Toothbrush with 104 triples, Hairstylist-Taxidriver with 516 triples, Daughter-in-law-Son-in-law with 203 triples and Mountainbike-Motorboat with 379 triples. Each of our dataset is composed of two datasets (e.g. Taxi-Taxidriver dataset includes two Taxi and Taxidriver datasets). You can see the query which we have used for creating the Taxi dataset in the following.

```
SELECT*WHERE{
{
select distinct* where {?s ?r ?o.
?s rdfs:label "Taxi"@en.
FILTER(!isLiteral(?o) && (?r !=
<http://www.w3.org/1999/02/22-rdf-syntax-ns # type>))
}
}
UNION{
select distinct*where{?s ?r ?o.
?o rdfs:label "Taxi"@en.
FILTER(!isLiteral(?s) && (?r !=
<http://www.w3.org/1999/02/22-rdf-syntax-ns # type>))
}
}
UNION{
select?q ?p ?o where{
?s rdfs:label "Taxi"@en.
?s ?r ?o.
?q ?p ?o.
FILTER(!isLiteral(?o) && (?p !=
<http://www.w3.org/1999/02/22-rdf-syntax-ns # type>))
}
}
UNION{
select ?q ?p ?o where{
```

A. Appendix

```
?s rdfs:label "Taxi"@en.  
?s ?r ?q.  
?q ?p ?o.  
FILTER(!isLiteral(?o) && (?p !=  
<http://www.w3.org/1999/02/22-rdf-syntax-ns # type>))  
}  
}  
UNION{  
  select ?s ?p ?q where{  
    ?o rdfs:label "Taxi"@en.  
    ?q ?r ?o.  
    ?s ?p ?q.  
    FILTER(!isLiteral(?q) && (?p !=  
    <http://www.w3.org/1999/02/22-rdf-syntax-ns # type>))  
  }  
}  
UNION{  
  select ?s ?p ?q where{  
    ?o rdfs:label "Taxi"@en.  
    ?s ?r ?o.  
    ?s ?p ?q.  
    FILTER(!isLiteral(?q) && (?p !=  
    <http://www.w3.org/1999/02/22-rdf-syntax-ns # type>))  
  }  
}
```

Bibliography

- [1] C. C. Aggarwal, C. K. Reddy. Data Clustering: Algorithms and Applications. Chapman and Hall/CRC, 1st edn., 2013.
- [2] Y. Y. Ahn, J. P. Bagrow, S. Lehmann. Link communities reveal multiscale complexity in networks. *Nature* 466(7307): 761–764, 2010.
- [3] A. Alzogbi, G. Lausen. Similar structures inside rdf-graphs, LDOW, 2013.
- [4] J. S. Andersen and O. Zukunft. Evaluating the Scaling of Graph-Algorithms for Big Data Using GraphX. In *International Conference on Open and Big Data (OBD)*, pages 1–8. IEEE, 2016.
- [5] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, and Z. Ives. DBpedia: A nucleus for a web of open data. In *Proc. of International Semantic Web Conference*, Lecture Notes in Computer Science book series, volume 4825, pages 722-735, 2007.
- [6] J. Bertolucci. Hadoop: From Experiment To Leading Big Data Platform, *Information Week*, 2013. Retrieved on 14 November 2013.
- [7] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The story so far. *Int. Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.
- [8] M. A. Bornea, J. Dolby, A. Kementsietsidis, K. Srinivas, P. Dantressangle, O. Udrea, and B. Bhattacharjee. Building an efficient RDF store over a relational database. In *Proc. of 2013 SIGMOD Conference*, pages 121-132, 2013.
- [9] L. Bühmann, J. Lehmann, and P. Westphal. DL-Learner—A framework for inductive learning on the Semantic Web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 39:15–24, 2016.
- [10] C. Carpineto, S. Osinski, G. Romano, D. Weiss. A survey of web clustering engines. *ACM Computing Surveys* 41(3), July 2009.
- [11] W. W. Cohen. TensorLog: A differentiable deductive database. arXiv preprint arXiv:1605.06523, 2016.

- [12] S. Decker, P. Mitra, and S. Melnik. Framework for the semantic web: an RDF tutorial. *IEEE Internet Computing*, 4(6):68-73, 2000.
- [13] N. Dedic, C. Stanier. Towards Differentiating Business Intelligence, Big Data, Data Analytics and Knowledge Discovery. 285. Berlin ; Heidelberg: Springer International Publishing, 2017.
- [14] A. Delteil, C. Faron-Zucker, R. Dieng. Learning ontologies from rdf annotations. *In Proceedings of the Second Workshop on Ontology Learning OL'2001*, Seattle, USA, 2001.
- [15] H. Derrick. 4 reasons why Spark could jolt Hadoop into hyperdrive. *Gigaom*, 2014.
- [16] R. Dieng, S. Hug. Comparison of personal ontologies represented through conceptual graphs. *In Proceedings of ECAI 1998*. pages 341-345, 1998.
- [17] D. Doan. Re: cassandra + spark / pyspark. *Cassandra User (Mailing list)*. Retrieved 2014-11-21.
- [18] V. Estivill-Castro. Why so many clustering algorithms - A Position Paper. *ACM SIGKDD Explorations Newsletter*. 4 (1): 65-75, 2002.
- [19] T. Evans, R. Lambiotte. Line graphs, link partitions, and overlapping communities. *Physical Review E*. 80(1), 016105, 2009.
- [20] S. Everts."Information Overload. *Distillations*. 2 (2): 26-33, 2016. Retrieved 17 February 2017.
- [21] N. Fanizzi, C. d'Amato. A hierarchical clustering method for semantic knowledge bases. *In Apolloni, B., Howlett, R.J., Jain, L. (eds.) KES 2007*, Part III. LNCS (LNAI), vol. 4694, pp. 653-660. Springer, Heidelberg, 2007.
- [22] L. Feigenbaum. The Semantic Web in Action. Scientific American, 2007. Retrieved February 24, 2010.
- [23] S. Fortunato. Community detection in graphs. *Physics Reports* 486(3-5): 75-174, 2010.
- [24] L. Galarraga, C. Teflioudi, K. Hose, and F. M. Suchanek. Fast Rule Mining in Ontological Knowledge Bases with AMIE⁺. *Very Large Databases Journal*, 24:707-730, 2015.

- [25] S. Giannini. RDF Data Clustering. *In proceeding International Conference on Business Information Systems*, Part of the Lecture Notes in Business Information Processing book series (LNBIP, volume 160), pp. 220-231, 2013.
- [26] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M.J. Franklin, I. Stoica. GraphX: Graph Processing in a Distributed Dataflow Framework, OSDI 2014.
- [27] A. Gottlieb, G. Y. Stein, E. Ruppín, R. Sharan. PREDICT: a method for inferring novel drug indications with application to personalized medicine. *Mol. Syst. Biol.*, 7:496, 2011.
- [28] G. A. Grimnes, P. Edwards, A. Preece. Learning Meta-Descriptions of the FOAF Network. *In Proceedings of ISWC04*, Hiroshima, Japan, Springer Verlag, 152-165, 2004.
- [29] S. Harispe, D. Sánchez, S. Ranwez, S. Janaqi, J. Montmain. A framework for unifying ontology-based semantic similarity measures: A study in the biomedical domain. *Journal of Biomedical Informatics*, 48:38-53, 2014.
- [30] T. Heath, C. Bizer. Linked Data: Evolving the Web into a Global Data Space. Synthesis Lectures on the Semantic Web, Morgan and Claypool Publishers, 2011.
- [31] A. Hliaoutakis, G. Varelas, E. Voutsakis, E. G. M. Petrakis, E. Milios. Information retrieval by semantic similarity. *Int. J. Semant. Web Inf. Syst.*, 2:55–73, 2006.
- [32] J. Huang, D. J. Abadi, and K. Ren. Scalable SPARQL querying of large RDF graphs. *Proc. of VLDB Endow.*, 4(11):1123-1134, 2011.
- [33] J. Jiang, D. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. *Int Conf Res Comput Linguist (ROCLING X)*, pp. 19-33, 1997.
- [34] J. H. Lee, M. H. Kim, Y. J. Lee. Information retrieval based on conceptual distance in is-a hierarchies. *J. Doc.*, 49:188–207, 1993.
- [35] J. Lehmann, G. Sejdiu, L. Bühmann, P. Westphal, C. Stadler, I. Ermilov, S. Bin, N. Chakraborty, M. Saleem, A. C. Ngonga Ngomo, H. Jabeen. Distributed Semantic Analytics Using the SANSA Stack. *In Proceeding International Semantic Web Conference (ISWC)*, pp. 147-155, 2017
- [36] D. Lin. An information-theoretic definition of similarity. *In 15th Int Conf Mach Learn*, Madison, WI. pp. 296-304, 1998.

- [37] F. Lin and W. Cohen, Power Iteration Clustering, *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel, 2010.
- [38] A. Maedche, V. Zacharias. Clustering ontology-based metadata in the semantic web. *In Proceedings of PKDD 2002*. pp. 348-360, 2002.
- [39] G. K. Mazandu, J.N. Mulder. IT-GOM: an integrative tool for IC-based GO semantic similarity measures. *BMC Bioinformatics*, 14:284, 2013.
- [40] M. Montes-y-Gómez, A. Gelbukh, A. Lóopez-Lóopez. Comparison of Conceptual Graphs. In: *Lecture Notes in Artificial Intelligence*. Volume 1793. Springer Verlag, pp. 548-556, 2000.
- [41] M. Morsey, J. Lehmann, S. Auer, A.-C. Ngonga Ngomo. DBpedia SPARQL benchmark- performance assessment with real queries on real data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) *ISWC 2011, Part I. LNCS*, vol. 7031, pp. 454–469. Springer, Heidelberg, 2011.
- [42] T. Neumann and G. Weikum. The RDF-3X engine for scalable management of RDF data. *The VLDB Journal*, 19(1):91-113, 2010.
- [43] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.
- [44] A.-C. Ngonga Ngomo, F. Schumacher. BorderFlow: A local graph clustering algorithm for natural language processing. In Gelbukh, A. (ed.) *CICLing 2009*. LNCS, vol. 5449, pp. 547-558. Springer, Heidelberg, 2009.
- [45] A. -C. Ngonga Ngomo. Parameter-free clustering of protein-protein interaction graphs. *In: Proceedings of Symposium on Machine Learning in Systems Biology 2010*, 2010.
- [46] S. Osinski, J. Stefanowski, D. Weiss. Lingo: search results clustering algorithm based on singular value decomposition. *In Proceedings of the International Conference on Intelligent Information Systems (IIPWM 2004)*, Zakopane, Poland, pp. 359–368, 2004.
- [47] S. Osinski, D. Weiss. Carrot: design of a flexible and efficient web information retrieval framework. In: Szczepaniak, P.S., Kacprzyk, J., Niewiadomski, A. (eds.) *AWIC 2005*. LNCS (LNAI), vol. 3528, pp. 439–444. Springer, Heidelberg, 2005.

- [48] C. Pesquita, D. Faria, A. O. Falcão, P. Lord, F. M. Couto. Semantic similarity in biomedical ontologies. *PLoS Comput. Biol.*, 5:12, 2009.
- [49] G. Pirró, J. Euzenat. A feature and information theoretic framework for semantic similarity and relatedness. In: *Proc 9th Int Semant Web Conf ISWC 2010*, Springer, pp. 615–30, 2010.
- [50] M. J. Rattigan, M. Maier, D. Jensen. Graph clustering with network structure indices. In *Proceedings of the 24th International Conference on Machine Learning*, pp. 783–790, ACM 2007.
- [51] P. Resnik. Using Information content to evaluate semantic similarity in a taxonomy. In *Proceeding 14th Int Jt Conf Artif Intell IJCAI*, pp. 448–53, 1995.
- [52] A. Rodríguez, M. J. Egenhofer. Determining semantic similarity among entity classes from different ontologies. *IEEE Trans. Knowl. Data Eng.* 15:442–56, 2003.
- [53] P. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20(1):53-65, 1987.
- [54] V. Salin, M. Slastihina, I. Ermilov, R. Speck, S. Auer, and S. Papshev, Semantic Clustering of Website Based on Its Hypertext Structure. *KESW 2015*, CCIS 518, pp. 182–194, 2015.
- [55] D. Sánchez, M. Batet, D. Isern, A. Valls. Ontology-based semantic similarity: a new feature-based approach. *Expert Syst. Appl.* 39:7718–28, 2012.
- [56] V. Satuluri, S. Parthasarathy, Y. Ruan. Local graph sparsification for scalable clustering. In *SIGMOD 2011*, pp. 721–732, 2011.
- [57] G. Scanniello, A. Marcus. Clustering support for static concept location in source code. In: *ICPC*, pp. 1–10, 2011.
- [58] S. Schaeffer. Graph clustering. *Computer Science Review* 1(1), 27–64, 2007.
- [59] A. Schatzle, M. Przyjaciół-Zablocki, S. Skilevic, G. Lausen, S2RDF: RDF Querying with SPARQL on Spark, *VLDB Endowment* 2150-8097/16/06, 2016.

- [60] A. Schatzle, M. Przyjaciół-Zablocki, T. Berberich, and G. Lausen, S2X: Graph-Parallel Querying of RDF with GraphX, *Conference: VLDB Workshop on Big Graphs Online Querying VLDB Workshop on Data Management and Analytics for Medicine and Healthcare*, 2016.
- [61] N. Schlitter, T. Falkowski, J. Lassig. Dengraph-ho: Density-based hierarchical community detection for explorative visual network analysis. In: *Springer (ed.) Proceedings of the 31st SGAI International Conference on Artificial Intelligence*, 2011.
- [62] S. Schmidt, Data is exploding: the 3 V's of big data. *Business Computing World*, 2012.
- [63] N. Seco, T. Veale, J. Hayes. An intrinsic information content metric for semantic similarity in WordNet. In: *16th Eur Conf Artif Intell*. IOS Press, pp. 1–5, 2004.
- [64] C. Snijders, U. Matzat, U.-D. Reips. Big Data: Big gaps of knowledge in the field of Internet. *International Journal of Internet Science*. 7: 1–5, 2012.
- [65] M-F. Sy, S. Ranwez, J. Montmain, A. Regnault, M. Crampes, V. Ranwez. User centered and ontology based information retrieval system for life sciences. *BMC Bioinformatics*, 13(Suppl 1):S4, 2012.
- [66] A. Tversky. Features of similarity. *Psychol Rev*, 84:327-52, 1977.
- [67] W. Y. Wang, K. Mazaitis, and W. W. Cohen. Structure learning via parameter learning. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pp. 1199–1208. ACM, 2014.
- [68] J. Webster. MapReduce: Simplified Data Processing on Large Clusters, *Search Storage*, 2004. Retrieved on 25 March 2013.
- [69] P. Yuan, P. Liu, B. Wu, H. Jin, W. Zhang, and L. Liu. TripleBit: A fast and compact system for large scale RDF data. *Proc. VLDB Endow.*, 6(7):517-528, 2013.
- [70] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, I. Stoica, Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing, *USENIX Symp. Networked Systems Design and Implementation*, 2011.

- [71] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica. Spark: Cluster Computing with Working Sets (PDF). *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2010.
- [72] O. Zamir, O. Etzioni. Web document clustering: a feasibility demonstration. In *SIGIR 1998: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Melbourne, Australia, pp. 46–54, 1998.
- [73] K. Zeng, J. Yang, H. Wang, B. Shao, and Z. Wang. A distributed graph engine for Web Scale RDF data. *Proc. VLDB Endow.*, 6(4):265-276, Feb. 2013.
- [74] X. Zhichao, C. Wei, G. Lei and T. Wang, Spark RDF: In-Memory Distributed RDF Management Framework for Large-Scale Social Data. *International Conference on Web-Age Information Management (WAIM)*, pp 337-349, 2015.