Querying semantic web/linked data graphs using summarization

Mussab Zneika

Querying Semantic Web/Linked Data Graphs Using Summarization

Mussab Zneika

ETIS Lab, UMR 8051
University of Paris Seine, University of Cergy Pontoise,
ENSEA,CNRS

A thesis submitted for the degree of

Doctor of Philosophy in Computer Science

20/09/2019

COMPOSITION OF THE JURY

Bernd Amann	Professeur, LIP6, Université Pierre et Marie Curie	Président du jury
Daniela Grigori	Professeur, LAMSADE, Université Paris-Dauphine	Rapporteur
Fatiha Sais	MCF, HDR, LRI, Université Paris Sud	Rapporteur
Vassilis Christophides	Professeur, University of Crete	Examinateur
Claudio Lucchese	Associate professor, Ca' Foscari University of Venice	Examinateur
Dimitris Kotzinos	Professeur, ETIS, Université de Cergy-Pontoise	Directeur de these
Dan Vodislav	Professeur, ETIS, Université de Cergy-Pontoise	Co-Directeur de these

I dedicate this thesis to my life-coaches; mom and dad. I could not have asked for better parents or role-models. I also dedicate this work to my beloved wife Waad and my precious daughter Naya, who is the hope and the light through which I see the beauty of life.

Acknowledgements

Foremost, I would like to express my gratitude to my PhD advisor Dimitris Kotzinos, for his enormous support of my research, immense knowledge, his confidence, and for his professional instruction which has been a key for the advancement of this thesis. I would like also to thank Dan Vodislav, the Co-advisor of the PhD. For his gentle advices, patience, motivation, availability. The guidance of my advisors helped me in all the time of research, writing of this thesis, and becoming an autonomous researcher, besides their generosity and help in personal issues.

Besides my advisors, my sincere thanks also go to the rest of my thesis committee: Mrs. Daniela Grigori and Mrs. Fatiha Sais for accepting to read and review my thesis even though it coincided with the time of their holidays. Many thanks also to Mr. Bernd Amann, Mr. Vassilis Christophides, and Mr. Claudio Lucchese for accepting to be part of the committee.

My sincere thanks go also to my research team MIDI, my colleagues in the ETIS lab, the atmosphere has always helped me to keep up the motivation for this work. Without their precious support, it would not be possible to conduct this research. I am also grateful to my friends who surrounded me with a warm atmosphere, and kind company.

Last but not the least, I would like to thank my family. To my passed father, and to my mother to whom I am grateful for raising me up in the way they did. They are the first reason to reach this moment. I thank them for their strengthen support all along my education way. I am indebted to them.

Of course a warm thanks go to my wife Waad who has been constant support and encouragement, thanks for taking care of the things when i could not. I thank her for her tolerance, and intelligence in keeping our house in order. A very gentle thanks go to my lovely daughter Naya, the smile of my life, who gives me a huge energy. Thanks for her because she made me laugh in the most difficult times.

Résumé

La quantité de données RDF disponibles augmente rapidement à la fois en taille et en complexité, les Bases de Connaissances (Knowledge Bases – KBs) contenant des millions, voire des milliards de triplets étant aujourd'hui courantes. Plus de 1000 sources de données sont publiées au sein du nuage de Données Ouvertes et Liées (Linked Open Data – LOD), qui contient plus de 62 milliards de triplets, formant des graphes de données RDF complexes et de grande taille. L'explosion de la taille, de la complexité et du nombre de KBs et l'émergence des sources LOD ont rendu difficile l'interrogation, l'exploration, la visualisation et la compréhension des données de ces KBs, à la fois pour les utilisateurs humains et pour les programmes. Pour traiter ce problème, nous proposons une méthode pour résumer de grandes KBs RDF, basée sur la représentation du graphe RDF en utilisant les (meilleurs) top-k motifs approximatifs de graphe RDF. La méthode, appelée SemSum+, extrait l'information utile des KBs RDF et produit une description d'ensemble succincte de ces KBs. Elle extrait un type de schéma RDF ayant divers avantages par rapport aux schémas RDF classiques, qui peuvent être respectés seulement partiellement par les données de la KB. A chaque motif approximatif extrait est associé le nombre d'instances qu'il représente ; ainsi, lors de l'interrogation du graphe RDF résumé, on peut facilement déterminer si l'information nécessaire est présente et en quantité significative pour être incluse dans le résultat d'une requête fédérée. Notre méthode ne demande pas le schéma initial de la KB et marche aussi bien sans information de schéma du tout, ce qui correspond aux KBs modernes, construites soit ad-hoc, soit par fusion de fragments en provenance d'autres KBs. Elle fonctionne aussi bien sur des graphes RDF homogènes (ayant la même structure) ou hétérogènes (ayant des structures différentes, pouvant être le résultat de données décrites par des schémas/ontologies différentes). A cause de la taille et de la complexité des graphes RDF, les méthodes qui calculent le résumé en chargeant tout le graphe en mémoire ne passent

pas à l'échelle. Pour éviter ce problème, nous proposons une approche générale parallèle, utilisable par n'importe quel algorithme approximatif de fouille de motifs. Elle nous permet de disposer d'une version parallèle de notre méthode, qui passe à l'échelle et permet de calculer le résumé de n'importe quel graphe RDF, quelle que soit sa taille. Ce travail nous a conduit à la problématique de mesure de la qualité des résumés produits. Comme il existe dans la littérature divers algorithmes pour résumer des graphes RDF, il est nécessaire de comprendre lequel est plus approprié pour une tâche spécifique ou pour une KB RDF spécifique. Il n'existe pas dans la littérature de critères d'évaluation établis ou des évaluations empiriques extensives, il est donc nécessaire de disposer d'une méthode pour comparer et évaluer la qualité des résumés produits. Dans cette thèse, nous définissons une approche complète d'évaluation de la qualité des résumés de graphes RDF, pour répondre à ce manque dans l'état de l'art. Cette approche permet une compréhension plus profonde et plus complète de la qualité des différents résumés et facilite leur comparaison. Elle est indépendante de la façon dont l'algorithme produisant le résumé RDF fonctionne et ne fait pas de suppositions concernant le type ou la structure des entrées ou des résultats. Nous proposons un ensemble de métriques qui aident à comprendre non seulement si le résumé est valide, mais aussi comment il se compare à d'autre résumés par rapport aux caractéristiques de qualité spécifiées. Notre approche est capable (ce qui a été validé expérimentalement) de mettre en évidence des différences très fines entre résumés et de produire des métriques capables de mesurer cette différence. Elle a été utilisée pour produire une évaluation expérimentale approfondie et comparative de notre méthode.

Abstract

The amount of RDF data available increases fast both in size and complexity, making available RDF Knowledge Bases (KBs) with millions or even billions of triples something usual, e.g. more than 1000 datasets are now published as part of the Linked Open Data (LOD) cloud, which contains more than 62 billion RDF triples, forming big and complex RDF data graphs. This explosion of size, complexity and number of available RDF Knowledge Bases (KBs) and the emergence of Linked Datasets made querying, exploring, visualizing, and understanding the data in these KBs difficult both from a human (when trying to visualize) and a machine (when trying to query or compute) perspective. To tackle this problem, we propose a method of summarizing a large RDF KBs based on representing the RDF graph using the (best) top-k approximate RDF graph patterns. The method is named SemSum+ and extracts the meaningful/descriptive information from RDF Knowledge Bases and produces a succinct overview of these RDF KBs. It extracts from the RDF graph, an RDF schema that describes the actual contents of the KB, something that has various advantages even compared to an existing schema, which might be partially used by the data in the KB. While computing the approximate RDF graph patterns, we also add information on the number of instances each of the patterns represents. So, when we query the RDF summary graph, we can easily identify whether the necessary information is present and if it is present in significant numbers whether to be included in a federated query result. The method we propose does not require the presence of the initial schema of the KB and works equally well when there is no schema information at all (something realistic with modern KBs that are constructed either ad-hoc or by merging fragments of other existing KBs). Additionally, the proposed method works equally well with homogeneous (having the same structure) and heterogeneous (having different structure, possibly the result of data described under different schemas/ontologies) RDF graphs. Given that RDF graphs can

be large and complex, methods that need to compute the summary by fitting the whole graph in the memory of a (however large) machine will not scale. In order to overcome this problem, we proposed, as part of this thesis, a parallel framework that allows us to have a scalable parallel version of our proposed method. This will allow us to compute the summaries of any RDF graph regardless of size. Actually, we generalized this framework so as to be usable by any approximate pattern mining algorithm that needs parallelization. But working on this problem, introduced us to the issue of measuring the quality of the produced summaries. Given that in the literature exist various algorithms that can be used to summarize RDF graphs, we need to understand which one is better suited for a specific task or a specific RDF KB. In the literature, there is a lack of widely accepted evaluation criteria or an extensive empirical evaluation. This leads to the necessity of a method to compare and evaluate the quality of the produced summaries. So, in this thesis, we provide a comprehensive Quality Framework for RDF Graph Summarization to cover the gap that exists in the literature. This framework allows a better, deeper and more complete understanding of the quality of the different summaries and facilitates their comparison. It is independent of the way RDF summarization algorithms work and makes no assumptions on the type or structure neither of the input nor of the final results. We provide a set of metrics that help us understand not only if this is a valid summary but also how a summary compares to another in terms of the specified quality characteristic(s). The framework has the ability, which was experimentally validated, to capture subtle differences among summaries and produce metrics that depict that and was used to provide an extensive experimental evaluation and comparison of our method.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Motivation

RDF has become one of the major standards in describing and publishing data, establishing what we call the Semantic Web. RDF represents data on the web in terms of triples of the form (s; p; o), explaining that the subject s has the property p, and the value of that property p is the object o. The RDF data triples are usually represented using labeled directed graphs called RDF graphs, in which subjects and objects are represented as labeled nodes and properties are represented as labeled directed edges. Publishing more and more data on the (Semantic) Web means that the amount of RDF data available increases fast both in size and complexity, making the appearance of RDF Knowledge Bases (KBs) with millions or even billions of triples something usual. Given that RDF is built on the promise of facilitating the linking together of relevant datasets or KBs and with the appearance of the Linked Open Data (LOD) cloud, we can now query KBs (both standalone or distributed) with millions or billions of triples altogether. Figure 1.1 shows the growth of the Linked Open Data (LOD) cloud in number of datasets. We can see that while in the beginning the LOD cloud consisted of only 12 dataset containing 1 billion triples, its current version has 1,205 RDF datasets containing more than 62 billion RDF triples and it forms big and complex RDF data graphs. The data in these KBs are not necessarily described by a known ontology(schema) and many times it is extremely time consuming to process all the interlinked KBs in order to acquire the necessary information. But even when the KB schema is known, we need actually to know which parts of the schema are used and how important the role of each part is, i.e. we need to start understanding the contents of the RDF KB.

This increased size and complexity of RDF KBs has a direct impact on various applications that we would like to perform with or against these RDF KBs. For

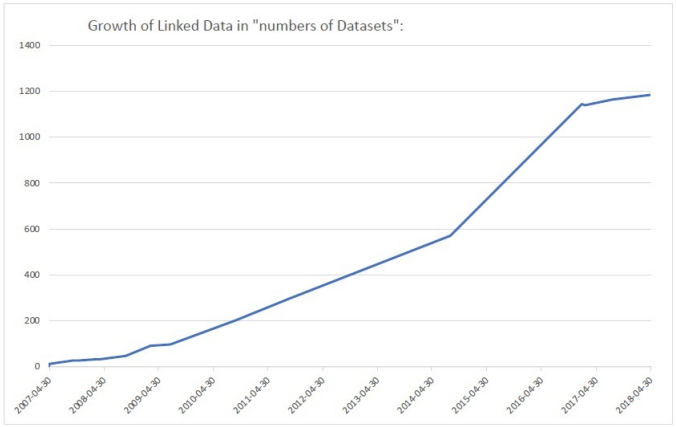


Figure 1.1 – The growth of the LOD cloud in number of datasets

example, the evaluation of SPARQL queries we express against these RDF KBs, the proper visualization of the contents of a large RDF KB, the data representation and description, etc. Especially on the LOD cloud, we observe that a query against a big, complex, interlinked and distributed RDF KB might retrieve no results at the end because either the association between the different RDF KBs is weak (is based only on a few associative links) or there is an association at the schema level that has never been instantiated at the actual data level. Moreover, a lot of these RDF KBs carry none at all or only partial schema information (mainly contain instances build and described separately or elsewhere).

One way to address the concerns described above is by trying to reduce the size of these KBs or at least represent them in a "reduced" way. Actually, data reduction is one of the prominent problems of the Big Data era. One of the proposed solutions in this direction is the creation of summaries of the RDF KBs, which in general will preserve the original inherent structure (classes and properties and their relationships) of the KB and carry some statistical and other cumulative information. In that respect, we could advise the user or the system/application to decide whether or not to post a query to the actual KB, since she knows whether information is present or not based on the summary. This would provide significant cost savings in processing time since we will substitute queries on complex RDF KBs with queries first on the summaries (on much simpler structures with no instances) and then with queries only towards the KBs that we know will produce some useful results. The same is true in the problems of graph visualization or graph indexing.

Independently of the application area, graph summarization techniques allow in

general the creation of a concise representation of the KB regardless of the existence or not of ontology (schema) information in the KB. Actually, the summary will represent the actual situation in the KB, namely should capture the *existing/used* classes and relationships by the instances and not what the schema proposes (and might have never been used). This should facilitate the query building for the end users with the additional benefit of exploring the contents of the KB based on the summary. And this should be even more useful in cases of linked datasets, when the actual information about various resources resides in another usually remote KB, which might contain additional but irrelevant information. And this holds regardless if we use heterogeneous or homogeneous, linked or not, standalone or distributed KBs. In all these cases we can use the RDF summary to concisely describe the data in the RDF KB and possibly add useful information for the RDF graph queries, like the distribution and the number of instances for each involved class or group of entities.

Additionally, given the size of some of the existing RDF KBs but more importantly considering the world of Linked Data, where KBs are linked together since elements of one are used in another, the computation of the RDF summary itself becomes a computational problem in terms of the scalability of the solutions. Most current solutions are memory based (as will be discussed in Chapter 3) and this limits their ability to scale. But this is more and more a computational bottleneck, since the size of the KBs keeps growing. In that respect, more scalable cloud based solutions need to be sought.

Moreover, given the wealth and diversity of applications associated with RDF graph summaries, one remains sometimes confused of which one to use and how to decide if the results are suitable in terms of quality but also according to its purpose. As discussed already, given the inherent complexity of the problem and the fact that these efforts try to preserve the semantics of the underlying KB, we need ways to compare and evaluate the quality of the solution. But identifying what is important to preserve in a summary and what is not, is a difficult and demanding task even for humans with a knowledge of a specific application domain. Nevertheless, generic solutions need to be sought in order to enable us to compare RDF graph summaries and help us decoding which algorithm(s) to use.

RDF graph summarization has many applications in areas of interest for data management in general, ranging from query answering/evaluation to graph visualization and from source selection to graph indexing. Work in this area has already provided partial solutions to some of these application areas but this is still a new area where more work is necessary. The explosion of the availability of Semantic Web

data and the effort to bring together and understand these datasets makes the ability to summarize them even more important. RDF graph summarization provides one of the steps towards better understanding our RDF datasets.

1.2 Problem Statement

In this thesis, we address the problem of creating RDF summaries of LOD/RDF graphs that is: given an input RDF graph, find the summary graph which reduces its size, while preserving the original inherent structure that exists in the Knowledge Base and correctly categorizing the instances included in it. This can be also seen as the problem of finding data representations, that will reduce the size while preserving the semantics of the dataset. But the problem becomes more complicated, since we would like to provide summaries of diverse types of RDF KBs (mainly in term of contents) and we want our summary to carry specific properties and characteristics that will make its use easy and will not require additional investment at the end. We summarize these requirements in the following:

- The summary should be an RDF graph itself, which allows us to use existing tools and methods to work with it(e.g. store it in RDF KBs, query it using SPARQL, etc.);
- Statistical information like the number of class and property instances per pattern should be included in our summary graph, which allows us e.g. to estimate a query's expected results' size towards the original graph or make visualization decisions;
- The summary should be much smaller than the original RDF graph and contain all the important concepts and their relationships based on statistical or other information. Customizing the notion of importance per application domain or use case or even user could be an interesting addition to the requirements fulfilled by any RDF summarization solution;
- Schema existence independence: the RDF summarization method should not require the existence of ontology level triples (this means that we should not require or assume any ontology information) and provide equally good results;
- Heterogeneity independence: RDF input graphs/KBs can be homogeneous or heterogeneous (both concerning the instance distribution and the existence of

different ontologies within the same KB); the RDF summarization method should summarize equally successfully both types of graphs;

- Scalability: the RDF summarization method should be (if possible infinitely) scalable, in order to be able to provide summaries on any size of RDF KB without being restricted by memory or computation requirements.

These requirements have a direct impact on how the algorithms built will work and what would be the expected acceptable results. They assure that a solution that fulfills those requirements will be able to work on any RDF KB, provide quality results that are directly exploitable using the existing infrastructure and allow us to better understand the underlying RDF KBs.

Finally, a last part of the problem we are interested in, deals with the assessment of the Quality of RDF summaries produced by the various algorithms. As already described, this is a hard problem because algorithms have been built with different application domain or user needs in mind and comparing them directly is difficult. Nevertheless, especially algorithms that follow some of the criteria described above, could be compared to one another in terms of how well they reflect the underlying RDF KB. This allows us, to understand strengths and weaknesses and choose the best algorithm for each problem. The problem we are dealing with is concerned with both comparing the quality of the results of the various summarization methods and also comparing our results to some ground truth summary (if and when it exists) to understand which algorithm comes closer and check and evaluate the details of the summarization. Needless to say that any solution to this problem should be independent of any algorithm and any ground truth summary. It should also be accommodating enough so as to be used to assess algorithms that will appear in the future.

1.3 Main Contributions

The first contribution of our work in this thesis is a novel solution into summarizing semantic LOD/RDF graphs [117, 118]. The generated summary graph is an RDF graph itself, so that we can process/work with the summaries and not the original graphs and exploit also the statistical information about the structure of the RDF input graph, which is included to our summary graph like the number of class and property instances per pattern. In summary, our solution is based on mining top-k

approximate graph patterns [66] and it offers the following features: (1) The summary is an RDF graph itself, which allows us to post simplified queries towards the summarizations using the same techniques (e.g. SPARQL), (2) statistical information (number of class and property instances per pattern) is included in our summary graph, which allows us to estimate a query's expected results' size, (3) the summary is much smaller than the original RDF graph, contains all the important concepts and their relationships based on the number of instances, (4) schema independence: it summarizes an RDF graph regardless of having or not schema and RDFS triples and (5) heterogeneity independence: it summarizes an RDF graph regardless if it is hetero- or homo-geneous. This contribution has been published in [117, 118]. This is mainly described in Chapter 4.

However, our first contribution was bound to datasets that fit in main memory, limiting the size of datasets for which our summaries could be generated. Thus, our second contribution is proposing a novel, parallel, scalable mechanism in order to generate summaries of billions of RDF triples. We have implemented our mechanism based on Hadoop / MapReduce Framework. This is described in Chapter 5.

But as we mentioned before several works are proposed in the literature for extracting summaries from RDF KBs. These proposed methods come from various scientific backgrounds ranging from generic graph summarization to explicit RDF graph summarization and from using rules to using the graph structure and thus and they produce different results while applied on the same KB. Additionally, there is no clear single definition of RDF summary, and not a single but many approaches to build RDF summaries. And given that RDF graph summarization plays a critical role in many different applications, including graph exploration and visualization, highlighting communities and query optimization, we need to have a way to compare the results that we get and decide which is the most suitable for our case.

One fundamental difficulty towards understanding the quality of the different generated summaries is a lack of widely accepted evaluation criteria or an extensive empirical evaluation. This leads to the necessity of a method to compare and evaluate the quality of the produced summaries. This method would allow a better understanding of the quality of the different summaries and facilitate their comparison and decide on their quality and best-fitness for specific tasks. So in this thesis, we provide a comprehensive Quality Framework for RDF Graph Summarization to cover the gap that exists in the literature. This framework allows a better, deeper and more complete understanding of the quality of the different summaries and facilitate their

comparison. This third contribution of this thesis is described in Chapter 6 and was published in [120].

Finally, we did an extensive experimental evaluation of the above contributions in order to validate their results and compare to the state-of-the-art.

The work in this thesis made contributions towards better understanding the contents of an RDF KB through summarization and by comparing the quality of the different summarization results. As we discuss in Chapter 8, more work is needed in this area in order to provide a complete, universal and dynamic solution.

1.4 Thesis outline

The thesis is structured as follows: Chapter 2 presents the necessary definitions and other preliminaries around the subject of RDF graph summarization, including definitions of the main RDF concepts. Chapter 3 provides a review of the existing works around RDF graph summarization, the quality assessment of RDF graph summarization methods and approximate frequent pattern mining.

The next three chapters introduce the main contributions of this thesis, starting with Chapter 4 that describes our approach for RDF graph summarization and including both the pre-processing of the data and the post processing of the results in order to construct a summary that is also a valid RDFS. Chapter 5 presents a parallel, cloud-based algorithm for approximate pattern mining that that accounts for memory limitation problems of the original implementation, while Chapter 6 presents our Quality Framework for RDF Graph Summarization that can be used to identify the different quality characteristics of any summary.

Chapter 7 is dedicated to the experiment evaluation of the proposed algorithms and Quality Framework and presents an extensive and coherent set of experiments that try to evaluate the effectiveness and appropriateness of the algorithm, as well as of the issue of the quality of the produced results. Finally, Chapter 8 concludes this thesis by providing a summary of the thesis' contributions and some directions for future work, aiming at different research directions.

Chapter 2

Foundations and Technical Background

In this chapter we provide an overview of the foundations and the technical background to help the readers better understand the work presented in this thesis. In Section 2.1 we introduce the basic concepts of semantic web. Section 2.2 gives an overview of RDF data model and its components where we introduce vocabularies and ontologies in Section 2.2.2 and we describe the most well-known languages for building ontologies; RDFS and OWL. SPARQL query language is introduced in Section 2.3. We conclude this chapter with the Section 2.4 which provides a formal definitions and notations of the main terminologies used in this thesis.

2.1 Web and Semantic Web

2.1.1 World Wide Web

The World Wide Web (WWW) [17] is a system of interlinked hypertext documents and other resources accessed via the Internet. During its life cycle, the World Wide Web has passed through three phases: namely, the web of documents (Web 1.0), the web of people (Web 2.0), and the web of data (Web 3.0). The first generation of the WWW ,called, Web 1.0, was based on the following principles:

- The web pages are formatted and published with Hypertext Markup Language (HTML).
- All the web resources are uniquely identified by Uniform Resource Locator (URL).

- Hypertext Transfer Protocol (HTTP) is used to access to documents according to their URL.
- Hyperlinks among documents allow the user to navigate and access different web pages.

Based on the principles above, the publisher of a website can publish the data as documents in the HTML language on the web, users access to these documents by the HTTP protocol based on their unique URL, or through the hypertext link. Data published on the Web 1.0 can be found on different formats such as PDF, HTML tables, plain text, etc. The data in these formats can be understandable only by humans, they are not machine-understandable. In other words, when you request a web page, it is downloaded and visualized in HTML format. But the content of this page remains incomprehensible to the machine and this makes the process of obtaining data as autonomous entities directly from the document on the web in a structured format impossible.

The interaction between the users and the web sites led to the birth of Web 2.0. The Web 2.0 is the second generation of the WWW, which was introduced in 2004 as a dynamic and read-write web. The Web 2.0 focused on how information is shared among people, thus in the Web 2.0 users can interact with each other or contribute to the content. Web 2.0 facilitates user interaction and the creation of social networks. In this sense, Web 2.0 sites act more as points of presence, or user-centric web portals rather than Web 1.0 web sites. The Web 2.0 sites allow users to do more than just retrieve information. By increasing what was already possible with Web 1.0, they bring users new interfaces and new computer softwares. Users can now bring information to and control over Web 2.0 sites. In summary we can say with the emergence of Web 2.0, the process of using the WWW began to move towards interaction between the users and the system through different technologies such as wiki (Wikipedia, Seedwiki), Really Syndication Simple (RSS), web feeds service and social networks (Facebook,Twitter). The data in Web 2.0 are published in raw dumps, in different formats such as CSV, XML or marked up as (HTML) tables [18].

Despite the significant developments of web technologies that presented in the Web 2.0 and the moving from the static web to the dynamic web, the data published on the Web 2.0 are not in a machine-readable format, and are not connected, making them inaccessible to machine processing and still requiring humans for their interpretation. To this end, the key goal behind the third generation of the WWW, called Web 3.0, is to lift this distinction making web resources understandable both by the human

and the machine, thanks to the representation semantics of their contents. Web 3.0 is also known as the semantic web, with the gal of giving meaning to the entities and the relationships between them. Thus, we can say that the semantic web aims to transform the WWW from a web of document (an information space of linked documents) to a web of data (both documents and data are linked by typed links).

2.1.2 Semantic Web

Until the semantic web, the content of the existing web pages was understandable/usable by humans but not by machines/software. The term "Semantic" refers to a sequence of symbols that can be used to communicate meaning. Thinking about the role of semantics for automating approaches to exploit the web resources led to a new generation of the web termed as the semantic web. The aim of the semantic web is to represent knowledge about resources in a machine-readable way, where the data is connected by typed links and is described and stored using a generic triple-based format, called Resource Description Framework (RDF)[43]. In other words, the semantic web is designed to help machines for understanding and reasoning on the meaning of information published on the web. The goal is to set up, in addition to the network of hyperlinks between the classic web pages, a network of typed links between structured data.

As we mentioned above, the main goal of the semantic web is to express meaning. In order to achieve this, in the semantic web certain technologies and tools are proposed and used. All these technologies and tools are part of the semantic web stack in the Figure 2.1. This semantic web stack represents all semantic web technologies and it is the W3C[1] vision of semantic web architecture. All the technologies (languages and protocols) that it refers to, are standardized by the W3C. As shown in Figure 2.1, there are multiple layers and each layer benefits from the technologies of the layer below, and has a well-defined function in the architecture. Some of these layers are already implemented. Indeed, the languages and protocols that fulfill their functions already existed or were designed and created to meet the specifications of each layer. The top layers of the stack consists of technologies that are not yet standardized by the W3C or are only at the "recommendation" stage. The unifying Logic, Proof and Trust layers have not been implemented yet. They will build on each other to enable the identification and validation of information collected through RDF data. The purpose of the Crypto layer is to ensure and verify that the statements from the

[1]The World Wide Web Consortium (W3C) is an international community that develops open standards to ensure the long-term growth of the Web.

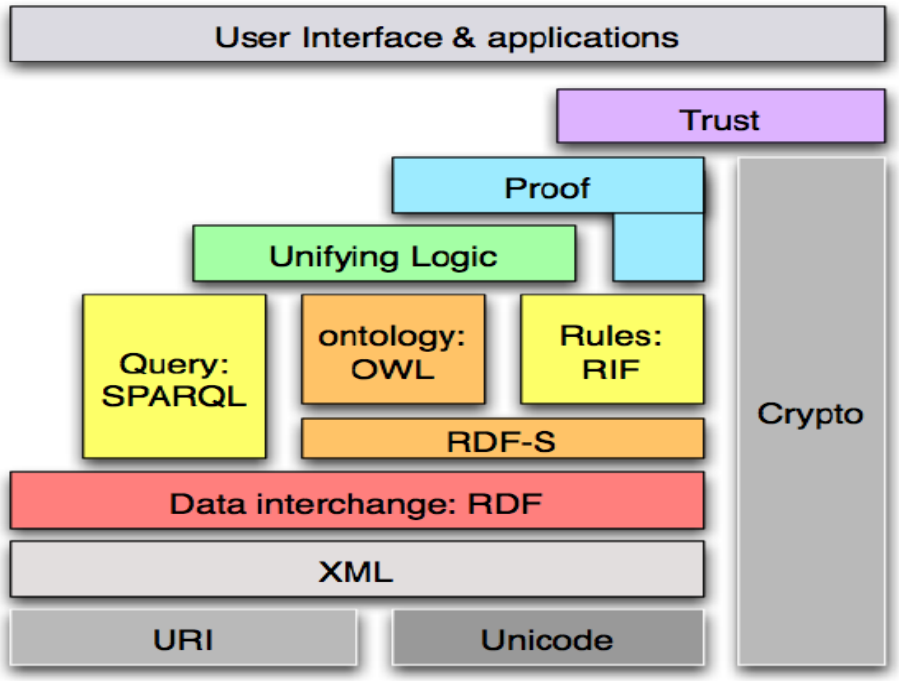


Figure 2.1 – Semantic Web Stack

semantic web come from a trusted sources. The user interface layer is the last top layer that allows humans to use semantic web applications. The bottom layers of the stack represent the basic hypertext web technologies (URI, Unicode, XML).

The middle layers contain the implemented and standardized semantic web technologies. The Data interchange layer represents the Resource Description Framework (RDF). The RDF is a language for describing web resources (any kind of real or abstract concept or thing) through triples of the form (subject, predicate, object). All resources are supposed to have unique URIs and are connected using a typed links, called properties (or predicates). Like resources, all properties have unique URIs. This way of describing resources is a major component in the W3C's semantic web activity, where an automated software can read, understand and exchange these machine-readable information distributed throughout the Web. RDF Schema (RDFS) is a model for RDF data, providing a data-modeling vocabulary [23]. The OWL layer represents the Web Ontology Language which is an RDF-based language. It extends the RDFS model by defining a rich vocabularies for the description of complex ontologies. The SPARQL (Protocol and RDF Query Language) is a protocol and a language that allows us querying the published RDF data in the Web.

In the following we will describe in more details the representative languages of semantic web, part of middle layers, RDF, RDFS, OWL and SPARQL and introduce

the concepts necessary to facilitate reading and understanding of the rest of the document.

2.2 The Resource Description Framework (RDF)

2.2.1 Data Model

The foundation of RDF is a model for representing the resources and the relations between theses resources. The RDF data model is the standard model for representing data on the Web in terms of triples of the form (s; p; o), explaining that the subject s has the property p, and the value of that property p is the object o. Each triple represents a statement of a relationship between two entities/resources and has three parts: The subject denotes a resource. The predicate denotes the binary relationship between subject and object and describes some aspects of the subject, while the object is the the value of the RDF triple. For example, the triple

< `http://dbpedia.org/resource/Pablo_Picasso`>
< `http://dbpedia.org/ontology/author`>
< `http://dbpedia.org/resource/Guernica_(Picasso)`>.

denotes that the resource `http://dbpedia.org/resource/Pablo_Picasso` which was defined by dbpedia.org[2] to represent the artist *Pablo Picasso*, has an author relation to the resource `http://dbpedia.org/resource/Guernica_(Picasso)` which was also defined by the dbpedia.org to represent the *Guernica* painting. The subject and the predicate are always expressed by URIs (with the exception of blank nodes, but this is beyond the scope of this thesis). The object can be expressed as a URI or a literal (or again a blank node). The literal can be of various data types, for instance, string, date, number etc.

To make the URIs more readable, short formats are used. Theses short format are called prefixes or *namespaces[22]*. For example If we define the following prefixes,

@prefix dbr : <`http://dbpedia.org/resource/`>
@prefix dbo : < `http://dbpedia.org/ontology/`>

then the above triple would read:

dbr:Pablo_Picasso dbo:author dbr:Guernica_(Picasso).

[2]http://wiki.dbpedia.org/

In this thesis we will also use **rdf:** as the namespace for <http://www.w3.org/1999/02/22-rdf-syntax-ns#> and **rdfs:** the namespace for <http://www.w3.org/2000/01/rdf-schema#>.

Based on the value of the their objects the triples can be classified into two categories:

- Object triple where the object of a triple is a URI. An example of the object type triples is the following:
  < http://dbpedia.org/resource/Guernica_(Picasso)>
  < http://dbpedia.org/ontology/museum>
  < http://dbpedia.org/resource/Museo_Nacional_Centro_de_Arte_Reina_Sof>.

- literal triple, where the object of a triple is a literal. An example of the literal triples is the following:
  < http://dbpedia.org/resource/Guernica_(Picasso)>
  < http://dbpedia.org/property/year>
  1937.

The intuitive way to view a collection of RDF data statements/triples is to represent them as a labeled, directed graph, called RDF graph, in which entities (subjects/objects) are represented as nodes and named relationships (properties/predicates) as labeled directed edges. Formal definition of the RDF graph given in Section 2.4.2

RDF data statements are usually accompanied with an ontology which provides a data-modeling vocabulary for RDF data. It defines set of classes C for declaring and describing the resources types and set of properties P for declaring and describing the resources relationships and attributes.

2.2.2 Vocabularies and Ontologies

In RDF, resources usually belong to classes that group them by types (persons, concepts, cars, etc.). They are qualified by properties (predicates) that define an aspect, a characteristic, an attribute, or a specific relation of these resources. These classes and properties are described in RDF vocabularies that then allow machines to understand and exploit them. Thus, we can define the Ontology as a set of concepts (classes) and relationships (properties) that are used to describe a particular domain with ability of inference. A Class is a category grouping multiple resources having common characteristics. Ontology provides a way for defining a metadata model that allows to:

- make sense of the properties associated with a resource;
- add constraints on the values associated with a property to also ensure its meaning. For example, if we have a property that represents an author, we want the values of that property to be a reference to a person (not a car or a house).

Some applications need a simple ontology consisting of few classes having many relations between them, while others may need more a complex ontology with thousands of classes and properties. Ontologies can be developed either using the RDF schema or the Ontology Web Language (OWL)for describing objects properties and their relationships.

2.2.2.1 RDF Schema

RDFS is an ontology definition language that can be used to define classes, properties, class and property hierarchies, and their behaviors. A class in RDFS corresponds to the generic concept of a type grouping multiple resources with common characteristics, somewhat like the notion of a class in object-oriented programming languages with some differences. For example, and not limited, in the RDF modeling an instance of a class is just a resource URI without any value (e.g., the *dbr:Pablo_Picasso* is an instance of *dbo:Artist* class regardless of any property related to it); a resource may belong to different classes which are not necessarily related. The instances of a same class may have very different properties, while it is not necessary to find a class on which the union of these properties is defined. A class is identified by a URI where to specify that a URI/resource is a class, it must define a triple where its subject is the URI/resource, the predicate is the predefined property *rdf:type*[3] while the object is the predefined RDF class *rdfs:Class*. For example, in order to declare that dbo:Artist is a RDFS class we should define the following triple:

dbo:Artist rdf:type rdfs:Class.

Then to clarify that a resource is an instance of a class, you must state that this resource has for rdf:type this class. Then we can state that *dbr:Pablo_Picasso* and *dbr:Rembrandt* are instances of the *Artist* class by following triples:

dbr:Pablo_Picasso rdf:type dbo:Artist.
dbr:Rembrandt rdf:type dbo:Artist.

[3]rdf:type property is used to to declare that resource I is an instance of class C using the triple in form: I rdf:type C

A property is defined by: (1)a name; (2) a domain (types of resources on which the property may bear) ; (3) a range (types of objects allowed for the property). As all the RDF resources, it is identified by a URI. A URI/resource P can be defined as a RDFS property using a triple of the form:

P rdf:type rdfs:Property.

For example to state that the *ex:paints* is a RDFS property, we can use the following triple:

ex:paints rdf:type rdfs:Property.

The basic elements of RDF Schema are:

- rdfs:Class. As we have seen above this is the set of resources that are RDF classes.
- rdfs:Property. This is the class of RDF properties.
- rdfs:domain. It is used to define the type of allowed subjects for a particular property. We can achieve that by defining a triple where its subject is the particular property and the predicate is the *rdfs:domain*, while the object is the class which is the domain of this property.
- rdfs:range. It is used to define the type of allowed objects for the property. In other words, it is used to state that the values of a particular property should be instances of a specific class. We can do that by defining a triple its subject being the URI of the particular property and the predicate being the *rdfs:range* while the object is the class which is the range of this property.
- rdfs:subClassOf. It is used to declare that a class is a subclass of another class. A class can be subclass of one or more classes, where any instance of the subclass is instance of all superclasses. To declare that a class C1 is a subclass of a class C2 we use a triple of the from:

  C1 rdfs:subClassOf C2.

- rdfs:subPropertyOf. It is used to state that a property is subproperty of another property, which means that all the resources related by the first property are also related by the second property. To declare a property p1 as a subproperty of a property p2 we use a triple of the from:

p1 rdfs:subPropertyOf p2.

RDF Schema can also be represented as a directed labeled graph, where the labeled nodes represent the classes and the labeled edges represent properties relating class instances. Formal definition of the RDF schema graph given in Section 2.4.2

2.2.2.2 Web Ontology Language

The Web Ontology Language(OWL) [104, 105] is designed as an extension of the RDF and the RDF Schema. Like the RDFS OWL is designed for the description of classes and types of properties but it is more expressive than RDFS, to which some lack of expressiveness due to the unique definition of relations between resources by assertions. As we have seen in the previous section, the concepts of class, resource, literal, and properties of the subclasses, subproperties and application domains already present in RDFS. In contrast to RDFS, the OWL adds the concepts of equivalent classes, of equivalent property, of equality of two resources, of the property opposites, property restrictions, of symmetry and cardinality. For instance, in OWL and unlike in RDFS, we can add cardinality and value constraints in order to restrict the range of the property and the number of values a property can take respectively. For example we can use *owl:maxCardinality* to describe that all the resource having the property P must have at most N distinct values. Moreover, in OWL one can use *owl:sameAs* to state that two resources belonging to two different sources are equivalent, *owl:equivalentClass* to state that two classes are equivalent (both represent exactly the same set of instances) and *owl:equivalentProperty* to state that a two properties are equivalent.

2.3 RDF Query Language

SPARQL[4] is the standard W3C query language used to query RDF graphs. It is a language that is very similar to the Structured Query Language (SQL), which also includes keywords and constructs, but it is extended to manage relationships that are not defined in the manner of conventional SQL data schemas. It consists of clauses, keywords and expressions like predicates and functions, many of which will be familiar for the SQL users (like Select, FROM, WHERE, ORDER BY, SKIP LIMIT, AND, GROUP BY, HAVING, AVG). Unlike SQL, SPARQL is about expressing graph

[4]https://www.w3.org/TR/rdf-sparql-query/

patterns. Using SPARQL, users can write queries that consist of a set of triple patterns. Each triple pattern has three components that correspond respectively to the subject, the predicate, and the object. Each of the components of a triple pattern can be either a constant or a variable. Variable names are preceded by a question mark.

2.3.1 BGP Queries

We consider the well-known subset of SPARQL consisting of (unions of) basic graph pattern (BGP) queries, modeling the SPARQL conjunctive queries. Subject of several recent works [41, 95, 40, 81, 24], GP queries are the most widely used subset of SPARQL queries in real-world applications [81, 62]. A BGP is again a set of triple patterns, or triples in short. Each triple has a subject, property and object, some of which can be variables.

2.4 Graph concepts and notations

In this section, we give formal definitions and notations of the main terminologies used in this thesis. Section 2.4.1 presents the basic graph concepts and notations used in this thesis while the Section 2.4.2 introduces the foundations of RDF and RDFS, which are useful for defining later on some concepts in our work.

2.4.1 Basic graph concepts and notations

Let A be a label set. We denote by $G = (V, E)$ an A-labeled directed graph whose vertices are V, and whose edges are $E \subseteq V \times A \times V$.

A fundamental graph notion which has been frequently exploited for graph summarization is the Quotient graph:

Definition 1 (Quotient graph) *Let $G = (V, E)$ be an A-labeled graph and $\equiv \subseteq V \times V$ be an* equivalence relation *over the nodes of V. The* quotient graph of G using $\equiv$, *denoted $G_{/\equiv}$, is an A-labeled directed graph having:*

- *a node n_S for each set S of equivalent V nodes;*
- *an edge (n_{S_1}, l, n_{S_2}) iff there exist two nodes $n_1 \in S_1$ and $n_2 \in S_2$ such that the edge $(n_1, l, n_2) \in E$.*

A particularly interesting notion of equivalence in a labeled directed graph is based on bisimulation [45]. Bisimilarity in a directed labeled graph is an Equivalence Relation defined on a set of nodes N, such that two nodes (u, v) are *bisimilar* if and only if the set of outgoing edges of u is equal to the set of outgoing edges of v and also, all successor nodes of u and v must be bisimilar (in other words, the outgoing paths of u and v are similar). We call the *bisimilarity* relation when defined based on outgoing paths, Forward (FW) bisimulation, and when it is based on incoming paths, Backward (BW) bisimulation. And if a relation is both FW and BW bisimulation we called it a forward backward bisimulation of FWBW bisimulation. A formal definition of the FW and the BW bisimulation are given below:

Definition 2 (Backward Bisimulation) *In a labeled directed graph, a relation $\approx_b$ between the graph nodes is a **backward bisimulation** if and only if for any $u, v, u', v' \in V$:*

1. *If $v \approx_b v'$ and v is a root, then v' is also a root;*
2. *If $v \approx_b v'$ and v' is a root, then v is also a root;*
3. *If $v \approx_b v'$, then for any edge $u \xrightarrow{a} v$ there exists an edge $u' \xrightarrow{a} v'$ such that $u \approx_b u'$;*
4. *If $v \approx_b v'$, then for any edge $u' \xrightarrow{a} v'$ there exists an edge $u \xrightarrow{a} v$ such that such that $u \approx_b u'$.*

Definition 3 (Forward Bisimulation) *In a labeled directed graph, a relation $\approx_f$ between the graph nodes is a **backward bisimulation** if and only if for any $u, v, u', v' \in V$:*

1. *If $v \approx_b v'$ and v is a root, then v' is also a root;*
2. *If $v \approx_b v'$ and v' is a root, then v is also a root;*
3. *If $v \approx_b v'$, then for any edge $v \xrightarrow{a} u$ there exists an edge $v' \xrightarrow{a} u'$ such that $u \approx_b u'$;*
4. *If $v \approx_b v'$, then for any edge $v' \xrightarrow{a} u'$ there exists an edge $v \xrightarrow{a} u$ such that such that $u \approx_b u'$.*

It easily follows from the bisimilarity definition that if $v \approx v'$ (for instance, if they are forward-bisimilar), then any label path that can be followed from v in the graph G can also be followed from v' in G and the other way around. In other words, the same paths start (respectively, end) in two bisimilar nodes. This condition is hard to meet in graphs that exhibit some structural heterogeneity: in such cases, every node is bisimilar to very few (if any) other nodes.

A more relaxed condition is *bounded- or k-bisimilarity*, requiring that the paths which exit (respectively, enter) a node be equal only up to an integer length k.

2.4.2 RDF graph concepts and notations

Let C, P, I and L be the sets of *class* Universal Resource Identifiers (URIs), *property* URIs, *instance* URIs and *literal* values respectively, and let T be a set of RDFS standard properties (*rdfs:range*, *rdfs:domain*, *rdf:type*, *rdfs:subClassOf*, etc.). The concepts of RDF Schema and RDF data graphs can be formalized as follows.

Definition 4 ***(RDF schema graph).*** *An* RDF schema graph $G_s = (N_s, E_s, \lambda_s, C, P, T)$ *is a directed labeled graph where:*

- N_s *is the set of nodes, representing classes and properties.*
- $E_s \subseteq \{(x, \alpha, y) | \, x \in N_s, \alpha \in T, y \in N_s\}$ *is the set of labeled edges.*
- $\lambda_s : N_s \longrightarrow C \cup P$ *is an injective node labeling function that maps nodes of* N_s *to class and property URIs.*

We note $\lambda_e : E_s \longrightarrow T$ *the edge labeling function that associates to each edge* $(x, \alpha, y) \in E_s$ *the RDFS standard property URI* $\alpha \in T$.

Definition 5 ***(RDF data graph).*** *An* RDF data graph $G_i = (N_i, E_i, \lambda_i, I, P, L, C)$ *is a directed labeled graph where:*

- N_i *is the set of nodes, representing instances, literals and* class *URIs .*
- $E_i \subseteq \{(x, \alpha, y) | \, x \in N_i, \alpha \in P, y \in N_i\}$ *is the set of labeled edges.*
- $\lambda_i : N_i \longrightarrow I \cup L \cup C$ *is a node labeling function that maps nodes of* N_i *to instance URIs,* class *URIs or literals.*

We note $\lambda_{ei} : E_i \longrightarrow P$ *the edge labeling function that associates to each edge* $(x, \alpha, y) \in E_i$ *the property URI* $\alpha \in P$.

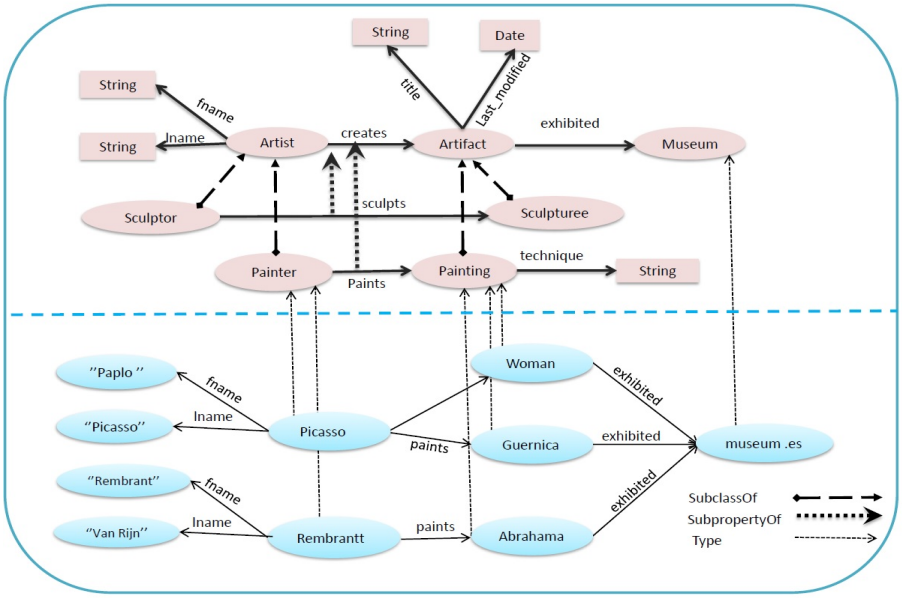


Figure 2.2 – Example of RDF Schema and data graphs [15]

Example 1 *The upper part of Figure 2.2 shows a visualization example of an RDF schema graph for the cultural domain, representing only class nodes, while properties are illustrated as edges between classes. For example, the class Artist denotes the set of resources which represent artists' entities, while the class Artifact denotes the set of resources which represent artifacts' entities. Note that properties serve to represent characteristics of resources as well as relationships between resources. For example the properties fname,lname represent the first name and the last name of an artist respectively, while property creates denotes that instances of the class Artist are related to instances of the class Artifact by a* create *relationship. Both classes and properties support inheritance, e.g., the class Painter is a subclass of Artist class while the property paints is sub-property of creates property. The lower part of Figure. 2.2 depicts an instance (data) graph building on this schema. This graph represents 6 different resources. For example the resource Picasso is an instance of the Painter class having properties fname, lname and paints.*

Path. We call a path in the RDF Instance Graph G_I a sequence of labeled edges labels $\{(n_1, a_1, n_2), (n_2, a_2, n_3)$,.....,$(n_v, a_v, n_(v+1))\}$.

Type edges. Edges labeled with *rdf:type* in the RDF data graph explicitly describe the type (class) of an instance, e.g. dashed edges in Figure.2.2, where for instance *Picasso* is declared to be of type *Painter*. We will note in the following the type edge label with τ. For an instance $x \in N_i$, we define $Types(x) = \{\lambda_i(y) | (x, \tau, y) \in E_i\}$

to be the set of types related to the node x via an explicit type edge definition, e.g., $Types(Picasso) = \{Painter\}$, while $Types(Guernica) = \{Painting\}$.

Properties. We denote by $Properties(x) = \{\alpha : \forall(x, \alpha, y) \in E_i : \alpha \neq \tau \wedge \lambda_i(y) \in I \wedge x \in N_i\}$, a set of labels of the non-Type edges which associate the node x with a set of entity nodes(nodes labeled by *instance* URIs).

Attributes. We denote by $Attributes(x) = \{\alpha : \forall(x, \alpha, y) \in E_i : \alpha \neq \tau \wedge \lambda_i(y) \in L \wedge x \in N_i\}$ a set of labels of the non-Type edges which associate the node x with a set of literal nodes(nodes labeled by *literal* values) ,

Example 2 *The set of properties associated with Picasso node in our example is* $\{paints\}$*, while the set of attributes of Picasso node is* $\{fname, lname\}$*.*

Definition 6 ***(Class Instances)*** *We denote by* $instances(c \in C) = \{\lambda_i(x) : \forall(x, \tau, y) \in E_i : y = c\}$ *a set of labels of the nodes which are associated to the node c (represent the class) via a typed edge* τ*, or in other words the set of resources(subjects) belonging to the class c.*

Definition 7 ***Property Instances.*** *We denote by* $instances(p \in P) = \{\lambda_i(x) : \forall(x, \alpha, y) \in E_i : \alpha = p\}$ *a set of labels of the nodes which are associated to other nodes via the property p, or in other words, is the set of resources (subjects) having the property p.*

Example 3 *The set of instances of the class Painting in our example is* $\{Woman, Guernica, Abraham\}$*, while the set of instances of the property exhibited (which is one of the Painting class's properties) is* $\{< Woman, exhibited, museum.es >, < Guernica, exhibited, museum.es >\}$

2.4.3 Knowledge Pattern

A knowledge pattern (or simply pattern from now on) characterizes a set of instances in an RDF data graph that share a common set of types and a common set of properties. More precisely:

Definition 8 ***(Knowledge Pattern)*** *A knowledge pattern* KP *in an RDF data graph is a quad* (Cl, Pr, Ins, SUP)*, where* $Cl = \{c_1, c_2, ..., c_n\} \subseteq C$ *is a set of classes,* $Pr = \{Pr_1, Pr_2, ..., Pr_m\} \subseteq P$ *is a set of properties,* $Ins \subseteq I$ *is the set of instances that have all the types of* Cl *and all the properties of* Pr*, and* $SUP = |Ins|$ *is called the support of the knowledge pattern in the RDF data graph (i.e. the number of instances that have all types and all properties).*

Patternid	classes C	Properties Pr	Instances Ins	Sup
p1	Painter	fname, lname, paints	Picasso, Rembrandt	2
p2	Painting	exhibited	Woman, Guernica	2
p3	Painting	-	Abraham	1
p4	Museum	-	museum.es	1

Table 2.1 – Knowledge patterns example (computed based on the bisimilarity relation)

Pattern Instances. We denote by $instances(pa) = Ins$ a set of the original KB resources having the same set of the properties/types of the pattern pa, or in other words is the set of bindings for the $?a$ variable over the RDF data graph in the following SPARQL-like conjunctive pattern: $\{<?a, \tau, c_1>, <?a, \tau, c_2>, ..., <?a, \tau, c_n>, <?a, Pr_1, ?b_1>, <?a, Pr_2, ?b_2>, ..., <?a, Pr_m, ?b_m>\}$, e.g. instances(p2)= { Woman, Guernica }

Example 4 *Table 2.1 shows possible patterns which can be extracted from the RDF instance graph depicted in Figure 2.2 based on a FW bisimilarity relation.*

2.4.4 RDF Knowledge Base

An RDF Knowledge Base (RDF KB or KB for short) is a semantic system that is used to store schema and data described according to the RDF language. The data (instances) in an RDF KB might adhere to one or more schemas and can form one or many disjoint knowledge graphs. RDF KBs store two disjoint types of statements:

- statements that describe the semantics of a knowledge area in the form of a predefined controlled vocabulary, usually appearing as a set of classes and properties that form a semantic schema or an ontology and which we call the **TBox** and
- statements that describe properties and other facts about objects (that we call instances as described earlier) and which we call the **ABox**

Querying an RDF KB using one semantic query language like SPARQL, would return the schema and instance triples that match the graph pattern expressed in the query. Part of the results could be inferred by applying the axioms and rules that are declared in the TBox (schema) on the ABox (instances).

2.5 Summary

In this chapter we introduced the basic notions and formalisms that would be used in the rest of this thesis and would allow us more easily establish and describe in a common way both the related work and our contributions. We introduced the notions or RDF language, RDF Schema and RDF Knowledge Base while we also described useful terms like OWL, SPARQL, BGP and Knowledge Patterns.

Chapter 3

Related Work

This chapter gives an overview of state-of-the art approaches, techniques and tools in RDF graph summarization and those assessing the quality of the produced RDF summaries. We also survey the area of approximate frequent pattern mining algorithms since our proposal relies on such algorithm. At the beginning of this chapter we introduce the important summarization techniques for RDF graphs in Section 3.1. In Section 3.3 we present the the main frequent pattern mining approaches while the Section 3.2 provides a review of the existing works around quality metrics in graph summarization.

3.1 Graph Summarization

As we mentioned before the main problem that we address in this thesis is the problem of creating RDF summaries of RDF datasets. RDF Graph Summarization pertains to the process of extracting concise but meaningful summaries from RDF Knowledge Bases (KBs) representing as close as possible the actual contents of the KB. RDF summarization can be used in multiple application scenarios, such as graph exploration and visualization, query evaluation and optimization, schema discovery from the data, or source selection, as well as many other applications. In the literature, many approaches to build such summaries have been proposed. Some of these approaches only consider the graph data without the schema, some others consider only the schema, and some consider both. These approaches also differ in their usage scope and their output. Thus, in this section we review the various efforts proposing summarization techniques for RDF graphs. In Section 3.1.1, we review generic graph summarization approaches that have found some application in RDF graphs. While these have not been specifically devised for RDF, they have either been applied to RDF subsequently, or served as inspiration for similar RDF-specific proposals. while

in Section 3.1.2 we review the summarization techniques proposed specifically for RDF graphs.

3.1.1 Generic graph (non-RDF) summarization approaches

In this section we review generic graph summarization approaches. While these have not been specifically devised for RDF, they have either been applied to RDF subsequently, or served as inspiration for similar RDF specific proposals. Section 3.1.1.1 describes works proposed for facilitating visualization tasks, while Section 3.1.1.2 presents the graph summaries proposed as support for query evaluation.

3.1.1.1 Graph summaries for visualization

We begin with the summarization methods that can be achieved based on a *Grouping* method. These methods aggregate nodes into super-nodes and connect them with super-edges based on both structural properties and node attributes. This is done by grouping nodes that are structurally close and share similar attribute values. The main goal of these methods is to produce understandable concise graph representation in order to facilitate the visualization and to highlight communities in the input Ddta graph, which greatly facilitates its interpretation.

One of most popular algorithms for the labeled graph is SNAP (Summarization on Grouping Nodes on Attributes and Pairwise Relationships) [97] algorithm. SNAP is an algorithm for graph aggregation based on the descriptions of nodes and edges. Given a set of attributes the user is interested in, it produces a summary that groups the interesting nodes in the incoming graph, based on their roles with respect to the user-specified attributes. It begins by grouping nodes based on the set of user-selected attributes where all nodes belonging to the same group must have the same values for all set of attributes. Then it tries to divide the existing groups according to their neighbors' groups until the grouping is compatible with the relationships (all nodes belonging to the same group have the same list of neighbor groups). The super-nodes of the summary graph given by SNAP correspond to the groups, and the super-edges are the group relationships inferred from the node relationships within the selected edge types. Two super-nodes are connected by a super-edge if there is a pair of nodes, one from each group, connected in the original graph. Figure 3.1.a shows a simple RDF graph of researchers and their relationships, the nodes represent authors while the edges represent their relationships. Figure.3.1.b shows the summary graph of SNAP based on the research-filed attribute and the co-author

relation. This summary consists of four super-nodes/groups where the nodes in each super-node/group have the same values of research-filed attribute, and they have the same list of neighbor groups based on the co-authors relationship.

The initial property-based partitioning of SNAP may lead to summary graphs containing a large number of small groups, and, in the worst case, each node may end up in an individual group. Thus, for more flexibility, the authors also propose k-SNAP [98], where the homogeneity requirement for the relationships is relaxed and users are allowed to control (drill-down, roll-up) the sizes of the summaries. K-SNAP produces a summary graph with size K where and because of the possible different grouping strategies for k groups, the authors introduce a quality measure counting the minimum number of differences in participants of group relationships between the current grouping and a presumptive ideal grouping (the participation ratio for each two groups is 100% or 0%) of the same size. SNAP and K-SNAP algorithms require the nodes in each group to have the same attribution information, so the total number of possible attribute values cannot be too many, Otherwise, the size of summaries will be too large for users to explore.

K-SNAP allows summaries with different resolutions, but users may have to go through a large number of summaries until some interesting summaries are found. The second limitation of SNAP and K-SNAP is that they are only applicable for homogeneous graphs. In other words, they are only applicable for the graphs representing single community of entities (e.g., student community, readers community, etc.) where all these entities have to be characterized by the same set of attributes. Something is not suitable for the RDF graphs since they are usually heterogeneous and they might also contain nodes without attributes or with a different subset of attributes each time. They handle only categorical node attributes but in the real world, many node attributes are not categorical, such as the age of a user. Simply running the SNAP/K-SNAP on the numerical attributes will result in summaries with large sizes (at least as large as the number of distinct numerical values).

To generalize the K-SNAP from the categorical node attributes to numerical node attributes the authors of [113] introduce a new method, called CANAL, that automatically categorizes numerical attributes values based on both the attributes values and the link structures of nodes in the graph. To point users to the potentially most insightful summaries, the authors propose their measure which incorporates three tenets of interestingness:

1. Diversity, the number of strong relationships connecting groups with different attribute values, where strong relationships between groups with different at-

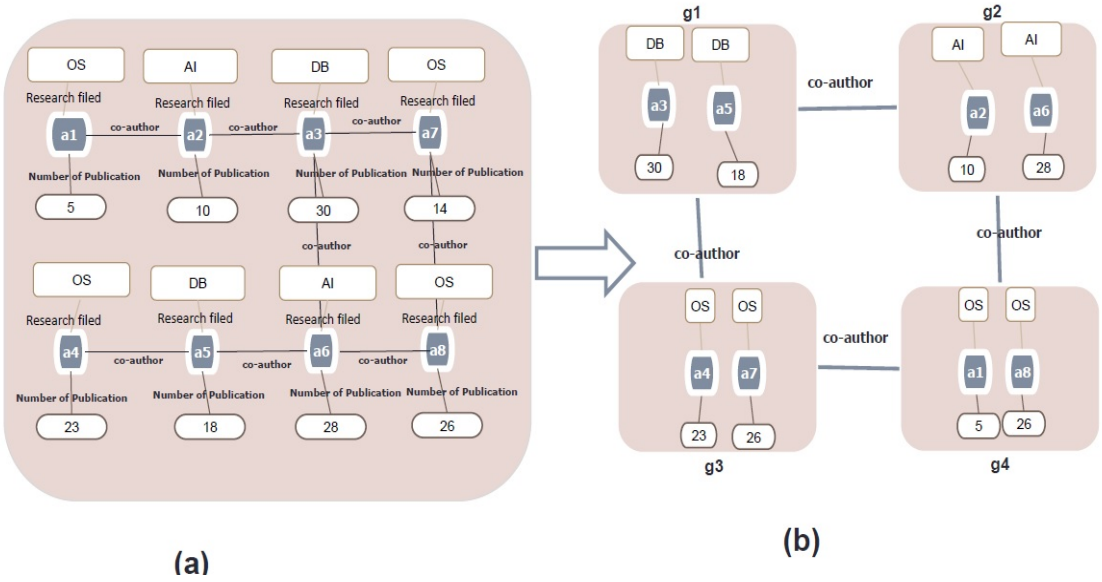


Figure 3.1 – Graph summarization by aggregation of SNAP :(a) graph about researchers ; and (b) its summary graph

tribute values reveal more insights into the original graph which makes the summary more informative;

2. Coverage, the fraction of nodes in the original graph that are present in strong group relationships. The more of nodes in strong group relationships, the generated summary will be more comprehensive;
3. Conciseness, the sum of the number of groups and the strong group relationships, where the summaries with fewer groups and strong group relationships are more concise, and hence are easier to understand and visualize.

Overall, interestingness is given as $\frac{Diversity(S)xCoverage(S)}{Conciseness(S)}$, where S is the summary graph. They evaluated the effectiveness of the CANAL algorithm and the Interestingness measure on two real datasets, the DBLP[1] and the CiteSeer [2]datasets. The experiments showed that CANAL 2-cutoffs produce high-quality summaries that match the manually selected Manual 2-cutoffs for the k-SNAP. The experiments also showed that the very small and the large k values often result in summaries with low interestingness values. It showed also that, there are two peaks which correspond to two types of interesting summaries:

1. The overall summary, with a small k value, concisely captures the general relationships among groups of nodes;

[1]http://dblp.uni-trier.de/
[2]http://citeseerx.ist.psu.edu

2. The informative summary contains more details that lead to new discovery of diverse relationships.

Comparably to k-SNAP, the summarization method provided in [96], is based on grouping nodes that have the same attribute values and share the same structural properties; given a user-specified space budget k, the algorithm creates a summary of size k by iteratively building groups (summary nodes) out of the original graph. Louati et al [64] propose a general tool for graphs summarization based on the k-SNAP. They use a classical Dynamic clustering or K-means algorithms in case it has no prior knowledge on the nodes (nodes not attributed). They propose two new aggregation criteria of evaluation which improve the quality of results while adopting the principle of k-SNAP in (Attribute-Relation)-groupement stage. These two criteria are based on the principle of common neighbors where the aim of these measures is to find the group that is not only little connected, but also containing a large number of nodes interacting with outsides to be split.

Although all above approaches can produce summaries for a directed labeled graphs, their main goal is to facilitate the visualization. In general, they are only applicable for homogeneous graphs while retrieved graphs on the semantic web are usually heterogeneous (graphs which are formed by different types of entities). In addition those summary graphs are not obviously RDF graphs. Furthermore, these approaches require the user to intervene to select a list of attributes and relations in order to split the set of nodes, something which have to be done dynamically by choosing the most effective attributes for this splitting.

3.1.1.2 Graph summaries intended for query evaluation

A pioneer proposal in this area is the Dataguide [42]. The summary in [42] is created by extracting all possible paths from a data graph. The generated summary is used as a covering index to answer queries from this index directly without referring to the original graph. A node can appear in the extent of more than one index node, allowing the index graph to be exponential in the size of the data graph in the worst case. However, this work relies on the graph being rooted, this does not fit the RDF graph who has no such structural constraint on the data.

In 1-index [70] approach nodes having the same set of incoming paths are grouped together to obtain the index graph. Compared to DataGuide, the size of the summary has an upper bound dependent on the length of the longest acyclic path. However, the size of the 1-index summary may become very large with the irregular and heterogeneous input data graphs. Furthermore, this work also relies on the data graphs

being rooted. To deal with the size problem of the 1-index the authors of proposed their k-index summarization. They create the summary graph based on the concept of k-bisimilarity in which only paths whose length are no longer than k are considered. Like the i-index they assume to have a rooted data graph.

The works proposed by [33, 10, 11, 37] are based on the intuition that nodes that have similar AxPRE[3] neighborhoods should be grouped together in the same extent. However, the size of this summary can approach the size of the data graph itself. They rely on the tree nature of XML data and on acyclicity assumptions that do not always hold for RDF datasets. The main problem with all these methods is that they assume there is only one single root node for the data, and every node in the graph has only one parent, and that for every node there is only one unique path that this node can be reached from. This may not hold for RDF graphs because first of all, RDF data in its nature forms a graph, which means there is no single root for the data. Second, RDF may contain loops or cycles that should be taken in consideration when computing the structural summaries.

A different approach towards using a summary for query evaluation is taken by [74, 54]. Given a graph G, they propose to produce a summary S which groups G's nodes into supernodes and its edges into superedges, together with a set of edge corrections C, such that applying the corrections on the "decompressed" (unfolded) summary S allows to retrieve exactly G. An intuition for this method is that S attempts to identify the regularity (repeated structure) in G whereas C stores the irregularities which make G diverge from "copies" of its summary. For a graph G there are many possible (S;C) pairs; the authors show how to find the one having the smallest total size.

[110] uses a frequent pattern mining algorithm to identify frequent subgraphs and store extent information from those, so it answers queries by asking for the respective part of the graph. This approach by design does not consider all data and is based on numeric information as opposed to our summaries. [115] is very similar, but considers trees instead of graphs.

Finally, Fan et al [38] propose their graph summarization technique, which compresses graphs while preserving query results. The summary graph can be directly queried without decompression rather than to restore the original graph. Query preserving graph compression is a triple $< R, F, P >$, where R is a compression function, F is a query rewriting function and P is a post-processing function. For any graph G, the summary graph is $G_r = R(G)$, $Q' = F(Q)$, $Q(G) = P(Q'(G_r))$. Any query

[3]Path regular expression on binary relations

evaluation algorithm for the query Q can be directly used to compute $Q'(G_r)$, without decompressing G_r, where the post-processing function P finds the answer in the original graph G by only accessing the query answer Q' in the compressed G_r and an index on the inverse of node mappings of R. The authors propose two parallel graph compression strategies, targeting different kinds of queries: (i) reachability queries, where we seek to know if one node is reachable from another; the authors show that they achieve a compression ratio of 95% for these queries; (ii) graph pattern queries, for which they attain a compression ratio of up to 57%. It should be stressed that the authors consider these queries under non-standard, more lenient semantics than the ones usually applied.

3.1.2 RDF Graph summarization

RDF graph summarization has been intensively studied, with various approaches and techniques proposed to summarize RDF graphs. These approaches can be grouped, based on the conceptual and algorithmic notions they are based on, into four main categories:

1. **Structural methods**: This category contains summarization algorithms which are prominently based on the graph structure (like the paths and subgraphs one encounters in the RDF graph) for generating their summaries. Structural summarization of RDF graphs aims at producing a summary graph, typically much smaller than the original graph, such that certain interesting properties of the original graph (connectivity, paths, certain graph patterns, frequent nodes, etc.) are preserved in the summary graph. Intuitively, each summary node corresponds to (or represents) multiple nodes from the input graph, while an edge between two summary nodes represents the relationships between the nodes from the input graph, represented by the two adjacent summary nodes.
2. **Pattern mining methods**: This category contains summarization algorithms which are based on data mining techniques for extracting the frequent patterns from the RDF graph, and use these patterns to represent the original KB in the summary. The existing approaches mainly attempt to identify frequent patterns or rules, which become representative nodes (supernodes), and thus reduce the size of the input graph and increase the query efficiency.
3. **Statistical methods**: This category contains the methods that summarize the contents of a graph quantitatively. The focus is on counting occurrences, such as counting class instances or building value histograms per class, property and

value type; other quantitative measures are frequency of usage of certain properties, vocabularies, average length of string literals, etc.Statistical approaches may also explore (typically small) graph patterns, but always from a quantitative, frequency-based perspective. While pattern mining aims to discover new patterns, the statistical methods quantitatively describe some known patterns.

4. **Hybrid methods**: This category contains the works that combine structural, statistical and pattern-mining approaches in order to build the summary.

3.1.2.1 Structural RDF summaries

We begin with the summarization methods that are based on the graph structure like the connectivity, paths, structural properties and node attributes, etc.. This is done by grouping nodes that are structurally close and share similar attribute values. An overview of the structural summaries is shown in Table 3.1.

Section 3.1.2.1.1 discusses the summarization methods based on the notion of bisimulation, while Section 3.1.2.1.2 is concerned with other structural summarization methods.

3.1.2.1.1 (Bi)simulation RDF summaries The classical notion of bisimulation (Section 2.4.1) has been used to define many RDF structural summaries. **ExpLOD** [55, 56] is an RDF graph summarization algorithm and tool that produces summary graphs for specific aspects of an RDF dataset, like class or predicate usage. The summary graph is computed over the RDF graph based on a forward bisimulation that creates group nodes based on classes and predicates. Two nodes v and u are bisimilar if they have the same set of types and properties. The generated summaries contain metadata about the structure of the RDF graph, like the sets of used RDF classes and properties. Some statistics like the number of instances per class or per property are aggregated with this structural information. Their summary graph is generated by the following mechanism:

1. Transform the original RDF dataset into an unlabeled-edge graph called ExpLOD-graph, where a node is created for each triple in the original RDF graph, labeled with the triple's property; unlabeled edges go from the original triple's subject and object, to the newly constructed property node. The edges of this new labeled graph are unlabeled and all nodes of this new labeled graph have a hierarchical label For example the predicate foaf:name may be represented as a node with hierarchical label *"P/foaf/name"* (for $Predicates/ < Vocabulary > / < identifier >$) or *"P/foaf"* depending on the desired granularity;

Work	RDF input	Input requirements	Handles implicit data?	Subsumption relationship	Purpose	Output type
ExpLOD [55, 56]	Instance	None	N	N	Data exploration	Graph
Campinas et al. [26]	Instance	None	N	N	Query formulation	RDF graph
Consens et al. [32, 58]	Instance	None	N	N	Query answering	RDF graph
Khatchadourian et al.[57]	Instance	None	N	N	Data exploration	Graph
Schatzle et al. [88]	Instance	None	N	N	Graph reduction	Graph
ASSG [112]	Instance	Required user-selected queries	N	N	Query answering	Compressed graph
Čebirić et al. [28, 29, 30]	Instance and schema	None	Y	N	Query optimization, visualization	RDF graph
Jiang et al. [48]	Instance	Type information, Each subject has exactly one type;	N	N	Semantic mining	Labeled graph
Picalausa et al. [81]	Instance	None	N	N	Indexing, query answering	Graph
Tran et al. [99]	Instance	neighborhood size	N	N	Indexing, data partitioning, query processing	Graph
SchemEX [59, 60]	Instance	Data stream window size, Type information	N	N	Indexing	RDF graph
Queiroz et al. [85]	Schema	Required schema, Parameterized user input, RDF/OWL	N	N	Visualization	Labeled graph
Kellou et al. [53, 52]	Instance and Schema	None	N	Y	Schema discovery	Graph
KCE [80, 72]	Instance and schema	Required schema, Parameterized user input, RDF/OWL	Y	N	Visualization	Isolated nodes
RDFDigest [102, 100, 78]	Instance and schema	Required schema, Parameterized user input, Semantics-aware	Y	N	Visualization, query answering tasks	Labeled graph

Table 3.1 – Structural RDF summaries.

2. then, the ExpLOD-graph is summarized by a forward bisimulation quotient, grouping together nodes having *the same RDF usage*. RDF usage can be the set of classes to which an instance belongs to or/and the set of properties describing an instance.

There are two sequential implementations of ExpLOD. The first implementation constructs usage summaries of datasets that fit in main memory. This approach computes

the relational coarsest partition of a graph using a partition refinement algorthim [75]. The second implementation uses SPARQL queries against an RDF triple store. Although the second implementation gives ExpLOD some additional scalability, it is slow due to the number of queries hat the store needs to answer. So, ExpLOD is limited to datasets that can be downloaded in main memory which limits the size of datasets for which usage summaries could be generated. Thus, and in order to deal with this limitation, the authors of [57] extend the ExpLOD approach and they propose a novel, scalable mechanism to generate usage summaries of billions of linked data triples based on a Hadoop-based implementation. The big disadvantage of all these approaches is the need for transforming the original RDF KB into a ExpLOD-graph, which requires the materialization of the whole dataset and this can be limiting in cases of large KBs. The created summary is not necessarily an RDF graph itself. If the summary is not an RDF graph, it would not be possible to use the same RDF standard tools (e.g. SPARQL) to query the summary.

To assist users in query formulation, Campinas et al. [26] are creating their own RDF summarization graph, whose nodes represent a subset of the original nodes based on their types or used predicates. This summary graph is generated by the following mechanism: (1) extract the types and predicates for each node in the original graph; (2) group the nodes having exactly the same set of types into the same node summary where two nodes, one of type A and one of types A and B, will end up in different disjoint summary nodes; (3) group based on attributes only if a node does not have a class definition. Like ExpLOD, a summary node is created for each combination of classes, i.e., two nodes, one of type A and one of types A and B, will end up in different disjoint summary nodes. Some statistics like the number of instances per class or the number of property instances are aggregated with this summary graph. Unlike ExpLOD, the summary nodes are not further partitioned based on their interlinks (properties), i.e., two nodes of type A, one has a, b and c properties and one has a and d properties will end up in the same summary node. Unlike ExpLOD, their summary graph is an RDF graph, where and in order to represent it as an RDF graph, they propose an RDF vocabulary, depicted in Figure 3.2 which makes it compatible for storing in RDF databases and be queried by SPARQL.

Schatzle, et al [88] propose a summarization approach for generating a summary graph of an RDF graph based on FW bisimulation relation(recall definition 3), where the nodes which have the same set of the outgoing paths are grouped in the same correspondence equivalence class. This algorithm is similar to [12] by using the same bisimulation algorithm. The main difference is that in this approach they provide

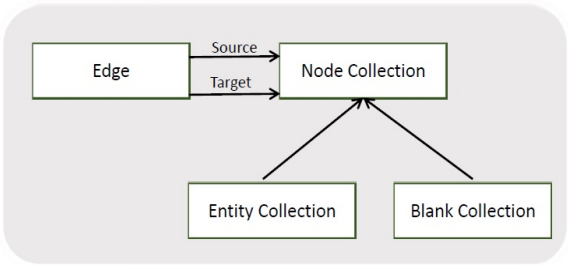


Figure 3.2 – RDF vocabulary for the data graph summary

two implementations for this bisimulation algorithm, one for sequential execution on a single machine using SQL and the other for distributed execution, taking advantage of MapReduce parallelization to reduce running time. Unlike our approach their summary is not an RDF graph.

S+EPPs [32], is a system that constructs a summaries based on FWBW bisimulation relation (recall definitions 3 and 2). It aims to construct a FWBW summary of a large graph with time similar or approximately similar to the time required to load the original KB plus write the summary. For this aim the authors propose in [58] an implementation in the GraphChi [61] a multi-core processing framework that supports the Bulk Synchronous Parallel (BSP) [103] processing model; an iterative, node-centric processing model by which nodes in the current iteration execute an update function in parallel that depends on values from the previous iteration. Their summarization approach is based on the parallel, hash-based approach of [19] which iteratively updates each node's block identifier by computing a hash value from the node's signature from the previous iteration to which the node's neighbors belong to. The main idea is that two bisimilar nodes will have the same signature, the same hash value, and thus have the same block identifier. The main problem of this approach is that as the size of the neighborhood increases, the size of summary grows exponentially and can be as large as the input graph.

Jiang et al. [48] propose two methods for summarizing RDF graphs. The first one, called *equivalent compression*, is based on grouping the nodes sharing the same type and the same set of edges adjacent labels (properties). In the second method, called *dependent compression*, two nodes n_i and n_j of the original RDF graph are grouped together if n_i is adjacent only to n_j, or vice-versa. The main limitation of this approach is that it does not consider the untyped resources, since it assumes that all the subjects and objects are typed. But in reality, many RDF datasets suffer from

the absence of type information. For example, only 63.7% of the data have complete type declarations in DBpedia, and 53.3% in YAGO[79].

The authors of [28, 29, 30] adapt the idea of the bisimilation to two characteristic features of RDF graphs: (*i*) the presence of type triples, and (*ii*) the presence of *schema triples*. They propose an algorithm summarizing the structure of the RDF graph. They group together nodes and edges that follow the same connection format and attributes relationship. For example, two subjects u and v will be put in the same subject block, if they share the same types and have the same set of properties. The main contribution of this approach is that it focuses on the implicit data triples; the implicit data triples can be obtained by an immediate entailment step based on an RDFS constraint. In particular, SPARQL query answers must be computed reflecting both the explicit and implicit data. Thus, including the implicit data to the summary will ensure that a query that can be matched on the original graph, would also be matched on the summary. Like our approach the generated summary is an RDF graph. These approaches require the existence of schema information, unlike our approach which allows the creation of a summary representation of the KB regardless of the existence or not of schema information in it.

Based on [38] Zhang et al. propose an Adaptive structural summary for RDF graphs (ASSG in short) [112], ASSG produces a summary graph for a part of an DRF data graph depending on a bisimulation relation between nodes. The nodes in the original graph are grouped into corresponding equivalence classes, where the nodes which have the same label and the same rank will be grouped in the same equivalence class. The rank of each node of the original graph is calculated as follows:

$$rank(n)) = \begin{cases} 0, & \text{if n is a leaf} \\ 1 + max\{rank(m) : (n,m) \in E\}, & \text{otherwise} \end{cases} \tag{3.1}$$

The rank is 0 for leaves and grows up with the shortest distance between the node and a leaf. Also and unlike to our approach their summary is not an RDF graph.

[81] introduces a summarization method based on triple (not node) equivalence. The summary is an edge-labeled graph (not an RDF graph): its nodes are set of blocks where each block groups set of equivalent triples from the input, while edge labels indicate positions in which triples in adjacent nodes join. Thus, the summary is used for reducing the query join effort, by pruning any dangling triples which do not participate in the join. Since the index contains only information on joins, and nothing of the values present in the input graph, the query language is restricted to BGPs comprising of *variables in all positions*; further, these BGPs must be *acyclic*.

The main problem with all the above approaches that they are based on a bisimulation relation, thus as the size of the neighborhood increases, the size of bisimulation grows exponentially and can be as large as the input graph. Thus, as we aim for both complete and compact summaries, bisimulation based approaches are not a good fit.

3.1.2.1.2 Other Structural Summaries One of the challenges when working with federated data sources is the lack of a concise summary or description of what kind of data can be found in which data source and how these data sources are connected. This leads to a problem that for a given query it is not clear to which data sources this query should be sent in order to retrieve results. In order to solve this problem Mathias et al. produce a schema extraction approach for Linked Open Data (LOD) (and thus normally a federated data sources' scenario) called SchemEX [59, 60]. SchemEX is an indexing and schema extraction tool for distributed, web scale RDF graphs such as the LOD cloud. In order to build their index/ summary they proposed a stream-based computation approach depicted in Figure 3.3, which provides access to the complete RDF KB that is indexed. Then they look up the RDF type information for each instance as well as for each of its referenced instances to extract the actual used schema for this RDF KB and use this schema as an index. As a result, SchemEX produces a three-layered index, based on the resource types. Each layer groups input data sources of the LOD cloud into nodes, as follows: (i) in the first layer, each node is a single class c from the input, to which, the data sources containing triples whose subject is of type c are associated; (ii) in the second layer, each node, now named as *an RDF type cluster*, is a set of classes C mapped to those data sources having instances whose exact set of types is C; (iii) in the third layer, each node is an equivalence class, where: two nodes u and v from the input belong to the same equivalence class if and only if they have the exact same set of types, they are both subjects of the same data property p, and the objects of that property p belong to the same RDF type cluster. The restriction to a certain window size of the data stream typically leads to incomplete results, thus the choice of the appropriate window size is an essential parameter for the quality of the extracted index. Unlike our approach, the specific approach does not consider the untyped resources or in other words it requires the existence of type information for all the subjects of the datasets to generate the appropriate summary where it it assumes that each resource has at least one type.

With the goal of extracting a schema describing an RDF dataset, [53, 52] propose an automatic approach for schema extraction based on a density-based clustering

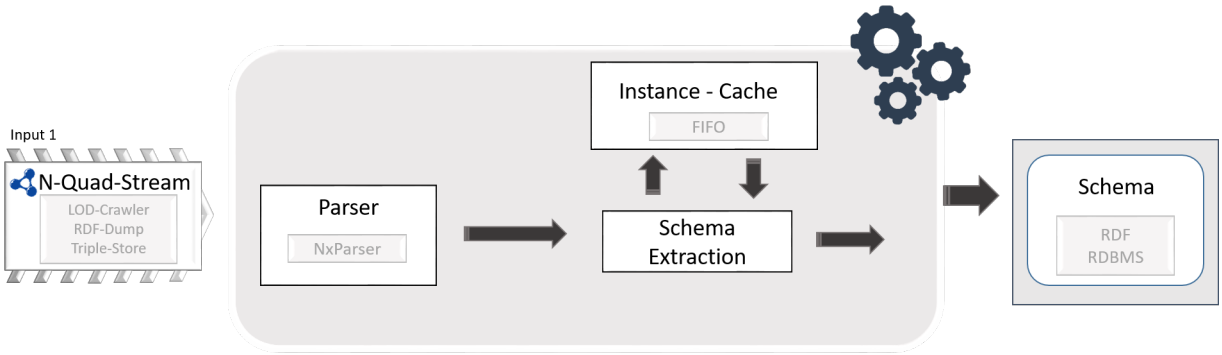


Figure 3.3 – Graph compression technique for SchemEX.

algorithm [36]. First and using the density-based clustering algorithm they extract the types describing a dataset where each of them is described by a *profile*, i.e., a set of (property, probability) pairs of the form $(\overrightarrow{p}, \alpha)$ or $(\overleftarrow{p}, \alpha)$, where α is the probability that a resource of that type has p as outgoing or incoming property. Then, the links between types as well as the hierarchical links through the analysis of type profiles are generated. There is a p-labeled edge from a type node T_i to a type node T_j, i.e., $T_i \xrightarrow{p} T_j$, where p is an outgoing property of T_i's profile and an incoming property of T_j's profile. Two node types T_i, T_j can be related by the *rdfs:subClassOf* based on hierarchical clustering algorithm applied to their profiles.

We now change our focus on the methods for ontology summarization (RDFS, OWL). These methods represent an ontology as a graph and then use some graph measures in order to find the key concepts of this ontology. One ontolgy summarization method, called RDFDigest, is introuduced in [101, 102, 100, 78]. It takes as input an RDF schema and an RDF data graph. Based on a graph centrality measures and the frequency of instances of concepts in the RDF data graph, RDFDigest identifies the most important concepts of the schema and links them in order to produce a valid subgraph of the input schema. In its first version [101], it identifies the most important concepts based on the relative cardinality, and the in/out degree centrality of the nodes. Then theses concepts are connected by adding edges which maximizes the most representative edges out of the whole input schema graph. The more recent version [78] uses a six well-known measures from graph theory (i.e., degree, betweennes, bridging centrality, harmonic centrality, radiality, and ego centrality [21]) in order to identify the most important schema concepts and adapting them for RDF/S KBs in order to consider instance information as well.

Work	RDF input	Input requirements	Handles implicit data?	Subsumption relationship	Purpose	Output type
Joshi et al. [50, 49]	Instance	None	N	N	Compression	Graph and logical rules
Pan et al. [77]	Instance	None	N	N	Compression	Graph and logical rules
Song et al. [90]	Instance	Bounded hop neighbors d, maximum size of patterns K,	N	N	Query answering	Graph patterns

Table 3.2 – Pattern mining RDF summaries.

[85] presents also an ontology summarization method based on the degree centrality and the closeness centrality measures. Given an ontology, the size of the summary and importance thresholds, it identifies the most important concepts in the ontology, based on the weighted sum of the the degree centrality and the closeness centrality. Then it links the selected concepts using a proposed Broaden Relevant Paths algorithm. This approach can deal with RDFS and OWL ontologies.

The authors of [80, 72] try to identify the key concepts in an ontology by combining cognitive principles with lexical and topological measures (the density and the coverage). The number of these concepts could be defined by human experts. Each ontology concept is assigned a score, which is a weighted sum of the scores assigned for each individual criterion; then the key concepts of the ontologies are taken to be those with the highest score. This approach extracts isolated (unlinked) schema elements. This approach also can work with the RDFS and OWL ontologies.

As we have seen all the mentioned ontology summarization methods aim at extracting key information from an already known ontology, while in this thesis we are interested in schema independent methods, which can summarize the RDF graphs regardless of having or not (OWL) ontology or RDFS triples.

3.1.2.2 Pattern- or rule-based RDF summarization

In this section, we discuss the summarization methods, which are based on data mining techniques, using them for discovering the frequent patterns in the RDF data graph, and use these patterns in the summary to represent the most important nodes and edges of the RDF graph. An overview of the methods in this category is shown in Table 3.2.

The algorithm in [90] takes as input an integer distance in d, which will be used to control the neighborhoods in which we will look for similar entities, and a bound k as the maximum number of the desired patterns and return the k patterns which

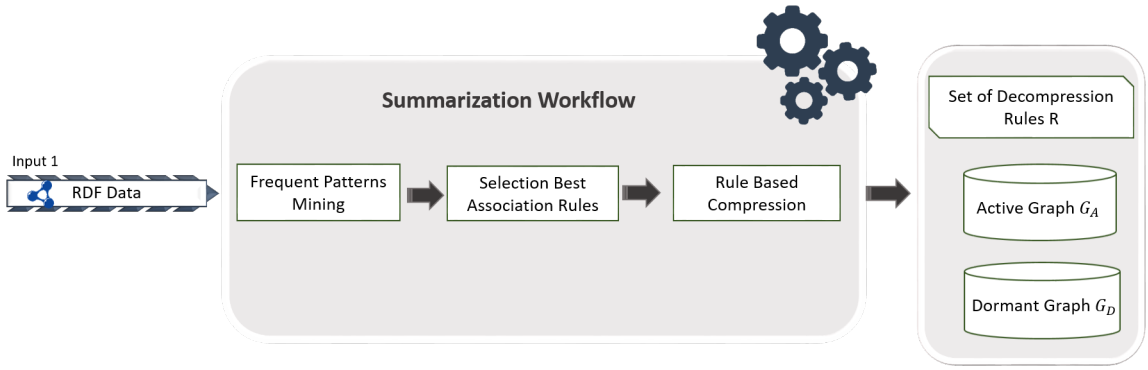


Figure 3.4 – Graph compression technique for Joshi et al. [50].

maximize an *informativeness* measure (an informativeness score function is provided as input). The authors use d-similarity to capture similarity between entities in terms of their labels and neighborhood information up to the distance d. Compared to other graph patterns like frequent graph patterns, (bi)simulation-based, dual-simulation-based and neighborhood-based summaries, d-similarity offers greater flexibility in matching, while it takes into account the extended neighborhood, something that provides better summaries especially for schema-less knowledge graphs, where similar entities are not equivalent in a strict pairwise manner. A node n of the original graph G is attributed to the base graph of the d-summary P, if and only if there is a node s of P which has the same label as n and for every parent/child s_1 of s in P, there exists a parent/child n_1 of n in G such that edges (s_1, s) and (n_1, n) have the same edge label. Then the d-summaries are used e.g. to facilitate query answering.

A d-summary P is said to dominate another d-summary P', if and only if $supp(P) \geq supp(P')$; a maximal d-summary P is one that dominates any summary P' that may be obtained from P by adding one more edge. The algorithm starts by discovering all maximal d-summaries by mining and verifying all k-subsets of summaries for the input graph G, then greedily adds a summary pair (P, P1) that brings the greatest increase to the informativeness score of the summary.

Methods described below use rule mining techniques in order to extract rules for summarizing the RDF graph. A common limitation of such methods is that, by design, the summary is not an RDF graph, thus it cannot be exploited using the common set of RDF tools (e.g., SPARQL querying, reasoning etc.)

[50, 49] propose compressing the RDF datasets by generating a set of logical rules from the dataset and removing triples that can be inferred from these rules. Thus, graph decompression infers such triples again, to retrieve the original graph. This approach, which is depicted in Figure. 3.4, generates, from a given RDF graph G,

an *active graph* G_A containing the triples that adhere to certain logical rules, and a *dormant graph* G_D, which contains the set of triples of the original graph which none of the identified rule can infer. This leads to viewing an RDF graph G as being $R(G_A) \cup G_D$, where R represents the set of rules to be applied to the active graph G_A, while (G_A, G_D) together represent the compressed graph. An association rule mining algorithm is employed to automatically identify the set of logical rules.

The authors leverage the frequent pattern mining algorithms Apriori [9] or FP-Growth [44] to identify sets of association rules. First, for each property p, a "transaction" (in classical data mining terms) is a list of objects which are the values of property p for a given subject. Each rule thus is defined by: a property p, an object item k, and a frequent itemset x associated with k. One sample rule is: $\forall x,\ (x, p, k) \rightarrow \bigwedge_{i=1}^{n}(x, p, v_i)$, stating that the subjects that carry the value k for property p, carry also the values u_i for the same property. Based on such a rule, the triple (x, p, k) is encoded in the summary while the inferred triples $\bigwedge_{i=1}^{n}(x, p, v_i)$ can be removed. Further, the authors extend the approach to use as a transaction, the lists of all (p, o) pairs for a given subject, and similarly mine for frequent itemsets in this context, each of which will be interpreted as a logical compression rule.

This approach works well when the original graph contains many different nodes sharing many same "neighbors", but it is not effective when the contrary is true. To deal with the last issue, the authors of [77] extend the previous approach by exploiting a graph pattern with two variables instead of one, which makes it applicable to more generic graph structures, reducing the size of the summary graph. This is because the number of triples in the summary graph is halved (a rule can now represent more triples).

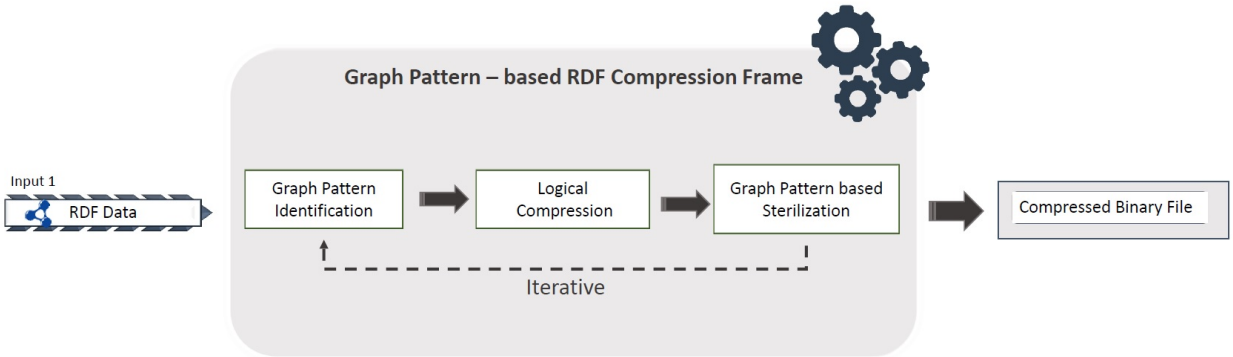


Figure 3.5 – Graph compression framework following [77].

Work	RDF input	Input requirements	Handles implicit data?	Subsumption relationship	Purpose	Output type
Hose et al. [46]	Instance	None	N	N	Source selection	Bloom filters Statistical information
Wu et al. [108]	Schema	Requires schema, RDF/OWL, minor user input	Y	Y	Visualization	Isolated schema nodes
Pires et al. [83]	Schema	Requires schema, OWL	N	Y	Query answering tasks	Labeled graph
LODSight [35, 73]	Instance and schema	None	Y	N	Compression, Visualization	Labeled graph
Presutti et al. [84]	Instance and schema	None	N	N	Querying Dataset	Labeled graphs

Table 3.3 – Statistical RDF summaries.

3.1.2.3 Statistical RDF summarization

In this section, we discuss the works focusing on quantitatively summarization of the contents of an RDF graph. An overview of these works is shown in Table 3.3.

A first motivation for statistical summarization works comes from the source selection problem. One solution for dealing with the source selection problem is using the SPARQl ASK query [89, 16], which asks each source if a result for a triple pattern exists in this source or not, then sends the query to all the sources which return a true value. The main problem of this solution is that many sources contain the same facts which means that we will have many duplicate results and therefore many unnecessary requests. The authors of [47] propose a strategy, which considers the problem of overlapping among sources, this strategy expands ASK so that it does not return just a boolean value but also a summary of results in the form of Bloom filters [20]. Based on these filters, it estimates the benefit of retrieving results for a triple pattern from a source, and ignores sources with low or zero benefit. Their idea is to extend ASK operation to provide a concise expressive summary of result bindings of each query variable (instead of boolean yes/no). They uses sketches to estimate the overlap among sources where each sketch has two different components:

1. a count of the number of results, possibly estimated from statistics available in the source's SPARQL endpoint;

2. a concise summary of the results. Where for a triple pattern p with variable set V (p), the sketch returned by a source S consists of the pair (c(p), S) where c(p) is the number of triples from T matching p, and S contains for each variable v

$\in$ V(p) a pair (c(v), B(v)), where c(v) is the number of distinct bindings for v and B(p) is a Bloom filter containing all bindings of v.

The experiments show that their approach has good effectiveness and efficiency for the Single Triple Pattern queries case and the Star-Shaped Queries case. But their approach is not effective for the complex graph patterns' queries (when the query contain at least two triple patterns and at least two different variables). That is because their summaries cannot be applicable for the whole query because there is no connection of the summaries of two different variables.

[108] proposes an algorithm for extracting the most important schema concepts. It takes as input an ontology and extract the most important concepts and relations in an ontology. As it can be easily inferred, this algorithm is based completely on the schema/ontology and does not consider the instances. The importance of concept based on the number of outgoing properties, its properties to more important concepts, and the weights of these properties. The approach considers implicit information. The approach works only with RDF KBs having a full schema, but in reality a lot of RDF KBs carry none at all or only partial schema information. Additionally, in the LOD cloud the number of KBs which do not use the full schema or they use multiple schemas has increased due to the absence of the schema information, which describes the interlinks between the datasets, and the combinatorial way of mixing vocabularies.

Another work for the ontology summarization is presented in [83]. The goal is to help peer clustering, where an incoming peer must search for semantically similar peers in order to join. To do that, a schema summary of the new node is compared with the schema summaries of the existing peers in order to decide where to join. The relevance of a concept is computed as a combination of a degree centrality and frequency measures. The algorithm starts by computing the relevance, then it selects the top-k nodes, and subsequently groups adjacent relevant concepts. Like the previous approach [108], this approach works only at the schema level, thus it dose not consider the RDF data level information.

LODSight [35] is an RDF dataset summary visualization tool that displays typical combinations of types and predicates. It relies solely on SPARQL queries and as such, given a SPARQL endpoint, it can theoretically summarize all accessible data, without requiring any user input. Through those SPARQL queries, it collects statistical information on the available combinations of types and predicates, and visualizes them in a labelled graph. Implicit RDF data is only accounted for to the extent that the endpoint returns full answers based on reasoning. The tool provides dynamic means

of changing the level of detail, and is able to summarize very large datasets. The system is available online[4].

LODSight was extended in [73] in order to further improve the understanding of a dataset, by instantiating the summary patterns identified by LODsight. To do that, the authors propose an approach to select instances through three methods, namely random, distinct and representative. In random selection, random examples of each RDF summary path are selected; this runs the risk of returning duplicates. The distance selection method aims to select data paths as distinct from one another as possible; to this effect, distance measures are used to find how similar two paths are, and a greedy heuristic is employed to construct a sufficiently diverse set of pairs.

[84] proposes an approach for extracting the main knowledge components of an RDF dataset based on recognizing and discovering patterns. Its main goal is supporting query answering. As such, the authors create initially an ontology that depicts the organization of the dataset and identifies its main features, i.e. information about triples, paths, and types and properties occurring in the paths. In addition, it includes statistics about these elements, such as the number of occurrences of each path. Using this ontology, the core types and properties can be distinguished based on their frequencies and the position in paths. According to these observations, central knowledge patterns (containing a central type and properties) are extracted in order to define prototypical queries.

3.1.2.4 Hybrid RDF summarization

We present here the RDF summarization approaches that combine methods from the structural, statistical and pattern mining categories. An overview of theses approaches is shown in Table 3.4.

With the purpose of reducing the size of a given RDF graph considerably without losing too much of the original inherent structure, [13] proposes a hybrid summarization technique for RDF graphs. It combines the bisimulation and clustering methods by having a bisimulation step followed by an agglomerative clustering step. The objective of the first step is to collapse equivalent structures based on the FW bisimulation relation; in the beginning, all subjects will be in one block and then the approach splits iteratively the blocks by computing the node's signature based on block identifiers from the previous iteration to which the node's neighbours belong to. This step continues until any two subjects of every block have the same signature, where the signature of a subject with respect to a certain partition P is the set of outgoing edges

[4] `http://lod2-dev.vse.cz/lodsight/about.html`

Work	Method	RDF input	Input requirements	Handles implicit data?	Subsumption relationship	Purpose	Output type
Alzogbi et al. [13]	Structural, clustering	Instance	None	N	N	Compression	Graph
Stefanoni et al. [93, 94]	Structural, data mining	Instance	Optional parameters for summary refinement	N	N	Conjunctive query cardinality estimation	Graph
ABSTAT [76] [91]	Statistical, pattern mining	Instance and schema	RDF/OWL, Semantic-aware	Y	N	Visualization, schema discovery	Abstract Knowledge Patterns
Glimm et al. [39]	Rule pattern mining	Instance and schema	Description Logics, Semantic-aware	Y	N	Query answering	Graph
Zheng et al. [116]	Structural quotient, pattern-mining	Instance and schema	Each instance should have a type, the list of meaning-equivalent instances should be provided	N	Y	Query optimization	Multi-layer Graph

Table 3.4 – Hybrid RDF summaries.

to objects in blocks of P: $sigP(s) = \{(a, B)|(s, a, o) and o \in B \in P\}$. The objective of the second step is to Collapse the similar structures, where it takes as input the extracted graph from the previous stage, and then apply a clustering agglomerative algorithm based on an similarity measure defined as follows :
$sim_k(v, w) = size(intersect(T_k(v), T_k(w)))/(size(T_k(v))+size(T_k(w)))/2$. Where $T_k(v)$ is the instance tree of the current summary graph G_r, which contains all nodes and edges which can be reached when following all possible paths in G_r starting at v and up to length K. It uses this measure to define the similarity matrix then apply hierarchical clustering to build an undirected, unweighted graph without self-loops or multiple edges. The generated summary is not an RDF graph. Furthermore, the FW bisimulation algorithm generates a large summary graph, which has a worst case size, the size of the original graph.

ABSTAT [92, 76, 91] presents a method for summarizing the RDF graphs containing subtype and subproperty triples. Based on the subtype and subproperty triples it extracts the minimal types for all the instances of the RDF dataset where the set of the minimal types of an instance x is the set the classes at the leafs of its hierarchical types. The summary is a set of isolated abstract knowledge patterns of the form $(c1, p, c2)$, stating that a dataset holds at least one resource of type $c1$ having the property p whose value is a resource of type $c2$. ABSTAT requires the presence of

RDF schema (triples) in order to work properly.

[93, 94] propose a summarization method based on grouping nodes having exactly the same set of types, outgoing and incoming properties. A labeled edge with the number of instances, that have been collapsed due to merging, is added for each summary node. The generated summaries may be too large; therefore, the authors propose an algorithm to reduce the summary to a target size specified by the user, by merging nodes that have similar incoming and outgoing properties. The similarity is determined by a Jaccard index, approximated by MinHashing [63]; to efficiently compute the similarity between all pairs of summary nodes, locality-sensitive hashing [63] is used. The approach gets as input only instances and optional parameters for summary refinement and returns an instance graph. This graph is further used to enable the estimation of the cardinality for easing query answering and evaluation.

[39] presents a method for abstracting the ABox of Horn ALCHOI ontologies obtained as a result of a fixed-point computation. The idea here is, instead of performing reasoning over a large ABox, materializing all the entailed information upfront for subsequent query answering, to reduce it to reasoning over a smaller *abstract* ABox. The computation of the abstract ABox involves (a) partitioning the individuals into equivalence classes based on told information and uses one representative individual per equivalence class, and (b) iteratively splitting (refine) the equivalence classes, when new assertions are derived that distinguish individuals within the same class. The aforementioned work considers implicit information in both instances and schema. The result of the whole process is an abstract, summarizing effectively the available instances.

Finally, [116] proposes a framework for mining equivalent structure patterns with equivalent semantic meaning. As in RDF KBs it is common to have different graph structures, sharing the same meaning, the authors aim is to ease end-user's querying task. As such, instead of demanding from the users to have the complete knowledge of the schema enumerating in the query all possible semantically equivalent graph structures, the authors propose an approach that performs query rewriting, exploiting automatically other possible graph structures with the same meaning. To achieve that, they define the notion of *semantic graph edit distance* and present a framework that tries first to rewrite the input query to one considering semantic equivalences and then finding the subgraphs minimizing the semantic graph edit distance. For the efficiency, they build offline a *semantic summary graph* over which they perform a two-level pruning at query time in order to finally provide answers. The semantic summary graph is a multi-layer graph where the first layer is consisted of the linked

types of the instances (they call them semantic facts). Then, they abstract this graph in the layers above, replacing/abstracting in each layer classes with their superclass. The aforementioned method does not consider the untyped instances and it can only be applied in fully typed RDF KBs.

3.2 Quality of RDF summaries

As discussed in the previous section, RDF graph summarization has been intensively studied, with various approaches and techniques proposed to summarize (semantic) RDF graphs. In Section 3.1 we presented the state-of-the-art approaches and tools that deal with problems related to RDF summarization. As we have seen in Section 3.1, these approaches come from various scientific backgrounds ranging from generic graph summarization to explicit RDF graph summarization, from visualization to query answering and from source selection to schema extraction problems. These approaches produce different results while applied on the same KB, thus a way to compare and evaluate the quality of the produced summaries is necessary. This would allow a better understanding of the quality of the different summaries and facilitate their comparison and decide on their quality and best-fitness for specific tasks. Despite the existence of this number of RDF summarization approaches, the RDF summarization methods proposed so far do not address in depth the problem of the quality of the produced RDF summaries. A noticeable exception is the work in [25], which proposes a model for evaluating the precision of the graph summary, compared to a gold standard summary, which is a forward and backward bisimulation summary. The main idea of the precision model is based on counting the edges or paths that exist in the summary and/or in the gold summary graph. The precision of a summary is evaluated in the standard way, based on the number of true positives (the number of edges existing in the summary and in the input graph) and false positives (the number of invalid edges and paths existing in the summary but not in the input graph).

The first limitation of this quality model [25] is that it works only with the summaries generated by an algorithm that uses a bisimulation relation. Similarly to our quality framework (see Chapter 6), they consider the precision at the instance level, i.e how many of summary class and property instances are correctly matched in the original KB. Unlike our work, this work does not consider the recall at the instance level, because it claims that the way summarization algorithms work, does not allow them to miss any instance. But this is not always correct, e.g. the approximate RDF

summarization algorithms like [117, 118] might miss a lot of instances. As it is well-known, the precision alone cannot accurately assess the quality, since a high precision can be achieved at the expense of a poor recall by returning only few (even if correct) common paths. Additionally and unlike our work, this model does not consider at all the quality of the summary at the schema level, e.g. what if one class or property of the ideal summary is missing or an extra one is added or a property is assigned to the wrong class. In all these cases, the result will be the same, while it is obvious that it should not. Finally, [25] is missing completely any notion of evaluating the connectivity of the final summarization result.

One more effort, [31], addressing the quality of hierarchical dataset summaries is reported in the literature. The hierarchical dataset summary is based on the grouping of the entities in the KB using their types and the values of their attributes. The quality of a given/computed hierarchical grouping of entities is based on three metrics:

1. the weighted average coverage of the hierarchical grouping, i.e. the average percentage of the entities of the original graph that are covered by each group in the summary;
2. the average cohesion of the hierarchical grouping where the cohesion of a sub-group measures the extent to which the entities in it form a united whole; and
3. the height of a hierarchical grouping, i.e. the number of edges on a longest path between the root and a leaf.

The main limitation of this approach is that it works only with the hierarchical dataset summaries, since metrics like the cohesion of the hierarchical groups or the height of the hierarchy cannot be computed in other cases. Moreover, the proposed groupings provide a summary that can be used for a quick inspection of the KB but cannot be queried by any of the standard semantic query languages. On the other hand and similarly to our quality framework, [31] considers the recall (namely the coverage) at instance level, i.e. how many of the instances of the original KB are correctly covered by the summary concepts. Contrary to our work, this model does not consider at all the quality of the summary at the schema level. Notions from [31] can also be found in the current work, where algorithms like [117, 118] that rely on approximation get penalized if they approximate too much, in fact loosing the cohesion of the instances represented by the computed knowledge patterns.

Besides that, only few efforts have been reported in the literature addressing the quality of the schema summarization methods in general [106, 82, 14], i.e. the quality of the RDF schema that can be obtained through RDF summarization. The

quality of the RDF schema summary in [82] is based on expert ground truth and is calculated as the ratio of the number of classes identified both by the expert users and the summarization tool over the total number of classes in the summary. The main limitation of this approach is that it uses a boolean match of classes and fails to take into account similarity between classes when classes are close but not exactly the same as in the ground truth or when classes in the ground truth are represented by more than one class in the summary. Works in schema matching (e.g. [106]) are also using to some extend similar metrics like recall, precision, F-Measure commonly used in Information Retrieval, but are not relevant to our work since even if we consider an RDF graph summary as an RDF schema, we are not interested in matching its classes and properties one by one; as stated above this binary view of the summary results does not offer much in the quality discussion. Additionally these works do not take into account issues like the size of the summary.

To the best of our knowledge, our work is the first effort in the literature to provide a comprehensive Quality Framework for RDF Graph Summarization, independent of the type and specific results of the algorithms used and the size, type and content of the KBs. We provide metrics that help us understand not only if this is a valid summary but also if a summary is different (and how much) from another in terms of the specified quality characteristics. And we can do this by assessing information, if available, both at schema and instance levels.

3.3 Approximate Frequent Pattern Mining

Since the algorithm proposed in this thesis is based on approximate pattern mining, we decided to briefly survey this area as well. While the focus of our work is not to contribute to approximate pattern mining, we perform a quick survey of this area in order to be able to better position our proposal and to justify some of the choices we made. Frequent pattern mining has been a focused theme in data mining research, the traditional exact model for frequent pattern requires that every item occurs in each supporting transaction. An intrinsic problem with the exact frequent pattern mining is the rigid definition of support, an itemset X is supported by a transaction T, if each item of X exactly appears in T, an itemset X is frequent if the number of transactions supporting it is no less than a user-specified minimum support threshold (denoted as $\min_{sup}$). However, in real applications, a database is typically subject to random noise or measurement error, which poses new challenges for the discovery of frequent itemsets. For example, in a customer transaction database, random noise

could be caused by an out-of-stock item, promotions or some special event like the world cup, holidays, etc. Such random noise can distort the true underlying patterns. Theoretical analysis shows that in the presence of even low levels of noise, large frequent itemsets are broken into fragments of logarithmic size, thus the itemsets cannot be recovered by the exact frequent itemset mining algorithms. In order to deal with noisy and large databases, the common approach is to relax the notion of support of an item set by allowing missing items in the supporting transactions, which poses new challenges for the efficient discovery of frequent patterns from the noisy data, the so called approximate pattern mining.

The classical definition of frequent itemsets requires that all the items of each mined set actually occur in the supporting transactions. In order to deal with noisy and large databases, the common approach is to relax the notion of *support* of an item set by allowing missing items in the supporting transactions. Different approaches proposed different cost functions which are tackled with specific greedy strategies. Asso [68] is a greedy algorithm aimed at finding the pattern set Π_k that minimizes the amount of noise in describing the input data matrix $\mathcal{D}$. This is measured as the L^1-norm $\|\mathcal{N}\|$ (or Hamming norm), which simply counts the number of 1 bits in matrix $\mathcal{N}$. The Hyper+ [109] algorithm also tries to minimize the patterns cost $\|P_I\| + \|P_T\|$ in order to find a compact pattern set. Finally, in [69] an information theoretical approach is adopted, where the cost of the pattern set and of the noise is measured by their encoding cost in bits.

PaNDa$^+$ was shown to be more computationally efficient, able to extract high quality patterns both from binary and from graph data [66], and that such patterns can be successfully exploited for other data mining tasks, e.g., classification [67]. Differently from other algorithms, PaNDa$^+$ allows to tune the maximum allowed row-wise and column-wise missing items (noise) accepted in each pattern. For these reasons, we adopted PaNDa$^+$ as a general approximate pattern mining algorithm.

3.4 Summary

In this chapter, we surveyed the relevant related work that exists in the literature along the three main axes our work tackles. We started by presenting a comprehensive state-of-the-art in RDF graph summarization. We introduced a taxonomy of the works in the area that can help practitioners and researchers to determine the method most suitable for their data and goal. In this taxonomy, we grouped the main methods of the algorithms presented into four main categories structural,

statistical, pattern-mining and hybrid, identifying subcategories whenever possible. Moreover, We provided comparative tables for each category of graph summarization approaches which can give the practitioners and researchers a synthetic and clear picture of existing works in this area. We then reviewed the existing works around quality metrics in graph summarization as well as a brief presentation of the related works on approximate frequent pattern mining since our proposal relies on such work to compute the summaries. We work around gathering and organizing the different efforts in a way that it is both usable but also conceptually clear. We tried to restrict ourselves to the most relevant works around RDF graph summarization and avoid to overextend the review to more generic graph-related efforts.

The work presented here has been published in two journal papers, [120] and [27].

Chapter 4

SemSum+: An algorithm for RDF graph summarization

4.1 Introduction

In this chapter we introduce the first contribution of this thesis, which is a novel solution into summarizing (semantic) RDF graphs, where our summary graph needs to:

- be an RDF graph itself so that we can use the same utilities and tools to process and do the same type of reasoning on the summary and the original graphs;
- contain statistical information, like the number of class and property instances per pattern, that can be exploited during query evaluation or other similar actions, that require knowledge of the statistical distribution of the contents of a specific RDF KB.

We worked having the federated query evaluation problem in mind but the algorithm can be used in different and diverse scenarios, since the produced summary is not application dependent and it fulfills different requirements. Our solution is based on mining top-k approximate graph patterns using an extended/adapted version of the PaNDa$^+$ [66] algorithm, which is according to the benchmarks [66] the best and most versatile available approximate pattern mining algorithm and which was adapted to be used in a Semantic Web setting. We named our algorithm **SemSum+** to signify the continuation of the PaNDa$^+$ work in the summarization of semantic graphs domain.

The proposed solution is responding to all the requirements by extracting the best approximate RDF graph patterns, construct a summary RDF schema out of them and thus concisely describe the RDF input data. The algorithm offers the following features:

- The summary is an RDF graph itself which allows us to post simplified queries towards the summarizations using the same techniques (e.g. SPARQL);
- Statistical information like the number of class and property instances per pattern is included in our summary graph, which allows us to estimate a query's expected results' size towards the original graph;
- The summary is much smaller than the original RDF graph, it contains all the *important* concepts and their relationships based on the number of instances;
- It summarizes the RDF input graphs regardless of having or not RDFS triples (schema independence);
- It summarizes the RDF graphs whether they are heterogeneous or homogeneous (heterogeneity independence).

4.2 PaNDa$^+$ Algorithm

Algorithm 1 PaNDa$^+$ Algorithm

K : max no. of patterns to be extracted
D : input binary matrix
D_R : residual binary matrix
J : cost function
C : core pattern to extend
E : items extension list
ϵ_r : max row noise threshold
ϵ_c: max column noise threshold
Π : set of patterns

function PaNDa$^+$(K, D, J, ϵ_r, ϵ_c)
    $\Pi \leftarrow \phi$
    $D_R \leftarrow D$
    for iter $\leftarrow 1, ..., K$ **do**
        C, E $\leftarrow$ Find-Core(D_R, Π, D, J)
        $C^+ \leftarrow$ Extend-Core$(C, E, \Pi, D, J, \epsilon_r, \epsilon_c)$
        if $J(\Pi, D) < J(\Pi \cup C^+, D)$ **then**
            break
        end if
        $\Pi \leftarrow \Pi \cup C^+$
        $D_R(i,j) \leftarrow 0\ \forall i,j$ where $C_T^+(i) = 1 \wedge C_I^+(j) = 1$
    end for
    return Π
end function

Firstly we introduce some notation which will be very useful in order to understand how the PaNDa$^+$ algorithm works. We start by the *binary* matrix $\mathcal{D} \in \{0,1\}^{N \times M}$ which denotes a *transactional dataset* of N transactions and M items, where $\mathcal{D}(i,j) = 1$ if the $j-$th item occurs in the $i-$th transaction, and $\mathcal{D}(i,j) = 0$ otherwise. $P =$

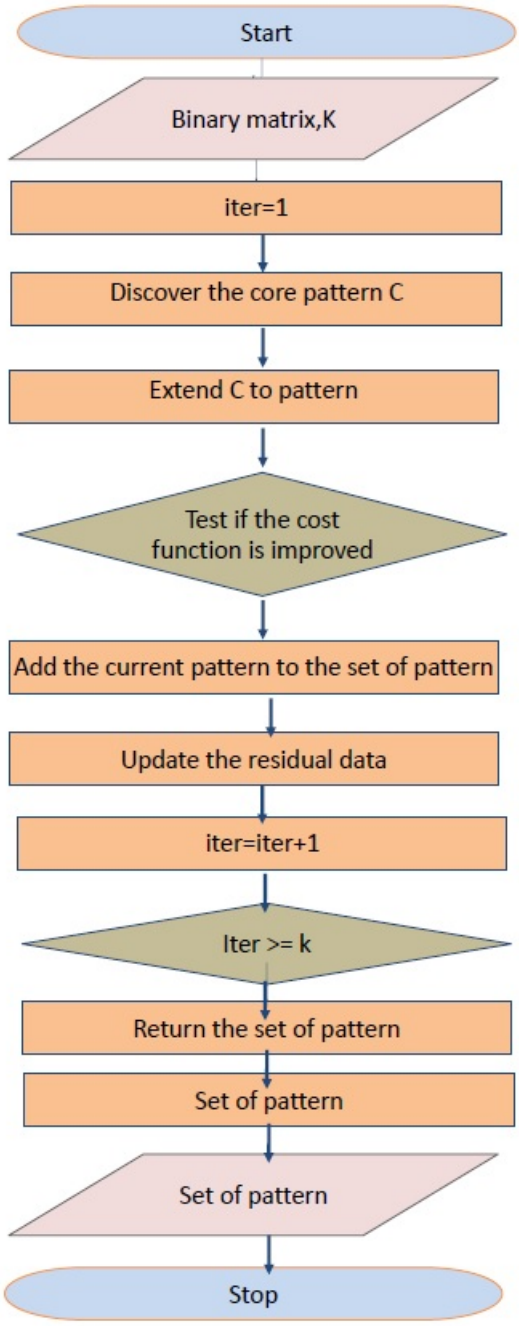


Figure 4.1 – Graphical representation of PANDA$^+$ algorithm

Algorithm 2 FIND-CORE Function

function FIND-CORE(D_R,Π,D,J)
  $E \leftarrow \phi$
  $S = \{s_1, ..., s_M\} \leftarrow$ SORT-ITEMS-IN-DB(D_R)
  C $\leftarrow \langle C_T = 0^N, C_I = 0^M \rangle$
  $C_I(s1) = 1$
  $C_T(i) = 1 \; \forall i$ where $D_R(i, s_1) = 1$
  for h $\leftarrow 2, ..., M$ **do**
    $C^* \leftarrow C$
    $C^*_I(s_h) = 1$
    $C^*_T(i) = 0 \; \forall i$ where $D_R(i, s_h) = 0$
    if $J(\Pi \cup C^*, D) < J(\Pi \cup C, D)$ **then**
      $C \leftarrow C^*$
    else
      E.append(s_h)
    end if
  end for
  return C
end function

Algorithm 3 EXTEND-CORE Function

function EXTEND-CORE(C, E, Π,D,J, ϵ_r, ϵ_c)
  for i $\in \{1, ..., N\}$ where $C_T(i) = 0$ **do**
    $C^* \leftarrow C$
    $C^*_T(i) = 1$
    if NOT TOO NOISY(C^*, ϵ_r, ϵ_c) **then**
      if $J(\Pi \cup C^*, D) < J(\Pi \cup C, D)$ **then**
        $C \leftarrow C^*$
      end if
    end if
  end for
  for each item e $\in E$ **do**
    $C^* \leftarrow C$
    $C^*_I(e) = 1$
    if NOT TOO NOISY(C^*, ϵ_r, ϵ_c) **then**
      if $J(\Pi \cup C^*, D) < J(\Pi \cup C^+, D)$ **then**
        $C \leftarrow C^*$
      end if
    end if
  end for
  return C,E
end function

$\langle P_I, P_T \rangle$ is an *approximate pattern* denoting two sets of items/transactions, where $P_I \in \{0,1\}^M$ and $P_T \in \{0,1\}^N$. The outer product $P_T \cdot P_I^\intercal \in \{0,1\}^{N \times M}$ identifies a sub-matrix of $\mathcal{D}$. An occurrence (i,j) is covered by the P *iff* $i \in P_T$ and $j \in P_I$.

The quality of a set of patterns $\Pi = \{P_1, \ldots, P_{|\Pi|}\}$ relies on how well they match the given dataset $\mathcal{D}$. We use a *noise matrix* $\mathcal{N} \in \{0,1\}^{N \times M}$ for accounting the mismatches between the set of patterns Π and the given dataset $\mathcal{D}$ i.e the occurrences $\mathcal{D}(i,j) = 1$ which are not covered by any pattern in Π (*false negatives*), as well as those $\mathcal{D}(i,j) = 0$ which are incorrectly covered by any of the patterns in Π (*false positives*). This *noise matrix* is defined as:

$$\mathcal{N} \;=\; \bigvee_{P \in \Pi} (P_T \cdot P_I^\intercal) \quad \veebar \; \mathcal{D}. \tag{4.1}$$

where $\vee$ and $\veebar$ are respectively the element-wise *logical or* and *xor* operators.

Approximate Top-k Pattern Discovery problem is defined as finding a small set of patterns Π that minimizes the noise matrix $\mathcal{N}$. More formally:

Problem 1 (Approximate Top-k Pattern Discovery) *Given a binary dataset* $\mathcal{D} \in \{0,1\}^{N \times M}$ *and an integer* k*, find the pattern set* $\overline{\Pi}_k$*,* $\left|\overline{\Pi}_k\right| \leq k$*, that minimizes a* cost function $J(\Pi_k, \mathcal{N})$*:*

$$\overline{\Pi}_k = \operatorname*{argmin}_{\Pi_k} J(\Pi_k, \mathcal{N}). \tag{4.2}$$

As we discussed in Section 3.3, PANDA$^+$ is considered the state of the art for the approximate pattern mining algorithms. PANDA$^+$ adopts a greedy strategy by exploiting a two-stage heuristic to iteratively select a new pattern: (a) discover a noise-less pattern that covers the yet uncovered 1-bits of $\mathcal{D}$, and (b) extend it to form a good approximate pattern, thus allowing some false positives to occur within the pattern.

4.2.1 Original version of PANDA$^+$

PANDA$^+$ greedily optimizes the following cost function:

$$J^+(\Pi_k, \mathcal{N}, \gamma_\mathcal{N}, \gamma_P, \rho) \;=\; \gamma_\mathcal{N}(\mathcal{N}) \;+\; \rho \cdot \sum_{P \in \Pi_k} \gamma_P(P) \tag{4.3}$$

where $\mathcal{N}$ is the noise matrix, $\gamma_\mathcal{N}$ and γ_P are user defined functions measuring the cost of the noise and patterns descriptions respectively, and $\rho \geq 0$ works as a regularization factor weighting the relative importance of the patterns cost.

Depending on the parameters of the J^+, PANDA$^+$ can greedily optimize several families of cost functions, including the ones proposed by other state-of-the-art algorithms [109, 68, 69, 65]. In this work, inspired by the MDL principle [87] we used $\gamma_{\mathcal{N}}(\mathcal{N}) = \mathsf{enc}(\mathcal{N})$, $\gamma_P(P) = \mathsf{enc}(P)$ and $\rho = 1$, where $\mathsf{enc}(\cdot)$ is the optimal encoding cost. It also allows to tune via the parameter ρ the relative importance of the patterns simplicity versus the amount of noise induced. These features make PANDA$^+$ a very flexible tool for approximate pattern mining extraction.

The pseudo code of Algorithm 1 gives an overview of PANDA$^+$. The Figure 4.1 shows also a graphical representation of it. Given a binary matrix D, an optional user parameter K determining the maximum number of extracted patterns and two maximum noise thresholds $\epsilon_r, \epsilon_c \in [0, 1]$ which bound the ratio of *false positive*, it extracts the top K patterns, based on the several families of cost functions of the J^+. We can see that it works iteratively and each iteration consists of two main functions: (a)FIND-CORE which extracts a noise-less pattern C (line 5) (b) EXTEND-CORE which extends the core pattern C to a new approximate pattern C^+ by allowing some false positives to occur within the pattern (line 6). The algorithm tests if the new approximate pattern C^+ reduces the current value of the J^+ cost function of the model, the algorithm adds it to the final results (lines 7 and 9). The algorithm normally stops producing further patterns when the cost function of a new patterns' set is larger than the corresponding noise reduction or the number extracted patterns is K or more. D_R is the residual parts of the D after removing the parts that are not yet covered by any previous pattern(line 11).

The Algorithm 2 shows the FIND-CORE function. Firstly, an extension list of items E is initialized as an empty set (line 2). The items are then sorted (line 3), where rather than considering all the possible exponential combinations of items, these are sorted in order to maximize the probability of generating large cores, and processed one at the time without backtracking. We mention two sorting strategies: (a) by frequency of an item in the full dataset, and (b) by the average frequency of every pair of items including the given item (named *charm* by [111]). A core pattern C is initialized with the first item in the generated ordered list lines(4-6). Then the algorithm treats the remaining items in the sorted list one by one, where for each item it creates a new candidate pattern C^* by adding this item to the current pattern C (lines 7-10). If the new candidate pattern C^* reduces the cost function of the pattern set, it will be promoted to be the new candidate and it is used in the subsequent iteration(lines 11 and 12). Otherwise, the item is appended to the extension list E and it can be used later to extend the pattern (lines 13 and 14).

The function which extends the extracted core patterns is shown in Algorithm 3. Given a core pattern C and a list of items E, the function iteratively tries to add transactions and items to C by allowing some false positives, as long as the cost function J^+ is reduced. It starts by extending the C with additional transactions. Where for each transaction of the dataset that is not yet included in C (line 2), it creates a new candidate pattern C^* by adding this transaction to the list of transactions of C (lines 3 and 4). Note that the addition of a transaction t to a pattern C assumes that t has all the items of C in original dataset (which is not always correct), thus this addition introduces some false positives (noise). Then if the introduced false positives respect both thresholds of the error parameters $\epsilon_r and \epsilon_c$ (line 5), and the C^* improves the cost function(line 6), then the C will be replaced by C^* (line 7). After treating all the transactions, the algorithm treats the items in extension list E one by one. For each item in E it creates a new candidate pattern C^* by adding this item to the current pattern C lines(12 and 13). Note and In contrast to the FIND-CORE function, when an item e is added to pattern C, the corresponding transaction set is not modified meaning that e is approximately supported by all the transactions of C. If new candidate pattern C^* does not introduce too many false positives (line 14) and improves the overall cost function (line 15), then the C will be replaced by C^* (line 16). This step cycle stops when E becomes empty.

4.2.2 The SemSum+ Version

After studying of the PANDA$^+$ algorithm, we have found some efficiency limitations. We have noticed that the second part of the algorithm (Extend Core) is time-consuming and memory-intensive. We have seen in the algorithm that for the extending transactions step, the algorithm must treat all the transactions of the original dataset that are not yet included in the actual pattern. Even if a lot of those transactions do not have any item included in the core pattern items. So we can imagine if we have a large matrix of transactions, the algorithm will have to go through all these rows in the original data, this is a huge waste of time and memory. Thus, in order to improve the efficiency of the PANDA$^+$ algorithm We proposed a method in the FIND-CORE function for determining the extended transactions list that will additionally be used in the FIND-CORE function. Thus the output of the FIND-CORE function will be: the extension list of items (E_I), the extension list of transactions (E_T) and the pattern C. The modification that we proposed does not influence the final result; it only avoids re-testing the transactions which do not have any item of the items of the corresponding pattern and thus it is certain that they cannot improve

Data structure	BitSet	Boolean
Instruction	BitSet b = new BitSet (1000);	Boolean tab [] = new Boolean [1000];
The memory usage	The implementation of BitSet uses an array of Long (1000/64 + overhead) ⇒ The memory usage = (15,63*8) + 16= 141 bytes.	Java uses a byte for each element ⇒ The memory usage= (The size of the array * the size of type) + the overhead= 1000 * 1 + 16 = 1016 bytes.

Table 4.1 – Comparison between data structures

the specific pattern. The Algorithms 4, 5, 6 show our optimized PANDA$^+$, FIND-CORE1 function and FIND-CORE1 function respectively, where our modifications are colored in gray.

4.2.3 Implementation Details and Experiments

In order to compare the efficiency of two versions of PANDA$^+$ covered in previous section, we implemented them in the Java language. Concerning the data structure that is used to represent the binary matrix in java. A study was carried out around the data structures which allow the manipulation of binary matrices, the sorting and the logical operations. We have made comparisons between several data structures, to find the right structures that make it possible to gain memory and to give better execution time in the presence of a large volume of data. We chose to code with the Java class BitSet, it implements a vector of bits that grows as needed. Each component of the BitSet has a Boolean value, by default, all bits in the set initially have the value false, the BitSet class use a single bit to represent a true/false Boolean value. This means that when we use the BitSet data structure we can save a lot of memory and perform faster bit-level operations. Table 4.1 shows the size of memory needed to represent a binary matrix of 1000 elements by the both data structures.

The two algorithms are evaluated over a set of datasets which will be described in section 7.1. The experiments ran on an Intel(R) Core (TM)i2 CPU6400 2.13 GHZ*2 with 3,8 GB of RAM, running Ubuntu 17.04. Table 4.3 shows the result, the first column shows the dataset name, the second and the third columns show the number of transactions and items respectively, the fourth column shows the execution time before the optimization of PANDA$^+$ and finally the last column shows the execution time after optimization. For example the execution time of Jpeel dataset before optimization is equal to 26 seconds, while it is 18 seconds after optimization (an improvement of over 30%), the execution time of Wordnet dataset is about 764 seconds before optimization, while it becomes 503 seconds after optimization (an improvement of about 35%).

Algorithm 4 Our Modified PANDA$^+$ Algorithm

function PANDA$^+$(K, D, J, ϵ_r, ϵ_c)
    $\Pi \leftarrow \phi$
    $D_R \leftarrow D$
    for iter $\leftarrow 1, ..., K$ **do**
        C, $E_I, E_T \leftarrow$ FIND-CORE1(D_R, Π, D, J)
        $C^+ \leftarrow$ EXTEND-CORE1($C, E_I, E_T, \Pi, D, J, \epsilon_r, \epsilon_c$)
        if $J(\Pi, D) < J(\Pi \cup C^+, D)$ **then**
            break
        end if
        $\Pi \leftarrow \Pi \cup C^+$
        $D_R(i, j) \leftarrow 0\ \forall i, j$ where $C_T^+(i) = 1 \wedge C_I^+(j) = 1$
    end for
    return Π
end function

Algorithm 5 FIND-CORE1 Function

function FIND-CORE1(D_R,Π,D,J)
    $E \leftarrow \phi$
    $S = \{s_1, ..., s_M\} \leftarrow$ SORT-ITEMS-IN-DB(D_R)
    C $\leftarrow \langle C_T = 0^N, C_I = 0^M \rangle$
    $C_I(s1) = 1$
    $C_T(i) = 1\ \forall i$ where $D_R(i, s_1) = 1$
    for h $\leftarrow 2, ..., M$ **do**
        $C^* \leftarrow C$
        $C_I^*(s_h) = 1$
        $C_T^*(i) = 0\ \forall i$ where $D_R(i, s_h) = 0$
        if $J(\Pi \cup C^*, D) < J(\Pi \cup C, D)$ **then**
            $C \leftarrow C^*$
        else
            E.append(s_h)
        end if
    end for
    return C
end function

Dataset	Items	transactions	Before optimization	After optimization
Jpeel	76,229	49	26	18
Jamendo	335,925	50	88,668	62,609
Sec	460,446	20	66,824	46,133
linkedMDB	694,400	428	1	1
Bank	200,429	34	6,465	4,741
Wordnet	647,215	123	764,47	503,27
DBLP	5,942,858	38	-	-
Linkedct	5,364,776	195	-	-

Table 4.2 – Execution time in seconds

Algorithm 6 EXTEND-CORE1 Function

function EXTEND-CORE1(C, E, Π,D,J, ϵ_r, ϵ_c)
  for each transaction t ∈ E_T **do**
    $C^* \leftarrow C$
    $C^*_T(i) = 1$
    if NOT TOO NOISY(C^*, ϵ_r, ϵ_c) **then**
      if $J(\Pi \cup C^*, D) < J(\Pi \cup C, D)$ **then**
        $C \leftarrow C^*$
      end if
    end if
  end for
  for each item e ∈ E_I **do**
    $C^* \leftarrow C$
    $C^*_I(e) = 1$
    if NOT TOO NOISY(C^*, ϵ_r, ϵ_c) **then**
      if $J(\Pi \cup C^*, D) < J(\Pi \cup C^+, D)$ **then**
        $C \leftarrow C^*$
      end if
    end if
  end for
  return C,E
end function

4.3 Computing RDF graph summaries

We present in this section our approach of RDF graph summarization, which is based on extracting the smallest set of approximate graph patterns that best describe the input dataset, where the quality of the description is measured by an information theoretic cost function. We use our modified version of the PANDA$^+$ algorithm presented in section 4.2.2, which uses a greedy strategy to identify the smallest set of patterns that best optimize the given cost function to the solution. As we mentioned above PANDA$^+$ algorithm normally stops producing further patterns when the cost function with a new pattern set, provides higher cost values than the corresponding noise reduction. It also allows the users to fix a value k to control the number of extracted patterns. One of the challenges that we faced is how we map the RDF KB

to a binary matrix while preserving the semantics of this KB and in addition producing always a valid RDF graph as a result. Our approach works in three independent but interwind steps that are described below and in Fig.4.2.

4.3.1 Binary Matrix Mapper

We transform the RDF graph into a binary matrix D, where the rows represent the subjects (and objects when applicable) and the columns represent the predicates. We preserve the semantics of the information by capturing distinct types (if present), all attributes and properties. In order to capture both subject and object of a property, we create two columns for each property, the first one captures the subjects participant as domains of the property where the second one (we call it *reverse property*) captures the objects participant as ranges of the property, eg. for the property paints we create two columns (paints, R_paints), see Table 4.3, where the column paints captures its subjects $\{Picasso, Rembrant\}$ while the column R_paints captures its objects $\{Woman, Guernica, Abrahama\}$. We extend the RDF URI information by adding a label to represent the different predicates carrying this information into the patterns. This label is of the following form: Usage prefix and the RDF URI element label where these two parts are concatenated with a forward slash ("/"), where the usage prefix is T for type, P for property and R for reverse properties. We do this indiscriminately for schema and instance related triples. This matrix is defined by the following form:

$$D(i;j) = \begin{cases} 1, & \text{the i-th URI has j-typeof or is j-property's} \\ & \text{domain/range or is j-attribute's domain} \\ 0, & \text{otherwise} \end{cases}$$

The Algorithm 7 shows the pseudo code of creating a binary matrix for a RDF KB. The function CreatMappingListOfSubjects(line 3) creates the mapping list for subjects where for each subject in the KB it generates a distinct number corresponding to the row number in the binary matrix that will be generated. The function createMappingListOfProperties(line 3) creates the mapping list for the predicates(properties) where for each predicate in the KB it generates a distinct number corresponding to the column number in the binary matrix that will be generated.

Example 5 *Table 4.3 shows the mapped binary matrix D for the RDF graph depicted in Figure.2.2. This matrix consists of 9 columns and 6 rows, where the columns represent 2 distinct attributes (fname,lname), 2 distinct properties (paints, exhibited), 2 distinct reverse proprieties (Reverse_paints, Reverse_exhibted), 3 distinct types*

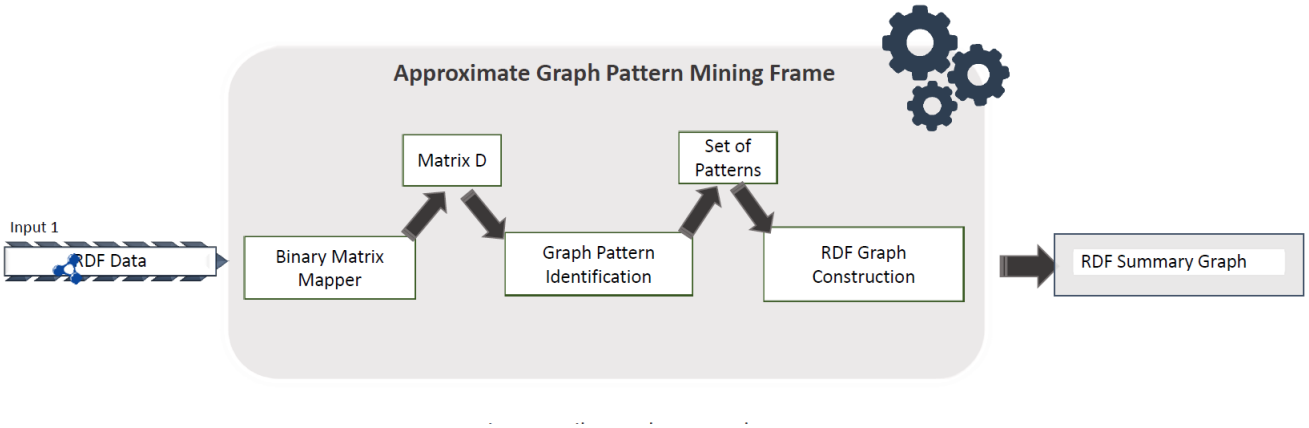


Figure 4.2 – Our RDF graph summarization approach

Algorithm 7 Binary Matrix Mapper Algorithm

D : the output binary matrix
KB: input RDF knowledge base
S: Mapping list of subjects
P: Mapping list of predicates
Tr: list of triples

1: **function** CREATBINARYMATRIX(KB)
2: $S \leftarrow createMappingListOfSubjects(KB)$
3: $P \leftarrow createMappingListOfProperties(KB)$
4: $Tr \leftarrow getAllTriples(D)$
5: **for each** triple $t \in Tr$ **do**
6: $sid \leftarrow S.get(t.subject)$ ▷ sid: corresponding row number of the current triple subject
7: $pid \leftarrow S.get(t.predicate)$ ▷ pid: corresponding column number of the current triple predicate
8: $D[sid, pid] \leftarrow 1$
9: $rpid \leftarrow S.get(R : +t.predicate)$
10: **if** (S.get(t.object) is not Null) **then**
11: $oid \leftarrow S.get(t.object)$
12: $rpid \leftarrow S.get(R : +t.predicate)$ ▷ rpid: corresponding column number of the reverse current triple predicate
13: $D[oid, rpid] \leftarrow 1$
14: **end if**
15: **end for**
16: return D
17: **end function**

(Painter(c), Painting(c),Museum(c)). In order to distinguish between the types/classes and the properties/attributes at the visualization level, we use Y(c) to denote that Y is type/class. The rows represent the 6 distinct subjects (Picasso, RembrantvanRijn, Woman, Guernica, Abraham,museum.es), e.g. D(1,1)=D(1,3)=D(1,4)=D(1,5)=1 because Picasso, who is described in the first row, is an instance of Painting class and has (lname, fname) attributes and paints properties respectively, while D(1,6)=0 because Picasso does not have the exhibited property.

	Painter(c)	Painting(c)	lname	fname	paints	exhibited	R_paints	R_exhibited	Museum(c)
Picasso	1	0	1	1	1	0	0	0	0
Rembrant	1	0	1	1	1	0	0	0	0
Woman	0	1	0	0	0	0	1	0	0
Guernica	0	1	0	0	0	1	1	0	0
Abraham	0	1	0	0	0	1	1	0	0
museum.es	0	0	0	0	0	0	0	1	1

Table 4.3 – The mapped binary matrix D for the RDF instance graph depicted in Figure 2.2

As we will discuss later on and as our experiments will show, the algorithm works adequately (even equally) well even in the absence of any schema information, or in other words, no schema information is required for the algorithm to work adequately well.

4.3.2 Computing Graph Patterns

We aim at creating a summary of the input RDF graph by finding patterns in the binary matrix produced in the previous step (see Table 4.3). By pattern, we mean properties (columns) that occur as a whole or partly (and thus approximately) in several subjects (rows). This problem is known in the data mining community as *approximate pattern mining*. This is an alternative approach to pattern enumeration. It aims at discovering the set of k patterns that best *describe*, or *model*, the input data. Algorithms differ in the formalization of the concept of *dataset description*. The quality of a description is measured with some cost function, and the top-k mining task is casted into an optimization of such cost. In most of such formulations, the problem is demonstrated to be NP-hard, and therefore greedy strategies are adopted. The Algorithm 8 shows the pseudo code of this step where we apply our modified PaNDa$^+$ algorithm with the xor cost function and the charm sorting method parameters. Our selection of these two parameters was done after a very various set of experiments over set real-world datasets from diverse domains.

Algorithm 8 Computing Graph Patterns Algorithm

K : max no. of patterns to be extracted
D : input binary matrix
Π : set of patterns
J: type of the cost function
S: sorting method

1: **function** ComputingGraphPatterns(D)
2: $S \leftarrow charm$
3: $j \leftarrow xorcost$
4: $\Pi \leftarrow PaNDa + (K, D, J, S)$
5: return Π
6: **end function**

Example 6 *Table 4.4 shows possible patterns which can be extracted from the mapped binary matrix depicted in Table 4.3. The first column represents the pattern id. The second column represents the predicates/properties included in a pattern and the third column represents the number of subjects/instances per pattern, e.g., the pattern P1 denotes that there are three subjects belong to the Painting class and have* $\{exhibited\}$ *an outgoing attribute and*$\{paints\}$ *an incoming attribute.*

ID	Pattern	Instances
P1	Painting(c),exhibited, revers_paint	3
P2	Painter(c),paints, fname, lname	2
P3	Museum(c)	1

Table 4.4 – Extracted patterns example

As already pointed out, in this work, we adopted the state-of-the-art PaNDa$^+$ algorithm [66] to extract top-k approximate patterns from the binary dataset resulting from the transformation of the original RDF graph.

4.3.3 Constructing the RDF summary graph

We have implemented a process, which reconstructs the summary as a valid RDF graph using the extracted patterns. For each pattern, we start by generating a node labeled by a URI (minted from a hash function), then we add an attribute with the *bc:extent* label representing the number of instances for this pattern. Then and for each item involved in this pattern, we use the labels generated in 4.3.1 to understand its type where if it is:

- Property: We generate a direct edge from the node representing the pattern containing this property to the node representing the pattern containing the reverse property.

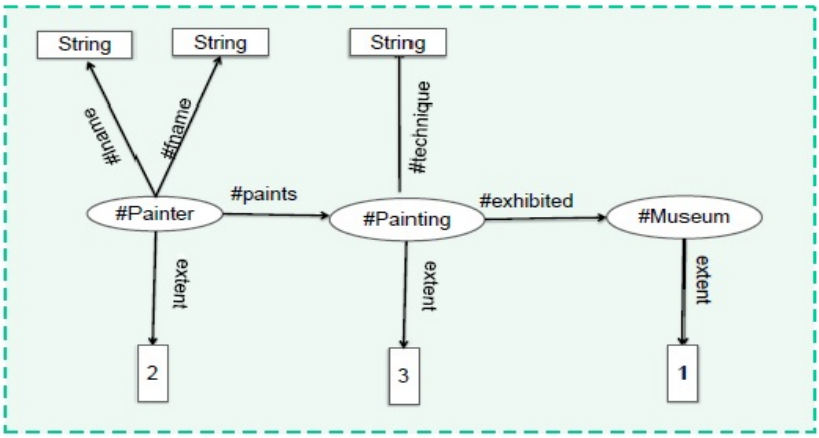


Figure 4.3 – RDF Summary graph for the set of patterns depicted in Table 4.4

- Attribute: We generate a direct edge to a new generated node labeled by a URI (e.g. from a hash function).
- Type: We generate a direct edge labeled with *RDF:type* label to a new generated node labeled with the RDFS label of this type.

The process exploits information already embedded in the binary matrix (e.g. property X range links) and tries to construct a valid RDF schema to represent the KB. This schema is enriched with statistical information since the algorithm returns for each pattern the number of instances it corresponds to.

Example 7 *Figure 4.3 shows the constructed RDF summary graph for the set of patterns depicted in Table 4.4.*

The summary construction process will capture links that are part both of the schema and the instances, given their statistical significance. This means that we treat the same way schema defined or instance created relationships (properties and attributes). This means that we will not necessarily capture subsumption (hierarchical, *subClassOf* or *subPropertyOf*) relationships. This is one of the reasons that our approach works equally well (as we demonstrate in Chapter 7) with or without schema information but on the other hand, we might miss some subsumption relationship of importance, especially in the absence of schema information. As we discuss in Chapter 8, this is part of our future work.

4.4 Summary

In this work we apply a top-k approximate graph pattern mining algorithm in order to extract a summary of an RDF KB. The summary is not necessarily the complete schema of the KB but it is the used/active schema of the KB, usually a subset of the original full schema, and always remains a valid RDF/S graph. Comparing it with the ideal RDF summary either provided by an expert or was used while creating the KB, shows us that the summary presented by our system is very close to it, which means that the algorithm performs exceptionally well without relying on the existing schema information (at least for the diverse set of experiments that are presented in Chapter 7).

This part of the work was consolidated in two papers and a conference presentation, namely one at EDBT 2016 [118], one paper in the Springer CCIS volume of the ISIP 2016 post-proceedings [117] and a presentation in the same workshop.

Chapter 5

Parallel RDF graph summarization

5.1 Introduction

As we mentioned in the previous chapter, our RDF graph summarization is based on extracting the smallest set of approximate graph patterns that best describe the input dataset. We use the SemSum+ algorithm presented in section 4.2.2 which uses a greedy strategy to identify the smallest set of patterns that best optimize the cost function and lead to a minimum cost solution. On the algorithmic side, the original PaNDa$^+$ algorithm (a modified version is part of our SemSum+ algorithm) is bound to datasets that fit in main memory. The algorithm's memory requirements grow linearly with the size of the input dataset, which is problematic for quite large datasets. This limits the size of datasets for which our summaries could be generated.

Few efforts have been reported in the literature and deal with the memory-constrained problem of the approximate pattern mining algorithms [86, 114, 71]. PARMA [86], is a parallel algorithm implemented in the Hadoop/MapReduce Framework. At first, the input dataset is split, using a random sampling approach, into a set of random samples. Then each Mapper applies the FP-growth pattern mining algorithm on one of these generated samples of the dataset. The reducers will then filter and aggregate the results of the mappers in order to produce the output collection. The two other algorithms [114, 71] are also implemented using the Hadoop/MapReduce Framework they are like the PARMA algorithm: they split the input dataset into a subset of datasets and then they apply one of Frequent Itemset Mining (FP-Growth [44], Apriori [9]) algorithms on them. Subsequently, a parallel merging step is taking place for getting the final results. The three algorithms can only work with the existing "exact" pattern mining algorithms and not with the actual approximate pattern mining algorithms. Their approximation comes from the

fact that splitting the data, treating the split data and then merging them will provide some level of approximate results. On the contrary, in this work, our goal is to parallelize an actual approximate pattern mining algorithm (while our proposal can work with any approximate pattern mining algorithm based on a binary matrix). This provides all the benefits of doing approximate calculations for the patterns and not approximating by necessity and only while merging the results.

In this chapter, we propose a parallel algorithm for the PANDA$^+$ algorithm. We have implemented our algorithm using MapReduce. Our parallel algorithm works in two phases, in the first phase we divide horizontally (by row) the input Binary Matrix into several smaller binary submatrixes, each Mapper runs our modified PANDA$^+$ algorithm on one of these Matrixes. In the second phase, we proposed a parallel merging algorithm that tries to merge all the possible pairs (p_1, p_2) of patterns from the set of patterns extracted in the previous phase. The merging still follows the principle of minimizing the overall cost function. The proposed parallelization is not limited to the current algorithm but it can be applied for all the approximate pattern mining algorithms which are based on a binary matrix; of course the fact that we are discussing approximate patterns helps since small differences in the merging phase might still give us the same approximations. With the part of the pattern computation parallelized, we can then integrate this to our SemSum+ algorithm and allow it to compute summaries for very large graphs, given that what we propose is highly scalable.

The chapter is structured as follows. Section 5.2 presents an overview of the short description of the Hadoop / MapReduce framework; section 5.3 describes our parallel PANDA$^+$ implemented on Hadoop/MapReduce (we call this the PANDA^{++} parallel algorithm). We then conclude our chapter in section 5.4.

5.2 Parallelization (Hadoop/MapReduce)

Hadoop [107] is an open-source programming framework that is capable of running applications for large-scale processing and storage on large clusters of commodity hardware. Hadoop cluster characteristics include the partitioning or distributing of data, computation across multiple nodes, and performing computations in parallel. Hadoop splits the files into large blocks and distributes them across the cluster nodes. To process the data, Hadoop transfers the code to each node and each node processes the data it has. This makes it possible to process all the data more quickly and more

efficiently than in a more conventional super calculator architecture, where data are exchanged among the nodes in order to perform the calculation.

The Hadoop framework consists of four core components:

- Hadoop common: it is also known as Hadoop Core and it provides the common utilities and libraries that are required by other Hadoop components.
- Hadoop distributed file system (HDFS): from its name, it is a distributed file system that stores and retrieves files in record time. This is one of the basic components of the Hadoop Apache framework, and more specifically its storage system.
- Hadoop YARN: is a resource-management framework for handling compute resources and job scheduling of user applications.
- Hadoop-MapReduce: is a programming model for parallel processing of large-scale datasets and it is an integral part of Hadoop.

All these components are specially designed to be highly fault-tolerant.The two important components in the Hadoop-MapReduce framework are the storage part (HDFS) and the processing part (MapReduce).

5.2.1 HDFS

HDFS is a distributed file system that provides high-performance access to data distributed in Hadoop clusters. Like any other filesystem we can create files, organize them into directories, list the contents of these directories, add permissions, etc. In short, everything we can expect from a file system. However, the HDFS is fundamentally different because of its distributed nature. HDFS provides a block replication system with a configurable number of replications. During the writing phase, each block corresponding to the file is replicated to multiple nodes. For the reading phase, if a block is unavailable on a node, copies of this block will be available on other nodes. So, data loss in HDFS is very rare even in the case of hardware failure.

An HDFS cluster relies on two major types of nodes: a NameNode and a number of DataNodes. The NameNode manages the file system namespace, it maintains the file system tree and the metadata for all the files and directories in the tree. This information is stored persistently on the local disk. The NameNode also knows the DataNodes on which all the blocks for a given file are located. The DataNodes are the workhorses of the file system, they store and retrieve blocks when they are told to (by

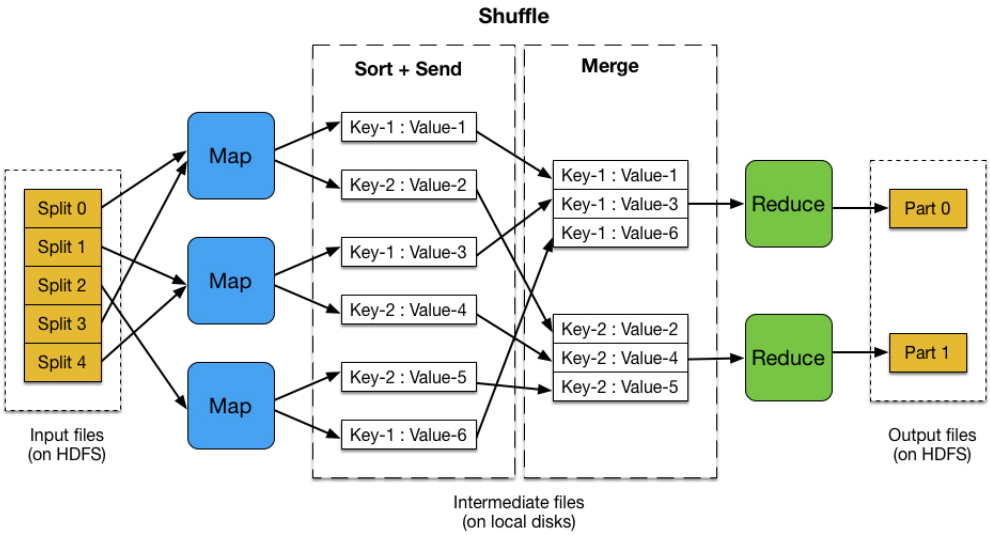


Figure 5.1 – MapReduce Model.

clients or the NameNode), and they report back to the NameNode periodically with lists of blocks that they are storing, without the NameNode, the file system cannot be used.

5.2.2 MapReduce

MapReduce [34] is a programming model for processing and generating large sets of data stored on a (Hadoop) cluster. It is a core component of the Apache Hadoop, which enables the resilient and distributed processing of massive data on computer clusters. The Map function transforms input data into (key,value) pairs, processes them, and generates another set of intermediate (key, value) pairs at the output. The Reduce function also transforms the input (the output of the Map function) into (key,value) pairs and generates one new set of the (key,value) pairs at the output. The terms Mapper and Reducer refer to the Hadoop servers that execute the Map and Reduce functions respectively. Figure 5.1 shows how the MapReduce framework works. The input data is divided into smaller blocks. Each block is then assigned to a Mapper for the processing. Each Mapper then will generate an intermediate set of (key, value) pairs. When all the Mappers finish the processing, the framework shuffles and sorts the results before passing them to the Reducers, where the pairs which have the same key will be assigned to the same Reducer. The Reducers can not start until all the Mappers finish the processing.

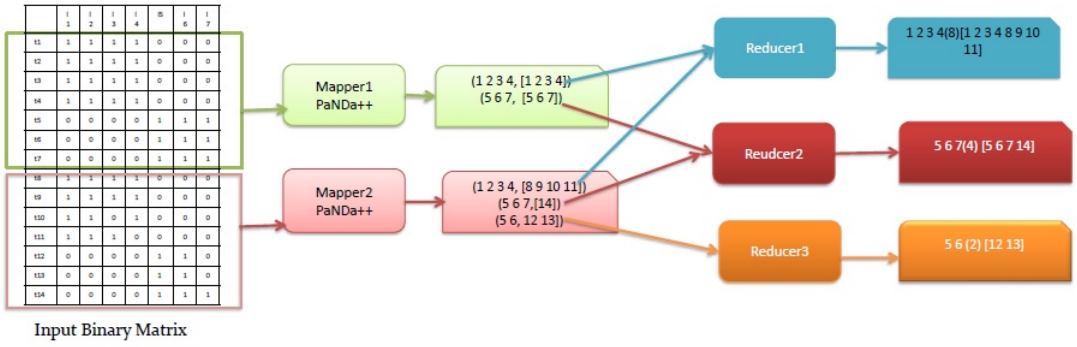


Figure 5.2 – Our Parallel algorithm: first phase.

5.3 Parallel Algorithm

In this section, we present a parallel approximate frequent pattern mining algorithm, which parallelizes our improved version of the state-of-the-art approximate frequent pattern mining algorithm PANDA$^+$ (described in Chapter 4). It is proposed to mine approximate patterns from a dataset, based on HDFS and MapReduce. The algorithm consists of two phases of MapReduce jobs.

5.3.1 Phase 1: computing the patterns

In this phase, we divide horizontally (by row) the input binary matrix into several smaller binary submatrixes, where the division is not only based on the number of rows but also based on the frequencies of bits equal to one. The division based on the frequencies ensures us avoiding cases like mapper finishes the task assigned to it very fast while another mapper needs substantially more time to finish the same task. This means that each mapper might receive different number of rows. Then each mapper applies the PANDA$^+$ algorithm on one of the generated submatrixes, which returns a list of patterns. The mapper then uses the returned list of patterns for generating a set of intermediate (key, value) pairs, where the key is a sorted list of the pattern items and the value is a list of the pattern transactions. When all the mappers finish, the reducers merge the patterns having the same list of items and replaces them by a new pattern where its items are the list of their items and its transactions are the union of their transactions. This merging operation is directly done without verifying again the overall cost after the merging, because it is evident that merging two patterns that share the exact same list of items will not add any

noise (false positive). Figure 5.2 shows the flow chart of our algorithm for this phase with an example. The pseudo code of this phase is also shown in Algorithm 9.

Algorithm 9 PANDA^{++} Phase 1 Algorithm: Computing the patterns

K : max no. of patterns to be extracted
D : input binary matrix
Π : set of patterns
Trs: list of transactions ids

```
1: function MAPPER(sub-matrix t)
2:     Π ← SemSum + (t)
3:     for each Pattern P ∈ Π do
4:         emit(P.itemsIds, P.transactionsIds)
5:     end for
6: end function
7: function REDUCER(key=itemsIds iids, value=transactionsIds tids[])
8:     Trs ← φ
9:     for each tids_i ∈ tids do
10:        add tids_i to Trs
11:    end for
12:    sort(iids)
13:    emit(iids, Trs)
14: end function
```

5.3.2 Phase 2: merging the patterns until reaching k

Phase 2 implements the merging step. We assume that in the previous step, a set of patterns Pi extracted independently and in parallel on each partition. This phase consists of rounds of MapReduce jobs where for each round it merges all the possible pairs of patterns, finds the best pair (p_1, p_2) that can be merged with the smallest cost. If merging (p_1, p_2) decreases the overall cost, remove (p_1, p_2) from Π and replace them with the new merged pattern. This is repeated till arriving to the computations indicating that the merging (p_1, p_2) does not decrease the overall cost.

The main problem is to compute the cost of merging two patterns without accessing the whole dataset again. To compute the cost after merging, we need data (columns and rows) from the neighborhood, denoted by the rectangles "Missing P1" and "Missing P2" in Figure 5.3. The difficult part is to consolidate the number of 0s (what we could call *holes*) in "Missing P1" plus the number of 0s (again *holes*) in "Missing P2". To solve it in a very generic way, during the extraction of a pattern p_1, we store for every other (not included in the pattern) item the number of its occurrences (i.e. the number of 1s) in the same transactions (rows) of p_1. This means that for every column of the dataset, we store the number of 1s present in the rows

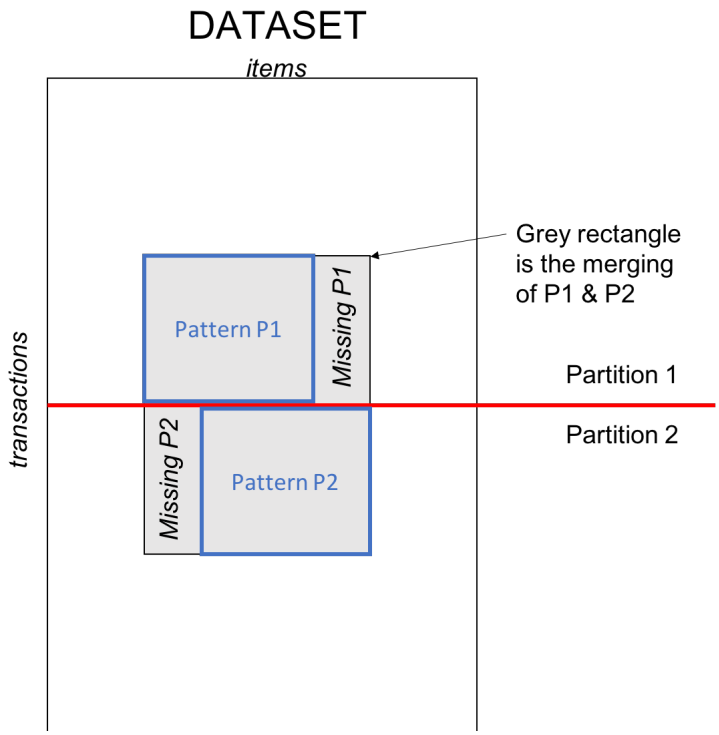


Figure 5.3 – Merging two patterns example.

corresponding to pattern p_1. We call this stats "Missing-Neighbors". To compute the number of *holes* in "Missing P1" it is sufficient to use the above statistic for the corresponding items of "Missing P1". The pseudo code of this phase is shown in Algorithm10 and it is also described in the following paragraph.

5.3.2.1 Explaining the pseudocode for the merging phase

Suppose that we have a cluster consisting of N machines, one of which is master H, the other are data nodes Di (i=1...N-1). The implementation can be outlined as follows:

1. Use the union of the set of patterns extracted in the phase 1 to build the global mapping structure of patterns in the master host.

2. The Master H distributes the tasks to each Di with the Pattern ID: pattern 1 assigned to D1; pattern 2 assigned to D2, Pattern 3 assigned to D3, etc.;

3. Each data Node Di will calculate the cost functions of merging (Pi,Pj) where j=1, 2, ...,m and saves them in a temporary file.

Algorithm 10 PANDA^{++} Phase 2: Merging patterns

K : max no. of patterns to be extracted
D: set of nodes
Π : set of patterns
$C[i, j]$: store the cost benefit in merging P_i with P_j.

```
1: t ← true
2: while t=true do
3:     apply the Mapper function
4:     t ← Reducer()
5: end while
6: function MAPPER(Taskid id, Π)
7:     for each Pattern P ∈ Π do
8:         if (P.id != id) then
9:             C[id,p.id]← CostOfmerging(P_id, P)
10:        end if
11:    end for
12:    for h ← 1, ....., Π.size do
13:        Temp[h]← C[id, h]
14:    end for
15:    emit(1, Temp)
16: end function
17: function REDUCER(key=1, value=Costs C[ , ])
18:    i, j ← GetIndexesOFMinValue(C)
19:    c ← CostOfmerging(P_i, P_j)
20:    if c < 0 then
21:        Pattern P← Merge(P_i, P_j)
22:        add P to Π
23:        return true
24:    else
25:        return false
26:    end if
27: end function
```

4. Each node sends its' result list to the Master, which processes merging operation with the help of a Reducer thread and gets the overall sorted list;

5. Check the result of this iteration: we select the pair (p1,p2) in this iteration with the smallest cost. If the value is bigger than 0, go to step 7 (this means that nothing improves the current situation), else go to step 6 (we merge and repeat);

6. Merge the two patterns (p1,p2) and update the global mapping structure and go to step 2.

7. Stop process

5.3.3 Implementation Details and Experiments

Experiments are implemented on a heterogeneous Hadoop/MapReduce cluster consisting of 3 nodes, a master and 2 slaves with the following characteristics:

- the *master node*: Intel(R) Core TM (i3) CPU 6300, 1.68 GHZ*2 with 4 GB of RAM
- the first *slave node*: Intel(R) Core TM (i2) CPU 6400 GHZ*2 with 2 GB of RAM
- the second *slave node*: Intel(R) Core TM (i2) CPU 6400 2.13 GHZ*2 with 2 GB of RAM

5.3.3.1 Efficiency test

Dataset	Transactions	Items	Sequential version (time in sec)	Parallel version (time in sec)	Speed-up ratio
Jpeel	76,229	49	16	12	1.33
Jamendo	335,925	50	71	36.13	1.96
Sec	460,446	20	46	26	1.76
Bank	200,429	34	6,465	4.741	1.36
Wordnet	647,215	123	947	403.27	2.34

Table 5.1 – Result of efficiency test

The two algorithms (the sequential and the parallel) are evaluated over a subset of the datasets, which we describe in detail in section 7.1. The sequential algorithm was evaluating running on only the master node. Table 5.1 shows the result, the first column shows the dataset name, the second and the third columns show the number of transactions and items respectively, the fourth column shows the execution time for the sequential algorithm, the fifth column shows the execution time for the parallel

version and finally the last column shows the speed-up ratio of our parallel framework. The speed-up ratio of a parallel framework consists of N computers is defined as the execution time of running the sequential algorithm in one computer divided by the time consumed in this framework on the same test dataset. The speed-up ratio of our cluster that consists only of three nodes (computers) is more than 2,3 for the wordnet dataset (the biggest dataset in these experiments) and the efficiency is improved by 57.5% for this dataset. In summary, we can say the speed-up ratio of our framework, which consists of 3 computers, is up to 1,7 for most of used datasets.

At the effectiveness level, our experiments so far provide the indication that the algorithm works very well at the level accuracy and recall rate of where their values are approximately equal to 1 for the five tested datasets.

5.4 Summary

In this chapter, we presented a novel parallel algorithm for the PANDA^{++} algorithm, which is one part of our RDF graph summarization approach. This novel parallel algorithm lets the computations of summaries to scale up and thus be able to summarize larger RDF graphs. This algorithm is not limited to the PANDA^{+} algorithm but it can be used for any approximate pattern mining algorithm, which is using a binary matrix and relies on a cost function to compute the level of the desired approximation. The implementation was done using the Hadoop/MapReduce Framework, which allows us to use it under any cloud infrastructe available. A paper on this subject is in the works, but at the time of writing of this thesis it has not yet being submitted.

Chapter 6

Quality Metrics For RDF Graph Summarization

In this chapter we address the problem of the quality of the different RDF summaries. The question that we will try to answer is not necessarily what is the best summary but, better, how a summary compares to another and what are their differences. So when different algorithms compute the summary of the same KB, we need to have an established methodology to compare the produced summaries and decide on their quality and best-fitness for specific tasks. So, we provide a comprehensive Quality Framework for RDF Graph Summarization that allows a better, deeper and more complete understanding of the quality of the different summaries and facilitates their comparison.

The chapter is structured as follows: Section 6.1 presents an overview of our contribution in the chapter; Section 6.2 presents our proposed Quality Metrics for RDF Graph Summaries while Section 6.3 presents our implementation of this framework. Section 6.4 provides a working example in order to show the appropriateness of the proposed metrics; full experimental results that demonstrate the versatility of the proposed framework are presented in Chapter 7. We then conclude this chapter in section 6.5.

6.1 Overview

The main difficulty towards understanding the quality of the different generated RDF summaries is a lack of widely accepted evaluation criteria or an extensive empirical evaluation. This leads to the necessity of a method to compare and evaluate the quality of the produced summaries. This method would allow a better understanding of the quality of the different summaries and facilitate their comparison and decide

on their quality and best-fitness for specific tasks. Despite the existence of a good number of RDF summarization approaches, there is a very little effort in the literature into addressing in a comprehensive and coherent way the problem of evaluating these summaries against different criteria and have some mathematical metrics to describe the quality of the results. As we have mentioned in Section 3.2, only sparse efforts have been reported in the literature for this problem, usually tailored to a specific method or algorithm. So in this thesis, we provide a comprehensive Quality Framework for RDF Graph Summarization to cover the gap that exists in the literature and which functions independently of the summarization algorithm, the underlying KB and the intended application domain. This framework would allow a better, deeper and more complete understanding of the quality of the different summaries and facilitate their comparison.

The framework is independent of the way RDF summarization algorithms work and makes no assumptions on the type or structure neither of the input nor of the final results. We provide metrics that help us understand not only if this is a valid summary but also how a summary compares to another in terms of the specified quality characteristics. In order to achieve this, we compare the summaries against two levels of information possibly available for a RDF KB: the level of the ideal summary of the KB and the level of the instances contained by the KB. For the first level, when an ideal summary is available, either because it has been proposed by a human expert or because we can assume that an existing schema represents perfectly the data graph, we compute how close the proposed summary is to the ideal solution by computing its precision, recall and F-measure against the ideal solution using a novel customized definitions for precision and recall. We compute the precision and recall for each class and its neighborhood (properties and attributes having as domain that class) of the produced summary against the ideal one. We also compute the precision and recall of the whole summary against the ideal one. The first will capture the quality of the summary at the local (class) level, while the second will give us the overall quality in terms of classes' and properties/attributes' precision and recall. Using these metrics, we can understand if classes or properties at the different levels (local vs. schema) are missing or are added in excess (although the latter is not common, it can happen in extreme cases of different algorithms). For the second level, if the ideal summary is not available or usually in addition to it, we are computing if the existing instances (including both class and property instances) are covered (i.e. can be retrieved) and at which degree by the proposed summary. Again we define and compute the summary precision, recall and F-measure against the data

contained in the original KB. One more important aspect that we also consider is the connectivity of the summary, i.e. is the summary a connected graph? So, we propose a new metric to compute the connectivity of the proposed summary compared to the ideal one, since in many cases (like, e.g., when we want to query) this is an important factor. Ideally, a summary should be a connected graph.

We evaluated the Quality Framework using a set of ten different and diverse datasets (RDF KBs) with different characteristics like size (in number of triples), number of classes and properties, number of instances, etc. We used three different algorithms for RDF Graph Summarization picked from the literature (that work in substantially different ways) and used the Quality Framework to understand the different behavior of each algorithm when summarizing each KB. Results show that the Quality Framework captures different behaviors at a very detailed level and thus provides to the user adequate information to decide which algorithm is more suitable for each use case and KB. In summary we can say that the proposed framework allows for understanding the quality of the different summaries at different levels. The users can pick the metrics that better fit to the task for which they need to pick a summary. Finally, we could summarize our contribution as presenting a quality framework that:

- Evaluates the quality of RDF Graph Summaries, where a combined effort is made to summarize, while preserving existing important semantics, basic structure and coherence;
- Works at different levels, both trying to understand the comparison of the two summaries (ideal and computed) at the schema and the instance levels, while previous approaches were mainly dealing with one level (which corresponds to the instance level in our approach);
- Provides novel customized definitions for precision and recall for summaries, thus allowing better capturing of the quality of the results – so we go beyond the standard property and recall definitions;
- Adds the discussion on the connectivity of the computed summary and tries to promote summaries that are more connected. This is quite crucial if we want to later on query the summary using standard RDF tools.

Measure	What it indicates	How it is computed
$SchemaRecall(c, \Pi)$	Schema recall of a class c over the set of patterns Π.	Divide the number of relevant class's properties that are reported in Π on the total number class's properties.
$SchemaRec_{ClassAll}$	Overall schema class recall.	Compute the mean of the various $SchemaRecall(c, \Pi)$ for all the classes c of the ground-truth Schema S.
$Sim(pa, c)$	Similarity between a class c and a pattern pa.	Divide the number of common properties between the class c and the pattern pa on the total number of pa propertiespa.
$Nps(c)$	The number of patterns that represent the class c	Count all the patterns having $Sim(pa, c)$¿0.
$SchemaPrec(c, \Pi)$	Schema class precision of the class c over the set of patterns Π.	Sum the $sim(pa, c)$ for all the patterns of Π.
$SchemaPrec_{ClassAll}$	Overall schema class precision.	Compute the mean of the various class precision values $SchemaPrec(c, \Pi)$ for all the retrieved classes of the ground-truth Schema S.
$SchemaF1_c$	Schema class F-Measure.	Combine the $SchemaPrec_{ClassAll}$ and $SchemaRec_{ClassAll}$ using the standard formula of the F-Measure.
$SchemaRec_{PropertyAll}$	Overall Schema property recall.	Divide the number of relevant properties extracted by the summary on the total number of properties in the ground truth schema.
$SchemaF1_p$	Schema property F-Measure.	Combine the $SchemaPrec_{PropertyAll}$ and $SchemaRec_{PropertyAll}$ using the standard formula of the F-Measure
$SchemaF1$	Overall schema F-measure.	Combine the class schema F-Measure $SchemaF1_c$ and property schema F-Measure $SchemaF1_p$.

Table 6.1 – Summary description of the proposed Schema Metrics

6.2 Quality Assessment Model

In this section, we present our quality assessment framework that allows us to evaluate the quality of different RDF summaries in a comprehensive and coherent way. The framework is independent of the way the summarization algorithms work and makes no assumptions on the type or structure neither of the input nor of the final results, besides being expressed in RDF; this is required in order to guarantee the validity of the result but can be easily extended to other cases of semantic summarization, like for graphs expressed in OWL or Description Logics. In order to achieve this, we work at two levels:

- *schema level*, where if an ideal summary exists, the summary is compared with it by computing the precision and recall for *each* class and its neighborhood (properties and attributes having as domain that class) of the produced summary against the ideal one; we also compute the precision and recall of the whole summary against the ideal one. The first will capture the quality of the summary at the local (class) level, while the second will give us the overall quality in terms of classes' and properties'/attributes' precision and recall.

- *instance level*, where the coverage that the summary provides for class and property instances is calculated, i.e. how many instances will be retrieved if we query the whole summary graph. We use again precision and recall against the contents of the original KB.

At the end, a metric is presented that provides an indication of the quality of the graph summary by measuring whether or not the summary is a connected graph. Ideally, a summary should be a connected graph but this also depends on the actual data stored in the Knowledge Base. Thus a disconnected graph could be an indication of the data quality in the KB and not necessarily a problem of the summarization process. Nevertheless, we present it here as another indicator of the quality process, especially if the summary is compared with an ideal one, but for the reason mentioned before we avoid to combine it with the rest of the presented metrics. Finally, we discuss some results that combine these metrics and interpret their meaning.

6.2.1 Quality Model at the schema level

In this section, we present the part of our quality assessment framework used to evaluate the quality of an RDF graph summary against a ground truth summary **(S)** (e.g. one provided by an expert). We measure how close the proposed summary is to the ground truth summary by computing its precision and recall against this ground truth. We suggest that we compute both the precision and recall at the class and at the property level and at the overall summary level. Table 6.1 gives us a summary description of the schema-level proposed measures.

6.2.1.1 Precision and Recall for classes

We present here the recall and the precision metrics for the classes of the detected patterns against a ground truth summary S. We first introduce the recall over the classes, which is the fraction of relevant classes that are reported in the summary. Given a set of knowledge patterns Π (as defined in Section 2.4.3 and referred commonly as patterns from now on) and a set of classes C $\in S$, we start by defining the recall of a class c $\in$ C over the set of patterns Π as the fraction of the relevant class's properties (namely properties that have this class as their domain) that are reported in Π, we denote it by schema class recall $SchemaRec(c, \Pi)$:

$$SchemaRecall(c, \Pi) = \frac{|\bigcup_{pa \in \Pi} (A(c) \cap A(pa))|}{|A(c)|} \tag{6.1}$$

The A(pa) is the set of properties and attributes involved in the pattern pa, and the A(c) is the set of properties and attributes of the ideal class c. Thus, the overall summary recall using the classes $SchemaRec_{ClassAll}$ is computed as the mean of the various schema classes recall $SchemaRecall(c, \Pi)$ for all the classes c of the ground-truth Schema S.

$$SchemaRec_{ClassAll} = \frac{1}{|C|} \sum_{c \in C} SchemaRecall(c, \Pi) \tag{6.2}$$

The precision is the fraction of retrieved classes and properties of the summary that are relevant. If a knowledge pattern of a summary carries a *typeof* link then this pattern is relevant to a specific class if the typeof points to this class, if not this is not relevant to this class. If no *typeof* information exists then we use the available properties and attributes to evaluate the similarity between a class and a pattern. Thus we define the L(c, pa) function to capture this information and we add this to the similarity function.

$$L(c, pa) = \begin{cases} 1, & if \text{typeof(pa)} = c \text{ or typeof(pa)} = \emptyset \\ 0, & \text{otherwise} \end{cases} \tag{6.3}$$

The similarity between a class c in the ideal summary and a pattern pa, denoted $Sim(pa, c)$, in the computed summary is defined as the number of common properties between class c and patterns pa divided on the total number of the properties of the patterns pa:

$$Sim(pa, c) = L(pa, c) * \frac{|A(c) \cap A(pa)|}{|A(pa)|} \tag{6.4}$$

Given that a class might be represented by more than one knowledge patterns, depending on the algorithm used, we are interested in introducing a way to penalize cases where this happens, thus favoring smaller summaries over bigger ones. We achieve this by introducing a weight function that allows us to reduce the similarity value if this is based on consuming multiple patterns. Thus we introduce the following exponential function $W(c)$, which uses coefficient a to allow variations if needed in the future, and is chosen based on experimental evaluation of the functions that could provide us with a smooth decay in similarity as patterns' number increases. The $Nps(c)$ is the number of patterns that represent the class c and $\alpha \in [1, 10]$.

We define the T(c, pa) function to capture if a pattern pa can be used to represent the class c; this function returns 1 if the similarity function between the pattern pa

and c is bigger than zero (so the pattern covers some of the elements that define the class) and zero otherwise.

$$T(c, pa) = \begin{cases} 1, & if Sim(pa, c) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{6.5}$$

Based on the T(c, pa) function, the number of patterns $Nps(c)$ that represent the class is defined as follows:

$$Nps(c) = \sum_{pa \in \Pi} T(c, pa) \tag{6.6}$$

$$W(c) = e^{1 - \sqrt[\alpha]{Nps(c)}} \tag{6.7}$$

Based on this weight function we define the class precision metric for every pattern pa in the computed summary and every class c in the ground truth summary as follows:

$$SchemaPrec(c, \Pi) = W(c) * \frac{\sum_{pa \in \Pi} Sim(pa, c)}{Nps(c)} \tag{6.8}$$

Thus, we define the schema class precision $SchemaPrec_{ClassAll}$ as the mean of the various class precision values $SchemaPrec(c, \Pi)$ for all the classes of the ground-truth Schema S.

$$SchemaPrec_{ClassAll} = \frac{\sum_{c \in C} SchemaPrec(c, \Pi)}{|C1|} \tag{6.9}$$

where C1 $\subseteq C$ is the list of all the ground truth's retrieved classes, or in other words, is the list of the ground truth's classes for which $SchemaPrec(c, \Pi) > 0$.

However, neither precision nor recall alone can accurately assess the match quality. In particular, recall can easily be maximized at the expense of a poor precision by returning as many correspondences as possible. On the other side, a high precision can be achieved at the expense of a poor recall by returning only few (correct) correspondences. Hence it is necessary to consider both measures and and express this through a combined measure; we use the F-Measure for this purpose, namely $SchemaF1_c$:

$$SchemaF1_c = 2 * \frac{SchemaPrec_{ClassAll} * SchemaRec_{ClassAll}}{SchemaPrec_{ClassAll} + SchemaRec_{ClassAll}} \tag{6.10}$$

6.2.1.2 Precision and Recall for properties

The overall recall at the property level, namely $SchemaRec_{PropertyAll}$ is computed as the ratio between the number of common properties extracted by the summary and the ones in the ground truth summary divided by the number of properties in the ground truth summary:

$$SchemaRec_{PropertyAll} = \frac{|\bigcup_{pa \in \Pi} A(pa) \cap \bigcup_{c \in C} A(c)|}{|\bigcup_{c \in C} A(c)|} \tag{6.11}$$

We note that the schema precision at the property level in our experiments is always equal to 1 (see Section 6), which means that in our examples there are no false positives for properties. Summarization algorithms do not invent new properties but they might report some properties that are not present in the ground truth summary. So, precision for properties namely $SchemaPre_{PropertyAll}$, is computed as the ratio between the number of common properties between the extracted summary and the number of properties existing in the ground truth summary and is as follows:

$$SchemaPrec_{PropertyAll} = \frac{|\bigcup_{pa \in \Pi} A(pa) \cap \bigcup_{c \in C} A(c)|}{|\bigcup_{pa \in \Pi} A(pa)|} \tag{6.12}$$

Thus, the F-Measure for the schema properties, namely $SchemaF1_p$ will be calculated as:

$$SchemaF1_p = 2 * \frac{SchemaPrec_{PropertyAll} * SchemaRec_{PropertyAll}}{SchemaPrec_{PropertyAll} + SchemaRec_{PropertyAll}} \tag{6.13}$$

6.2.1.3 Overall Schema level F-measure

After defining the individual metrics for the class schema F-Measure $SchemaF1_c$ and property schema F-Measure $SchemaF1_p$, we can define the combined overall schema F-measure $SchemaF1$ as the weighted harmonic mean of the class schema F-Measure and property schema F-Measure :

$$SchemaF1 = \beta * SchemaF1_p + (1 - \beta) * SchemaF1_c \tag{6.14}$$

where the weight $\beta \in [0, 1]$. The overall schema F-measure provides a better insight on the combination of the number of classes found by the summarization algorithm and the overall number of properties discovered. The metrics used to compute precision and recall at schema class level include (all) the properties discovered (equations (1),

(4) and (11), (12) respectively). But by penalizing the expression of a class by more than one patterns while computing the schema class F-measure, the quality of the results of the summarization algorithms towards the properties gets blurred and is also penalized, which should not be the case. So, we use the schema property recall and precision to recover the notion of quality on property discovery in all cases for the whole schema, so algorithms that will discover all or most of the properties will get acknowledged, even if they use multiple knowledge patterns to do that. Even in the case of not having multiple patterns representing a class the computations for the schema property recall and precision are not redundant because they capture different aspects of the summary's quality, since the overall schema class level precision and recall is an average and thus not the same as the overall property level precision and recall. So the first one tells us how much of the semantics of the classes is recovered in the summary, while the second tells us how many of the overall schema properties are present regardless of where they belong.

6.2.2 Quality Model At the Instance Level

We measure the quality with regard to the instances by introducing the notion of the coverage of the instances of the original KB, i.e. how many of the original class and property instances are successfully represented by the computed RDF summary graph (e.g. can be retrieved in the case of a SPARQL query). This requires computing both the precision and recall at the class instance and at the property instance levels. Table 6.2 gives us a summary description of the proposed instance level metrics.

6.2.2.1 Precision and Recall for class instances

The overall recall at the instance class level is the total number of the class instances represented by the computed summary divided on the total number of instances of the original KB D.

$$InstanceRec_{ClassAll} = \frac{|instances(\Pi)|}{|instances(D)|} \tag{6.15}$$

The class $instances(\Pi)$ is the list of instances covered by the set of patterns Π, $instances(D)$ is the list of all instances of the original KB D. To avoid the problem of overlapping of instances in several patterns which will cause the over-coverage, we calculate the $instances(\Pi)$, $instances(D)$ as follows:

$$instances(\Pi) = \bigcup_{pa \in \Pi} instances(pa) \tag{6.16}$$

Measure	What it indicates	How it is computed
$instances(c)$	The list of class c instances.	——-
$instances(p)$	The list of subjects which have the property p.	——-
$instances(pa)$	The list of covered class instances by the pattern *pa*.	—-
$instances(\Pi)$	The list of class instances covered by the set of patternsΠ.	—-
$instances(D)$	The list of all class instances of original KB D.	—-
$Cov_c(c, pa)$	The list of the class instances which are represented by a pattern *pa*.	Get the $instances(pa)$ if the pattern pa is relevant to the class c or $\emptyset$ otherwise.
$instances(c, \Pi)$	The total number of class instances that are reported by a set of patterns Π representing the class c.	Sum the $\|Cov_c(c, pa)\|$ for all the patterns of the Π.
$InstancePrec(c, \Pi)$	The instance class precision of a class c over the set of patterns Π.	Divide the number of original instances of the class c reported in Π on $instances(c, \Pi)$.
$InstancePrec_{ClassAll}$	Overall instance class precision.	The mean of the various $InstancePrec(c, \Pi)$ for all the classes of the ground-truth Schema S.
$InstanceF1_c$	Instance class F-Measure.	Combine the $InstancePrec_{ClassAll}$ and $SchemaRec_{ClassAll}$ using the standard formula of the F-Measure.
$Cov_p(p, pa)$	The list of the original property instances which are successfully represented by a pattern pa.	Get the $instances(p)$ if the property p is reported in the pattern pa or get $\emptyset$ otherwise.
$instances(p, \Pi)$	The list of the original property p instances that are successfully covered by a set of patterns Π.	The Union of the $Cov_p(p, pa)$ for all the in Π.
$InstanceRec(p, \Pi)$	The instance property recall.	Divide $\|\Pi\ instances(p, \Pi)\|$ on $instances(p)$.
$nstanceRec_{PropertyAll}$	Overall recall at the instance property lebel	Weighted mean of the various $InstanceRec(p, \Pi)$ for all the properties of the ground-truth.
$InstancePrec(p, \Pi)$:	The precision of a property p in P over the set of patterns Π.	
$InstancePrec_{PropertyAll}$	Overall instance property precision	Mean of the various $InstanceRec(p, \Pi)$ for all the covered properties of the ground-truth.
$InstanceF1_p$:	Instance property F-Measure	Combine the $InstancePrec_{PropertyAll}$ and $InstanceaRec_{PropertyAll}$ using the standard formula of the F-Measure.
$InstanceF1$	Overall instance F-measure .	Combine the class Instance F-Measure $InstanceF1_c$ and property Instance F-Measure $InstanceF1_p$.

Table 6.2 – Summary Description of the proposed Instance Metrics

$$instances(D) = \bigcup_{c \in C} instances(c) \tag{6.17}$$

The $instances(pa)$ denotes the list of covered instances by the pattern *pa* and the $instances(c)$ denotes the list of instances of the type *c* in the original KB D.

We denote by $Cov_c(c, pa)$, the list of the class instances which are represented by a pattern *pa*:

$$Cov_c(c, pa) = \begin{cases} instances(pa), & if L(c, pa) = 1 \\ \emptyset, & \text{otherwise} \end{cases} \tag{6.18}$$

Thus, we can define the total number of class instances $instances(c, \Pi)$ that are reported by a set of patterns Π representing the class c as:

$$instances(c, \Pi) = \sum_{pa \in \Pi} |Cov_c(c, pa)| \tag{6.19}$$

We define $InstancePrec(c, \Pi)$ the instance precision of a class c in C over the set of patterns Π as follows:

$$InstancePrec(c, \Pi) = \frac{|instances(c) \cap instances(c, \Pi)|}{|instances(c, \Pi)|} \tag{6.20}$$

Thus, we define the overall instance class precision denoted by $InstancePrec_{ClassAll}$ as the weighted mean of the various $InstancePrec(c, \Pi)$ for all the retrieved classes:

$$InstancePrec_{ClassAll} = \sum_{c \in C} wi(c) * InstancePrec(c, \Pi) \tag{6.21}$$

The *wi(c)* is the weight of a class c and it measures the percentage of class instances of the class c with respect to the total number of class instances in the KB. This is used to weight in the importance of the specific class in terms of the number of instances it "represents"; so the more instances it "represents" the bigger the weight. It is defined as the number of instances of class c in the KB $instances(c)$ compared to the total number of class instances in the KB $instances(D)$.

$$wi(c) = \frac{instances(c)}{instance(D)} \tag{6.22}$$

The overall instance class recall and the overall instance class precision are combined by the instance class F-Measure, namely $InstanceF1_c$:

$$InstanceF1_c = 2 * \frac{InstancePrec_{ClassAll} * InstanceRec_{ClassAll}}{InstancePrec_{ClassAll} + InstancePrec_{ClassAll}} \tag{6.23}$$

6.2.2.2 Precision and Recall at Property Level

The $Cov(p, pa)$ represents the list of the original property instances which are successfully represented by a pattern pa:

$$Cov_p(p, pa) = \begin{cases} instances(pa), & if p \in pa \\ \emptyset, & \text{otherwise} \end{cases} \tag{6.24}$$

We denote by the $instances(p, \Pi)$ the list of the original property instances that are successfully covered by a set of patterns Π:

$$Instance(p, \Pi)) = \bigcup_{pa \in \Pi} (Cov_p(p, pa) \cap instances(p)) \tag{6.25}$$

The *instances(p)* denotes the list of original instances which have the property p in original KB D. Thus, the instance property recall $InstanceRec(p, \Pi)$ defined as:

$$InstanceRec(p, \Pi) = \frac{|instances(p, \Pi) \cap instances(p)|}{|instances(p)|} \tag{6.26}$$

The overall recall at the instance property level
$InstanceRec_{PropertyAll}$ is computed as the weighted mean of the various instance property recall $InstanceRec$ for all the properties of the ground-truth.

$$InstanceRec_{PropertyAll} = \sum_{p \in P} wi(p) * InstanceRec(p, \Pi) \tag{6.27}$$

The *wi(p)* is the weight of the property p and it measures the percentage of instances of a property p with respect to the total number of property instances in the KB. It is defined as the number of instances of property p in the KB $instances(p)$ compared to the total number of property instances in the KB. Again the idea here is to capture the important properties by weighting in the number of property instances each one represents.

$$wi(p) = \frac{instances(p)}{\sum_{p1 \in P} instances(p1)} \tag{6.28}$$

We define $InstancePrec(p, \Pi)$, the precision of a property p in P over the set of patterns Π as follows:

$$InstancePrec(p, \Pi) = \frac{|instances(p) \cap instances(p, \Pi)|}{|instances(p, \Pi)|} \tag{6.29}$$

Thus, we define the overall instance precision for property instances denoted by $InstancePrec_{PropertyAll}$ as the mean of the various $InstancePrec(c, \Pi)$ for all the properties of the ground-truth Schema S:

$$InstancePrec_{PropertyAll} = \frac{\sum_{p \in P} InstancePrec(p, \Pi)}{|P1|} \tag{6.30}$$

where P1 $\subseteq P$ is the list of retrieved properties, or in other words the list of properties having
$InstancePrec(p, \Pi) > 0$. The overall instance recall and the overall instance precision for property instances are combined by the instance class F-Measure, namely $SchemaF1_c$:

$$InstanceF1_p = 2 * \frac{InstancePrec_{PropertyAll} * InstanceRec_{PropertyAll}}{InstancePrec_{PropertyAll} + InstancePrec_{PropertyAll}} \tag{6.31}$$

Thus, the overall instance F-measure $InstanceF1$ is obtained by combining the overall instance schema F-Measure $InstanceF1_c$ and overall property instance F-Measure $InstanceF1_p$.

$$InstanceF1 = \beta * InstanceF1_p + (1 - \beta) * InstanceF1_c \tag{6.32}$$

where the weight $\beta \in [0, 1]$. The overall instance F-measure can be viewed as a compromise between overall class instance F-Measure and overall property instance F-Measure. It is high only when both overall class and property instance F-Measure are high. It is equivalent to the class instance F-Measure when $\beta = 0$ and to the property instance F-Measure when $\beta = 1$.

We need also to make one last point for the computation of the instance-level metrics, for the case when our KB contains no schema information. In this case and in order to make the instance level class precision and recall computable we need to annotate the KB with $typeof(class)$ so as to be able to compute the metrics presented above. If not, we will declare the instance level class precision and recall uncomputable but we will be able to continue the quality assessment using the rest of the metrics, including property precision and recall at the instance level. This demonstrates that the proposed Quality Framework will work under all circumstances.

6.2.3 Connectivity

One more important aspect that we need to consider, is the connectivity of the summary, i.e. the summary is or not a connected graph. So, we propose a new metric to measure how many disconnected (sub)graphs exist in the summary and what percentage of the classes in the ground truth they represent. The connectivity of a summary graph G_s Con(G_s) is defined as the number of the connected components (independent subgraphs) of the summary graph divided on the number of the connected components (independent subgraphs) of the ground truth.

$$Con(G_s) = \frac{numberofconnectedcomponentsofthesummary}{numberofconnectedcomponentsofthegroundtruth} \tag{6.33}$$

We compute the number of connected components for the summary (and in the same manner for the ground truth) using the breadth-first search algorithm, where given a particular node n, we will find the entire connected component containing n (and no more) before returning. To find all the connected components of a summary (or the ground truth) graph, we loop through the nodes, starting a new breadth-first search, whenever the loop reaches a node that has not already been included in a previously found connected component. This metric gives an indication of the connectivity of a generated summary. If it is 1, it shows that the summary is a graph connected as well as the ground truth graph, but if it is bigger than 1 it means that the summary is more disconnected than desired. The higher the connectivity, the more the links that are missing between the classes of the computed graph compared to the

ground truth; this could even capture correctly a completely disconnected summary graph. This metric allows us to penalize (if needed) disconnected (compared to the ground truth) summary graphs and allows for progressive linear penalties. It is also theoretically possible that the summary graph will be more connected than the ground truth graph, this will give us values less than 1. The value of the connectivity can tend to but will never reach zero (0).

6.3 Implementation of the Quality Framework

We implemented our Quality Framework as a software that takes as input the results of any RDF Graph Summarization algorithm and the ideal summary and computes the different metrics that are required to capture the quality of the results at the different levels described earlier. It outputs the values for the different metrics in an automated fashion and allows to compute F-measures where applicable. In principle it can be used to compare the quality of any summary against an ideal one or to understand how close two summaries are to one another. It is implemented in Java and it is available as open source software, here: *https://github.com/ETIS-MIDI/Quality-Metrics- For-RDF-Graph-Summarization.*

We describe the different steps applied in the form of algorithmic pseudocode that allows to track the computations taking place at the different levels that the Quality Framework operates. The pseudocode of Algorithm 11 gives an overview of our implementation of the computations at the schema level. The function which computes the schema class recall is shown in Algorithm 12, while the one, which computes the schema class precision, is shown in Algorithm 13. The function which computes the schema property precision and recall is shown Algorithm 14. In the same manner, the pseudocode in Algorithm 15 gives an overview of the computations at the instance level. The function which computes the instance class recall is shown in Algorithm 16, while the one, which computes the instance class precision, is shown in Algorithm 17. The function which computes the instance property precision and recall is shown in Algorithm 18.

6.4 Illustrative Example

In an effort to better explain the way our quality assessment framework works and captures the differences among the different summaries we provide a working example. We have created an artificial dataset containing information about music artists and

Algorithm 11 Schema Level Metrics

INPUT: Set of knowledge patterns $\Pi = \{Pa_i : i : 1.....N\}$, ideal summary S=$\{C, P, I\}$ where C, P, and I are Set of classes, properties and instances, α and β.
OUTPUT: Rec_c schema class Recall, $Prec_c$ schema class precision , Rec_p schema property recall , $Prec_p$ schema property precision, F_c Schema class F-Measure , F_p Schema property F-Measure and SchemaF1 overall schema F-Measure .

1: **Begin**
2: $Rec_c \leftarrow$ Schema-Class-Recall(C,Π)
3: $Prec_c \leftarrow$ Schema-Class-Precision(C,Π, α)
4: $F_c \leftarrow \dfrac{Prec_c * Rec_c}{Prec_c + Rec_c}$
5: $Prec_p, Rrec_p \leftarrow$Schema-Property-Recall-Precision(C,Π)
6: $F_p \leftarrow \dfrac{Prec_p * Rec_p}{Prec_p + Rec_p}$
7: $SchemaF1 \leftarrow \beta * F_p + (1 - \beta) * F_c$
8: **End**

Algorithm 12 Function Schema Class Recall

1: **function** SCHEMA-CLASS-RECALL(C,Π)
2: $Rec_c \leftarrow 0$ ▷ schema class recall
3: **for** each c ∈ C **do**
4: $ListA \leftarrow \emptyset$ ▷ the list of common properties of c and Π
5: **for** each pa ∈ Π **do**
6: $ListA \leftarrow ListA \cup (A(c) \cap A(pa))$
7: ▷ where A(c), A(Pa) are the set properties of pa and c
8: **end for**
9: rec $\leftarrow \dfrac{|ListA|}{|A(c)|}$
10: $Rec_c \leftarrow Rec_c + rec$
11: **end for**
12: $Rec_c \leftarrow \dfrac{Rec_c}{|C|}$
13: **return** Rec_c
14: **end function**

Algorithm 13 Function Schema Class Precision

function SCHEMA-CLASS-PRECISION(C,Π, α)
  $Prec_c \leftarrow 0$ ▷ the Schema class precision
  for each c ∈ C **do**
    Nps ← 0
    prec ← 0
    for each Pa ∈ Π **do**
      compute the similarity Sim(pa,c) using the equation (4)
      prec ← $prec + Sim(pa, c)$
      if Sim(pa,c)>0 **then**
        Nps ← $Nps + 1$
      end if
    end for
    W ← $e^{1-\sqrt[\alpha]{Nps}}$
    Prec ← $w * \frac{prec}{Nps}$
    $Prec_c \leftarrow Prec_c + Prec$
  end for
  $Prec_c \leftarrow \frac{Prec_c}{|C|}$
  Return $Prec_c$
end function

Algorithm 14 Function Schema Property Precision and Recall

function SCHEMA-PROPERTY-RECALL-PRECISION(C,Π)
  $Rec_p \leftarrow 0$ ▷ the Schema property recall
  $Prec_p \leftarrow 0$ ▷ the Schema property precision
  $ListA \leftarrow \emptyset$ ▷ all the properties involved in C
  $ListB \leftarrow \emptyset$ ▷ all the properties involved in Π
  for each c ∈ C **do**
    $ListA \leftarrow (ListA \cup (A(c))$
  end for
  for each pa ∈ Π **do**
    $ListB \leftarrow (ListB \cup (A(pa))$
  end for
  $ListC \leftarrow (ListA \cap ListB)$ ▷ the common properties between ListA and ListB
  $Rec_p \leftarrow \frac{|ListC|}{|ListA|}$
  $Prec_p \leftarrow \frac{|ListC|}{|ListB|}$
  return $Rec_p, Prec_p$
end function

Algorithm 15 Instance Level Metrics

INPUT: Set of knowledge patterns $\Pi = \{Pa_i : i : 1.....N\}$, Ideal summary S and β.
OUTPUT: $InsRec_c$ Instance class Recall, $InsPrec_c$ Instance class precision , $InsRec_p$ Instance property recall, $InsPrec_p$ Instance property precision, $InsF_c$ Instance class F-Measure, $InsF_p$ Instance property F-Measure and InstanceF1 overall Instance F-Measure.

1: **Begin**
2: $InsRec_c \leftarrow$ Instance-Class-Recall(C,Π)
3: $InstPrec_c \leftarrow$ Instance-Class-Precision(C,Π, α)
4: $InsF_c \leftarrow \dfrac{InsPrec_c * InsRec_c}{InsPrec_c + InsRec_c}$
5: $InsPrec_p, InsRrec_p \leftarrow$ Instance-Property-Recall-Precision(C,Π)
6: $InsF_p \leftarrow \dfrac{InsPrec_p * InsRec_p}{InsPrec_p + InsRec_p}$
7: $InstanceF1 \leftarrow \beta * InsF_p + (1 - \beta) * InsF_c$
8: **End**

Algorithm 16 Function Instance Class Recall

1: **function** INSTANCE-CLASS-RECALL(C,Π)
2: $InsRec_c \leftarrow 0$
3: $Instances_\Pi \leftarrow \emptyset$ ▷ list of all the class instances reported in Π
4: $Instances_C \leftarrow \emptyset$ ▷ list of all the class instances involved in D
5: **for** each c ∈ C **do**
6: $Instances_C \leftarrow (Instances_C \cup instances(c))$
7: **end for**
8: **for** each pa ∈ Π **do**
9: $Instances_\Pi \leftarrow (Instances_\Pi \cup (instances(pa))$
10: **end for**
11: $InsRec_c \leftarrow \dfrac{|Instances_\Pi|}{|Instances_C|}$ ▷ $InsRec_c$ Instance class recall
12: **return** $InsRec_c$
13: **end function**

Algorithm 17 Function Instance Class Precision

function INSTANCE-CLASS-PRECISION(C,Π)
  $InsPrec_c \leftarrow 0$
  for each c ∈ C **do**
    $Cov_c \leftarrow 0$
    CovList ← ∅
    prec ← 0
    for each pa ∈ Π **do**
      if L(pa,c)=1 **then**
        $Cov_c \leftarrow Cov_c + |instances(pa)|$
        CovList ← $CovList \cup (instances(c) \cap instances(pa))$ ▷ the list of class c instances reported in π
      end if
    end for
    $InsPrec_c \leftarrow InsPrec_c + \frac{CovList}{Cov_c}$
  end for
  $InsPrec_c \leftarrow \frac{InsPrec_c}{|C1|}$
  Return $InsPrec_c$
end function

Algorithm 18 Function Instance Property Precision and Recall

function SCHEMA-CLASS-PRECISION(P,Π)
  $InsRec_p \leftarrow 0$
  $InsPrec_p \leftarrow 0$
  for each p ∈ P **do**
    CovList ← ∅
    $Cov_p \leftarrow 0$
    for each pa ∈ Π **do**
      if p ∈ pa **then**
        CovList ← $CovList \cup (instances(p) \cap instances(pa))$
        $Cov_p \leftarrow Cov_p + |instances(pa)|$
      end if
    end for
    $InsRec_p \leftarrow InsRec_p + \frac{CovList}{|instances(p)|}$
    $InsPrec_p \leftarrow Prec_p + \frac{CovList}{cov_p}$
  end for
  $InsRec_p \leftarrow \frac{InsRec_p}{|P|}$
  $InsPrec_p \leftarrow \frac{InsPrec_p}{|P1|}$
  Return $InsPrec_p, InsRec_p$
end function

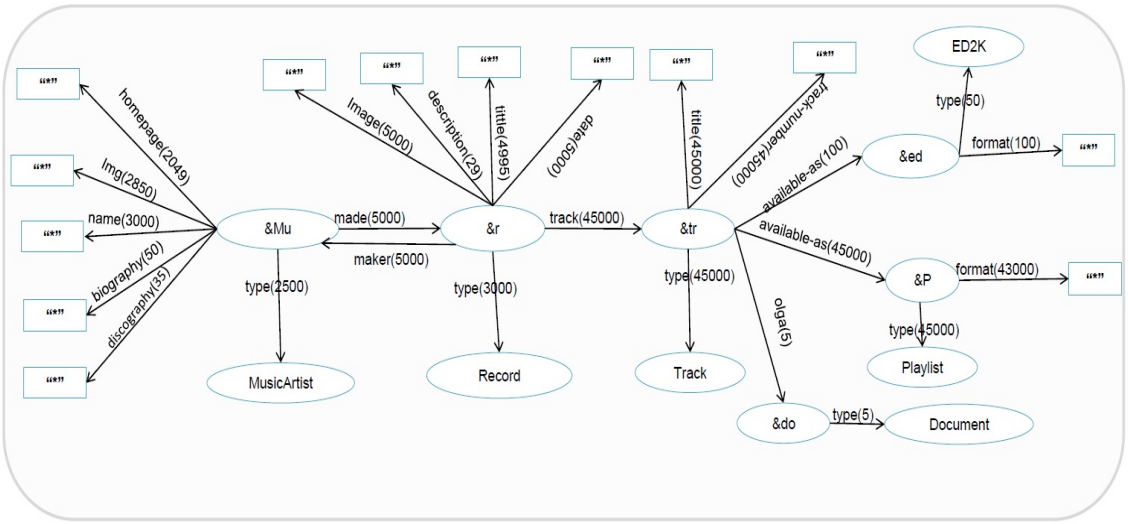


Figure 6.1 – An artificial dataset about music artists and their productions

their productions. Figure 6.1 shows a visualization example of the RDF graph of this dataset. We have 3000 resources describing the music-artists and all of them have the name and made properties, while only 2500 resources have the rdf:type property, 2049 resources have the homepage property, 2850 have the img property, 50 resources have the biography property. We can also notice that we have 5000 resources describing the records and all of them have the date, image, track and maker properties, while 4995 resources have the title property and only 28 resources have the description property. There are also 45000 resources describing the tracks and all of them have the rdf:type, title, track-number and available-as properties, while only 5 resources have the olga property (used to link a track to a Document for tracking in the On-Line Guitar Archive). These tracks are available as a Playlist or/and as ED2K formats. Figure 6.2 shows an ideal summary for this dataset as was suggested by an expert.

6.4.1 Results on the Illustrative Example

As we have already mentioned in section 3.1.2, several RDF graph summarization algorithms are reported in the literature grouped into four main categories. Thus, in addition to our RDF graph summarization approach which was described in chapter 4, we have selected two of the most well performing RDF graph summarization algorithms [55, 26] according to their authors and based on the results reported in the literature. These two approaches were already described in section 3.1.2. Our selection of these algorithms was also based on specific properties and features that they demonstrate: (a) they do not require the presence of RDF schema (triples) in

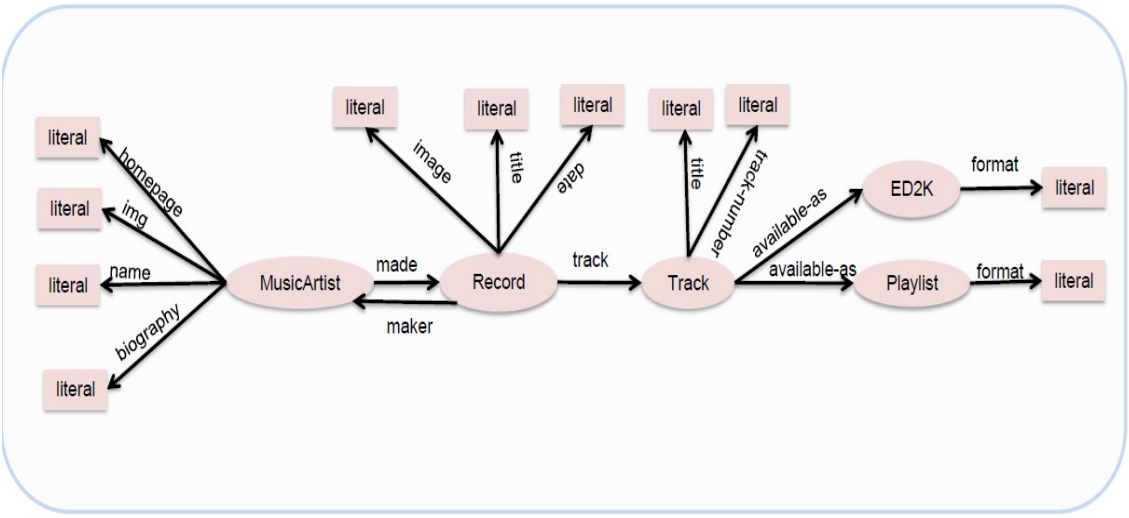


Figure 6.2 – The ideal summary of the dataset depicted in Figure 6.1

order to work properly, (b) they work on both homogeneous and heterogeneous KBs, (c) they provide statistical information about the available data (which can be used to estimate a query's expected results' size), and (d) they provide a summary graph that is considerably smaller than the original graph. The implementations of two algorithms were not available from the original authors so we had to implement them ourselves in Java, based on the corresponding papers.

Tables 6.3, 6.4 and 6.5 present the three RDF summaries generated using the three algorithms: ExpLOD, Campinas et al and ours respectively. The first column shows the pattern id, the second shows the predicates involved in the pattern, while the third column shows the corresponding ideal summary class for a pattern. The last column shows the number of instances per pattern. The Figures 6.3, 6.4 and 6.5 are a visualization representing for three RDF summaries generated using ExpLOD, Campinas et al. and our algorithm receptively.

6.4.1.1 Schema-level metrics

Here we calculate the precision for the MusicArtist class for the three summaries. We start with the ExpLOD summary described in Table 6.3, Sim(Pa1,MusicArtist)=1, because all the properties of the pattern Pa1 are properties of the MusicArtist in the ideal summary. Actually for each $Pa \in \{Pa1, Pa2, Pa3, Pa4, Pa6, Pa7\}$ Sim(Pa,MusicArtist)=1 for the same reason. Concerning the pattern Pa5, it has 6 properties, 5 of which are properties of MusicArtist that are included in the ideal summary. But the pattern Pa5 has also chosen the discography property, which is not included in the ideal

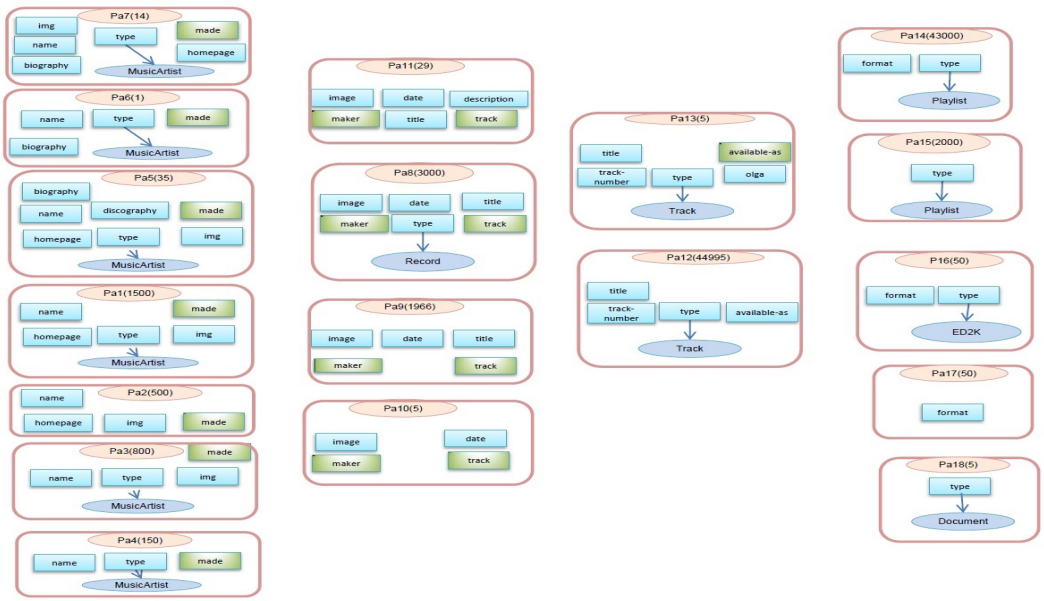


Figure 6.3 – The ExpLOD Summary of the dataset depicted in Figure 6.1

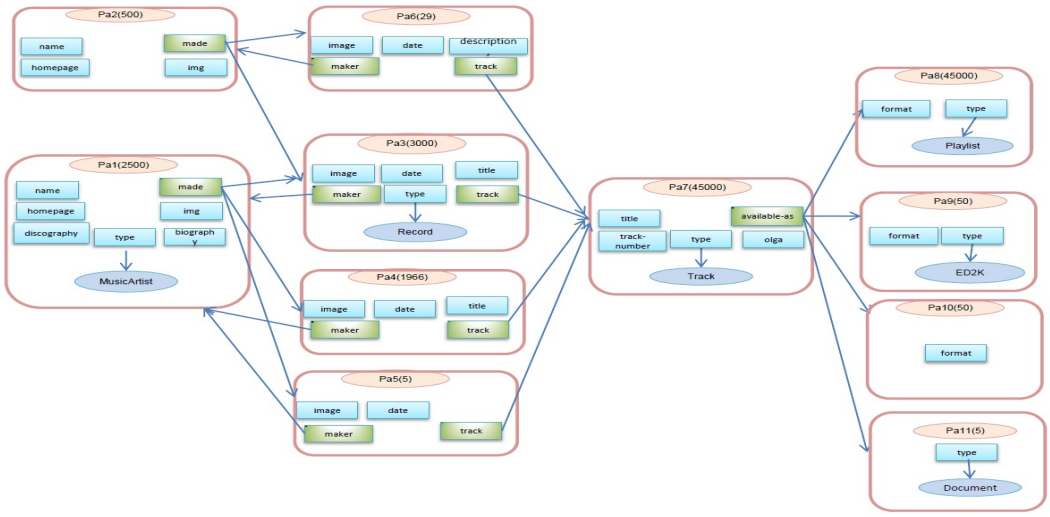


Figure 6.4 – The Campinas et al. Summary of the dataset depicted in Figure 6.1

summary. That makes the Sim(Pa5,MusicArtist) = $\frac{5}{6}$. Any other pattern Pa in the table has $Sim(MusicArtist, Pa) = 0$, because it has a different *typeof* and there are no common properties between these patterns and the MusicArtist class. So the Nps(MusicArtist)=7, and with the $\alpha = 3$ then $W(MusicArtist) = e^{1-\sqrt[3]{7}} = 0.40$. Hence, the precision of the patterns corresponding to the MusicArtist class is: $SchemaPrec(MusicArtist, \Pi) = 0.40 * \frac{1+1+1+1+0.83+1+1}{7} = 0.39$.

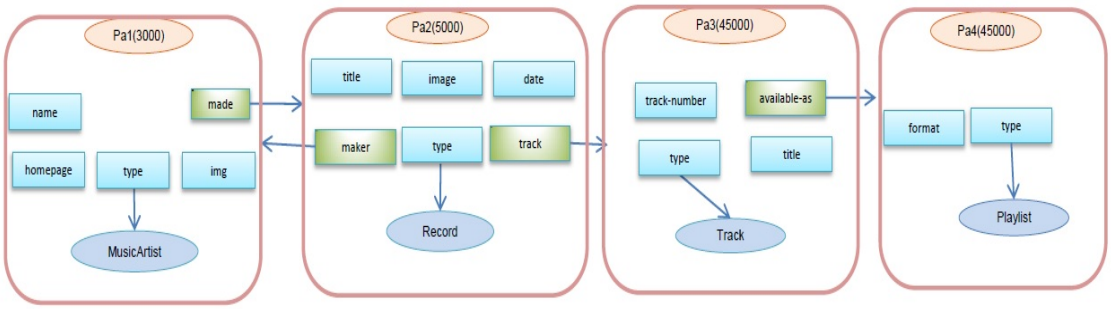


Figure 6.5 – SemSum+ of the dataset depicted in Figure 6.1

Now let us take the Campinas et al. summary described in Table 6.4. In this table we can see that we have two patterns Pa1 and Pa2 represent the MusicArtist class, so $Nps(Musicartist) = 2$, thus the weight: $W(MusicArist) = e^{1-\sqrt[3]{2}} = 0.77$. The first pattern Pa1 has 6 properties where 5 of these 6 properties are properties of MusicArtist in the ideal summary. But it has chosen the discography property, too, which is not included in the ideal summary. That makes Sim(MusicArtist,Pa1)=$\frac{5}{6}$. Respectively, Pa2 has Sim(Musicartist,Pa2)=1, since all of its properties are included in the ideal summary. From all above we conclude the precision of Campinas et al.: $SchemaPrec(MusicArtist, \Pi) = 0.77 * \frac{1 + 0.83}{2} = 0.70$.

Now let us compute the precision of the MusicArtist for our RDF summary depicted in Table 6.5, Sim(Pa1,MusicArtist)=1, because all the properties of the pattern Pa1 are properties of the MusicArtist in the ideal summary. Any other pattern Pa in Table 6.5 has Sim(Musicartist,Pa)=0, because it has a different typeof link and no common properties exist between each one of these patterns and the MusicArtist class. So $Nps(MusicArtist) = 1$, and keeping $\alpha = 3$ then $W(MusicArist) = e^{1-\sqrt[3]{1}} = 1$. Hence, the precision of class MusicArtist is: $SchemaPrec(MusicArtist, \Pi) = 1 * \frac{1}{1} = 1$.

Following the same procedure, we can calculate the precision for each class in the set of classes of the ideal summary; these results are reported in Table 6.6a. We should also note that the class Document, which is reported in the summaries of the ExpLod and Campinas et al., is not a class in the ideal summary.

Table 6.6b shows the values of the recall for the list of ideal summary classes. We can note that for ExpLOD and Campinas et al, all recall values are 1, as their patterns cover all the properties in the ideal summary. While for ours, the recall for the MusicArtist is 0.8, because pattern Pa1, which represents the MusicArtist class,

does not cover the biography property, so its recall equals 4 properties over 5 in the ideal summary, $SchemaRec(MusicArtist, \Pi) = \frac{4}{5} = 0.80$.

To calculate the schema-level property precision, we notice that each one of the ExpLOD and Campinas et al. has 16 properties, 13 of these 16 are included in the ideal summary, the other three: discography, description, and ogla are not. That makes the property precision for each one of these two summaries $SchemaPrec_{PropertyAll} = \frac{13}{16} = 0.81$. The properties reported by the our RDF summary are all included in the ideal summary, thus its precision is 1.

Concerning the recall at the property level, ExpLOD and Campinas et al. recall equals 1, as they included all the properties in the ideal summary, while ours missed one property which is biography, so its recall is $SchemaRec_{PropertyAll}\frac{12}{13} = 0.92$.

ID	Pattern	corresponding class	Instance number
Pa1	MusicArtist(c), name, img, homepage, made	MusicArtist	1500
Pa2	name, img, homepage, made	MusicArtist	500
Pa3	MusicArtist(c), name, img, made	MusicArtist	800
Pa4	MusicArtist(c), name, made	MusicArtist	150
Pa5	MusicArtist(c), name, img, homepage, made, biography, discography	MusicArtist	35
Pa6	MusicArtist(c), name, made, biography	MusicArtist	1
Pa7	MusicArtist(c), name, made, img, homepage, biography	MusicArtist	14
Pa8	Record(c), image, title, date, maker, track	Record	3000
Pa9	image, title, date, maker, track	Record	1966
Pa10	image, date, maker, track	Record	5
Pa11	image, description, title, date, maker, track	Record	29
Pa12	Track(c), title, track-number, available-as	Track	44995
Pa13	Track(c), title, track-number, available-as, olga	Track	5
Pa14	Playlist(c), format	Playlist	43000
Pa15	Playlist(c)	Playlist 2000	
Pa16	ED2K(c), format	ED2K 50	
Pa17	format	–	50
Pa18	Document	Document	5

Table 6.3 – ExpLOD summary for the dataset depicted in Figure 6.1

ID	Pattern	corresponding class	Instance number
Pa1	MusicArtist(c), name, img, homepage, made, biography, discography	MusicArtist	2500
Pa2	name, img, homepage, made	MusicArtist	500
Pa3	Record(c), image, title, date, maker, track	Record	3000
Pa4	image, title, date, maker, track	Record	1966
Pa5	image, date, maker, track	Record	5
Pa6	image, description, title, date, maker	Record	29
Pa7	Track(c), title, track-number, available-as, olga	Track	45000
Pa8	Playlist(c), format	Playlist	45000
Pa8	ED2K(c), format	ED2K	50
Pa9	format	-	50
Pa10	Document	Document	5

Table 6.4 – Campinas et al. summary for the dataset depicted in Figure 6.1

ID	Pattern	corresponding class	Instance number
Pa1	MusicArtist(c), name, img, homepage, made	MusicArtist	3000
Pa2	Record(c), image, title, date , maker, track	Record	5000
Pa3	Track(c), title, track-number, available-as	Track	45000
Pa4	Playlist(c), format	Playlist	45000

Table 6.5 – SemSum+ summary for the dataset depicted in Figure 6.1

	ExpLod	Campinas et al	our algorithm
$SchemaPrec(MusicArtist, \Pi)$	0.39	0.70	1
$SchemaPrec(Record, \Pi)$	0.52	0.52	1
$SchemaPrec(Track, \Pi)$	0.67	0.80	1
$SchemaPrec(Playlist, \Pi)$	0.64	0.64	1
$SchemaPrec(ED2K, \Pi)$	0.77	0.77	-
$SchemaPrec_{ClassAll}$	0.60	0.69	1

(a) Schema Precision at Class level

	ExpLod	Campinas et al	our algorithm
$SchemaRec(MusicArtist, \Pi)$	1	1	0.80
$SchemaRec(Record, \Pi)$	1	1	1
$SchemaRec(Track, \Pi)$	1	1	1
$SchemaRrc(Playlist, \Pi)$	1	1	1
$SchemaRec(ED2K, \Pi)$	1	1	0
$SchemaRec_{ClassAll}$	1	1	0.76

(b) Schema Recall at Class level

Table 6.6 – Schema Metrics at Class level

6.4.1.2 Instance-level metrics

Table 6.8b shows the values of the recall for the list of distinct properties of the dataset depicted in Figure 6.1. We can note that for ExpLOD and Campinas et al, all recall values are 1, as their patterns cover all the property instances of the datasets. While for our algorithm, the property instance recall values for the biography, discography and description are 0, because these properties are completely missing from our RDF summary.

While Table 6.8a shows the values of the property instance precision. We can note that for ExpLOD, all precision values are 1, as its patterns described in Table 6.3 are correctly identified all the property instances of the datasets. For the example, for the property homepage having 2049 instances in original dataset, you can see that it is included in 4 patterns $\{Pa1, Pa2, Pa5, Pa7\}$, thus *—Instance(biography,* Π*)— = 1500+500+35+14 = 2049.* Hence, the *InstancePrec(homepage,*Π*)* $= \frac{2049}{2049} = 1$. Following the same procedure, we can find that all property precision values are 1 for the Explod summary.

Now let us try to compute the instance precision value for the homepage property for the Campinas et al. summary described in Table 6.4. From this table 6.4 we can note that this property is included in the patterns Pa1 and Pa2, thus $|Instance(hompage, \Pi)| = 2500 + 500$. Hence,the InstancePrec(homepage,Π)=$\frac{2049}{3000} = 0.68$.

Now let us take our RDF summary described in Table 6.5. In this table we can see

	ExpLod	Campinas et al	our algorithm
$SchemaPrec_{PropertyAll}$	0.81	0.81	1

(a) Schema Precision at Property level

	ExpLod	Campinas et al	our algorithm
$SchemaRec_{PropertyAll}$	1	1	0.92

(b) Schema Recall at property level

Table 6.7 – Schema Metrics at Property level

	ExpLod	Campinas et al	SemSum+
$InstancePrec(name, \Pi)$	1	1	1
$InstancePrec(img, \Pi)$	1	0.95	0.95
$InstancePrec(homepage, \Pi)$	1	0.68	0.68
$InstancePrec(made, \Pi)$	1	1	1
$InstancePrec(biography, \Pi)$	1	0.02	-
$InstancePrec(discography, \Pi)$	1	0.01	-
$InstancePrec(image, \Pi)$	1	1	1
$InstancePrec(title, \Pi)$	1	1	0.999
$InstancePrec(date, \Pi)$	1	1	1
$InstancePrec(maker, \Pi)$	1	1	1
$InstancePrec(track, \Pi)$	1	1	1
$InstancePrec(descrption, \Pi)$	1	1	-
$InstancePrec(track - number, \Pi)$	1	1	1
$InstancePrec(available - as, \Pi)$	1	1	1
$InstancePrec(olga, \Pi)$	1	0.0001	-
$InstancePrec(format, \Pi)$	1	1	1
$InstancePrec_{PropertyAll}$	1	0.80	0.88

(a) Instance Precision at Property level

	ExpLod	Campinas et al	SemSum+
$InstanceRec(name, \Pi)$	1	1	1
$InstanceRec(img, \Pi)$	1	1	1
$InstanceRec(homepage, \Pi)$	1	1	1
$InstanceRec(made, \Pi)$	1	1	1
$InstanceRec(biography, \Pi)$	1	1	0
$InstanceRec(discography, \Pi)$	1	1	0
$InstanceRec(image, \Pi)$	1	1	1
$InstanceRec(title, \Pi)$	1	1	1
$InstanceRec(date, \Pi)$	1	1	1
$InstanceRec(maker, \Pi)$	1	1	1
$InstanceRec(track, \Pi)$	1	1	1
$InstanceRec(descrption, \Pi)$	1	1	0
$InstanceRec(track - number, \Pi)$	1	1	1
$InstanceRec(available - as, \Pi)$	1	1	1
$InstanceRec(olga, \Pi)$	1	1	0
$InstanceRec(format, \Pi)$	1	1	1
$InstanceRec_{PropertyAll}$	1	1	0.99

(b) Instance Recall at Property level

Table 6.8 – Instance Metrics at Property level

that only the pattern Pa1 has the homepage property. Thus $|Instance(hompage, \Pi)| =$ 3000. Hence,the InstancePrec(homepage,Π)=$\frac{2049}{3000} = 0.68$.

Following the same procedure, we can calculate the instance property precision for all the dataset properties; these results are reported in Table 6.8a.

On the other hand, the results for the class precision and recall at the instance level in this example is always equal to 1 or almost 1 (since in one case only a few class instances are missing) and thus their computation provides no further insights for this example. This is why, the corresponding tables were omitted.

6.4.1.3 Connectivity

Table 6.9 reports the connectivity metric values for the summaries produced by the three discussed algorithms. It shows that ExpLOD has a value of 6 for this metric because its summary ends up with 6 separate components while the ideal summary depicted in Figure 6.2 has exactly one connected component. This value means the ExpLOD provides a disconnected summary. The two other algorithms report a value of 1, which means that these two algorithm provide a summary as connected as the ideal one (one connected component in this case).

s	ExpLod	Campinas et al	our algorithm
Connectivity	6	1	1

Table 6.9 – Connectivity

6.5 Summary

In this chapter, we introduced a quality framework by defining a set of metrics, that can be used to comprehensively evaluate any RDF summarization algorithm that is reported in the literature. The metrics proposed are independent of the algorithm, the KB (thus the data) and the existence or not of schema information within the KB. The proposed Quality Framework captures correctly various desirable properties of the original KB. So, it accounts for:

- the conciseness of the summary by:
  - Penalizing the verboseness in the form of multiple patterns representing a single class in the ideal summary
  - Capturing the similarity of the different patterns or groups created by the summarization algorithm with the corresponding ideal summary parts, even if this similarity is not 100%
- the connectedness of the summary by:
  - Introducing a metric on the connectivity of the summary, thus prioritizing connected summaries against not so connected ones
- the comprehensibility of the summary by:
  - Covering the schema part and thus understanding how good a summary is at the structural level
  - Covering the instance part and thus understanding how good a summary is at covering the instances that are in the KB
  - Understanding how well connected the summary and thus the content of the KB is
  - Capturing subtle differences in the result summary, like the omission of just one property or the approximation over the number of instances that allows the user to really understand why and where there is a problem

- the overall quality of the summary so that it can be compared with other summaries by combining the different metrics like precision, recall, F-measure at different levels with connectedness in order to allow for the overall comparison, while the different metrics still provide a more detailed idea on where there are problems with a computed summary.

The Quality Framework for RDF Summaries provides an important understanding of the summaries and is a contribution of this thesis, already published at [120, 119].

Chapter 7

Experiments

In this chapter, we present the results of the experiments that were conducted in order to experimentally evaluate the algorithms proposed in this thesis. The goal of our experimental evaluation is threefold:

- to show that the summarization algorithms proposed in this thesis provide summaries for different and diverse RDF KBs, thus they work for any kind of KB;
- to provide a comparison with other commonly used techniques, so that to demonstrate the value that our proposal brings but also to be able to evaluate the cases or the elements where they excel;
- to validate and demonstrate experimentally the usefulness and appropriateness of our proposed quality framework.

Firstly, we provide an evaluation of our RDF graph summarization approach presented in Chapter 4 using a set of diverse real-world datasets from the LOD cloud. The diversity of the datasets can help us understand better how our approach works in different situations. Secondly, we made a big effort on validating that the proposed Quality Framework correctly captures the differences present in different summaries by evaluating three different RDF graph summarization algorithms (including our own) that work in substantially different ways over ten different and diverse datasets, showcasing that indeed the different aspects are correctly captured in terms of quality and that the results are easily matched towards the status of the KB.

The chapter is structured as follows: Section 7.1 describes the ten different datasets considered in the experiments. Section 7.2 describes the representative algorithms for the validation. Section 7.3 gives a quality evaluation of the created summaries based on ours and the two discussed approaches and using our proposed metrics described in chapter 6.

7.1 Datasets

Tables 7.1, 7.2 shows the datasets from the LOD cloud that are considered for the experiments. Where the columns of the Table 7.1 show the following information about each dataset: its name, the number of triples it contains, and the number of instances, classes, predicates, properties and attributes. The second and third columns of Table 7.2 show the class instance distribution metric which provides an indication on how instances are spread across the classes and it is defined as the standard deviation (SD) in the number of instances per class. When the number of class instances per class in a dataset is quite close then the standard deviation is small; while, when there are considerable differences, the standard deviation will be relatively large. While The last two columns of Table 7.2 show the property instance distribution metric which provides an indication on how instances are spread across the properties and it is also defined as standard deviation (SD) in the number of instances per property.

The main goal of our datasets selection is to use real-world datasets from diverse domains with different sizes (number of triples) and with different numbers of classes (and class instances) and properties (and properties instances). We are also interested in the distribution of the data which might indicate if the structure of the KB or the size of the represented knowledge could affect the quality of the generated summaries. So we have datasets from 270 thousand (Jpeel) to 263 million triples (Lobid), from one (Bank2) to 53 unique classes (LinkedMDB), from about 76 thousand(Jpeel) to about 18 million unique instances/entities and from 12 to 222 predicates. These datasets range from being very homogeneous (the Bank dataset where all subjects have the same list of attributes and properties) to being very heterogeneous (LinkedMDB where the attributes and properties are very heterogeneous across types). The diversity of the datasets can help us to understand better how the selected approaches work in different situations and thus validate that the proposed quality metrics will capture the different behaviors correctly.

7.2 Representative Algorithms used in the experiments

As we have already mentioned in chapter 3, the RDF graph summarization algorithms could be grouped into four main categories. In addition to our work presented in Chapter 4 and based on the results reported in the literature we have chosen two

Dataset	Triples	Instances	Classes	Predicates	properties	attributes
Jpeel [4]	271,369	76,229	9	26	14	12
Jamendo [3]	1,047,950	335,925	11	25	14	11
Sec [a]	1,813,135	460,446	5	12	3	9
linkedMDB [6]	6,148,121	694, 400	53	222	153	69
Bank [1]	7,348,860	200,429	1	33	0	33
Wordnet [8]	8,574,807	647,215	5	63	55	8
DBLP [2]	41,802,523	5,942,858	10	19	9	10
Linkedct [5]	49,084,152	5,364,776	30	121	44	77
Lobid [7]	263,215,517	17,854,885	24	104	40	64
DBpedia[b]	438,336,517	3,769,926	436	1894	919	975

Table 7.1 – Descriptive statistics of the datasets

[a]U.S. SEC data: http://www.govtrack.us/data/misc/sec.n3.gz
[b]http://wiki.dbpedia.org/data-set-38

Dataset	Class instance distribution		Property instance distribution	
	Mean	SD	Mean	SD
Jpeel [4]	8,449	8,289.61	9,374.48	15,988.21
Jamendo [3]	20,542	19,622.08	34,633.48	59,458.62
Sec [a]	66,861.8	41,233.64	144,041.83	63,388.13
linkedMDB [6]	13,971	37,368.26	24,758.70	80,271.76
Bank [1]	200,429	0	197,065.61	4,786.98
Wordnet [8]	129,147	69,768.22	59,947.92	113,775.88
DBLP [2]	497,153.9	971,029.76	538,837.42	805,531.71
Linkedct [5]	178,826	217,293.64	214,010.65	218,145.29
Lobid [7]	663,355.26	996,359.95	661,974.82	979,956.84
DBpedia[b]	129,248	188,372.89	86,136.66	227,632.07

Table 7.2 – Descriptive statistics of the datasets: Class and property instance distribution

[a]U.S. SEC data: http://www.govtrack.us/data/misc/sec.n3.gz
[b]http://wiki.dbpedia.org/data-set-38

of the most well performing (according to their authors) RDF graph summarization algorithms (ExpLOD [55] and Campinas et al. [26]). Our selection of these algorithms was based on specific properties and features that they demonstrate and as has already been introduced in 6.4.1: (a) they do not require the presence of RDF schema (triples) in order to work properly, (b) they work on both homogeneous and heterogeneous KBs, (c) they provide statistical information about the available data (which can be used to estimate a query's expected results' size), and (d) they provide a summary graph that is considerably smaller than the original graph.

As we also mentioned in in 6.4.1 and repeat here for completeness, we implemented the three algorithms used in the experiments hereafter ourselves. The implementations of two algorithms (ExpLOD and Campinas et al.) were not available from the original authors so we had to implement them ourselves in Java, based on the corresponding papers. We validated the implementation running tests with the datasets described in the original papers. Since we were getting the same results we are quite

confident that the implementations are correct. Given also that performance benchmarking is out of scope of this work, we did not have to deal with any kind of extreme optimizations. All the experiments ran on a Intel(R) Core(i5) Opteron 2.5 GHz server with 16 GB of RAM (of which 14 GB was assigned to the Java Virtual Machine), running Windows 7.

7.3 Experimental Evaluation

7.3.1 Evaluation Results

In this section, we discuss the quality results of the RDF graph summarization approaches covered in section 7.2, evaluated over all the datasets described in Table 7.1 for the following two cases:

- Typed Dataset: the KB contains schema information, like definition of classes and properties and more importantly a significant number of instances of a dataset have at least one typeof link/property.
- Untyped Dataset: there is no schema information in the KB and more importantly *none* of the datasets subjects/objects or properties has a defined type (we explicitly checked and deleted all of them).

The distinction for the experimentation is important because there are algorithms that try to exploit schema related information (mainly typeof links) in order to gain insights for the structure of the KB. While, wherever available using this information could be valuable, we would like to test the summarization algorithms in cases when this information is not available, too. With that we can validate that the proposed Quality Framework will correctly capture the differences in the results and will correctly identify, for example, algorithms that work well in both cases.

7.3.1.1 Results for schema level metrics

Table 7.3 reports the precision, recall and F-Measure values at the schema level for classes and properties of the generated RDF summaries over the set of datasets depicted in table 7.1 for the *typed* and *untyped* cases. The left part of Table 7.3 shows the results for the *typed* used datasets while the right part shows the results for *untyped* used datasets. The Figures 7.1 and 7.2 are representing the overall schema F-Measure and the class precision metrics values respectively, they were picked as visualization

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.46	0.63	1	1	1	0.81
Campinas et al	1	0.77	0.87	1	1	1	0.93
SemSum+	1	0.84	0.90	1	1	1	0.95

(a) Typed Jpeel

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.40	0.57	1	1	1	0.78
Campinas et al	1	0.40	0.57	1	1	1	0.78
SemSum+	1	0.66	0.79	1	1	1	0.89

(b) Untyped Jpeel

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.74	0.85	1	1	1	0.92
Campinas et al	1	0.83	0.90	1	1	1	0.95
SemSum+	1	0.92	0.95	1	1	1	0.97

(c) Typed Jamendo

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.60	0.75	1	1	1	0.87
Campinas et al	1	0.60	0.75	1	1	1	0.87
SemSum+	1	0.79	0.88	1	1	1	0.94

(d) Untyped Jamendo

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.21	0.34	1	1	1	0.67
Campinas et al	1	0.28	0.43	1	1	1	0.71
SemSum+	1	0.53	0.69	1	1	1	0.84

(e) Typed Sec

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.58	0.73	1	1	1	0.86
Campinas et al	1	0.58	0.73	1	1	1	0.86
SemSum+	1	0.83	0.90	1	1	1	0.95

(f) Untyped Sec

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.03	0.05	1	1	1	0.52
Campinas et al	1	1	1	1	1	1	1
SemSum+	1	1	1	1	1	1	1

(g) Typed Bank

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.03	0.05	1	1	1	0.52
Campinas et al	1	0.03	0.05	1	1	1	0.52
SemSum+	1	1	1	1	1	1	1

(h) Untype Bank

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.28	0.43	1	1	1	0.71
Campinas et al	1	0.33	0.49	1	1	1	0.74
SemSum+	1	0.87	0.93	1	1	1	0.96

(i) Typed LinkedMDB

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.20	0.33	1	1	1	0.66
Campinas et al	1	0.20	0.33	1	1	1	0.66
SemSum+	1	0.80	0.89	1	1	1	0.94

(j) Untyped LinkedMDB

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.27	0.42	1	1	1	0.71
Campinas et al	1	0.80	0.88	1	1	1	0.94
SemSum+	1	0.89	0.94	1	1	1	0.97

(k) Typed Wordnet

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.16	0.27	1	1	1	0.63
Campinas et al	1	0.16	0.27	1	1	1	0.63
SemSum+	1	0.70	0.85	1	1	1	0.92

(l) Untyped Wordnet

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.33	0.49	1	1	1	0.74
Campinas et al	1	0.73	0.84	1	1	1	0.92
SemSum+	1	0.82	0.90	1	1	1	0.96

(m) Typed DBLP

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.28	0.43	1	1	1	0.71
Campinas et al	1	0.28	0.43	1	1	1	0.71
SemSum+	1	0.66	0.79	1	1	1	0.89

(n) Untyped DBLP

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.14	0.09	1	1	1	0.54
Campinas et al	1	0.95	0.97	1	1	1	0.98
SemSum+	0.95	0.91	0.92	0.93	1	0.96	0.94

(o) Typed Linkedct

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.11	0.19	1	1	1	0.59
Campinas et al	1	0.11	0.19	1	1	1	0.59
SemSum+	1	0.75	0.85	1	1	1	0.94

(p) Untyped Linkedct

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.23	0.37	1	1	1	0.68
Campinas et al	1	0.82	0.90	1	1	1	0.95
SemSum+	1	0.85	0.91	1	1	1	0.96

(q) Typed Lobid

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	0.23	0.37	1	1	1	0.68
Campinas et al	1	0.23	0.37	1	1	1	0.68
SemSum+	1	0.80	0.87	1	1	1	0.93

(r) Untyped Lobid

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
Campinas et al	1	0.21	0.34	1	1	1	0.67
SemSum+	0.92	0.51	0.65	0.93	1	0.96	0.80

(s) Typed DBpedia

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
Campinas et al	1	0.12	0.21	1	1	1	0.60
SemSum+	0.91	0.57	0.70	0.95	1	0.97	0.83

(t) Untyped DBpedia

Table 7.3 – Precision, Recall and F-Measure at the Schema level. The R_c column reports the schema class Recall $SchemaRec_{ClassAll}$. The P_c column reports the schema class precision $SchemaPrec_{ClassAll}$. The $F1_c$ reports the schema class F-measure $SchemaF1_c$. The R_p column reports the schema property Recall $SchemaRec_{PropertyAll}$. The P_p coulmn reports the schema property precision $SchemaPrec_{PropertyAll}$. The $F1_p$ column reports the schema property F-measure $SchemaF1_p$. The F1 column reports the overall schema F-Measure $SchemaF1$

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.33	0.49	0.74
SemSum+	0.99	0.96	0.97	0.99	0.95	0.97	0.97

(a) Typed Jpeel

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	1	1	1
SemSum+	1	0.96	0.97	0.99	0.95	0.97	0.97

(b) Untyped Jpeel

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.49	0.65	0.82
SemSum+	1	0.98	0.99	1	0.98	0.99	0.99

(c) Typed Jamendo

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	1	1	1
SemSum+	1	0.98	0.99	1	0.98	0.99	0.99

(d) Untyped Jamendo

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.92	0.95	0.97
SemSum+	1	1	1	1	1	1	1

(e) Typed Sec

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	1	1	1
SemSum+	1	1	1	1	1	1	1

(f) Untyped Sec

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.97	0.98	0.99
SemSum+	1	1	1	1	0.97	0.98	0.99

(g) Typed Bank

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	1	1	1
SemSum+	1	0.93	0.96	1	0.73	0.84	0.89

(h) Untyped Bank

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.08	0.14	0.57
SemSum+	1	0.93	0.96	1	0.73	0.84	0.89

(i) Typed LinkedMDB

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	1	1	1
SemSum+	1	0.93	0.96	1	0.73	0.84	0.89

(j) Untyped LinkedMDB

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.32	0.48	0.74
SemSum+	1	0.80	0.88	1	0.82	0.90	0.89

(k) Typed Wordnet

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	1	1	1
SemSum+	1	1	1	1	1	1	1

(l) Untyped Wordnet

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.64	0.78	0.89
SemSum+	1	0.82	0.90	1	0.71	0.83	0.86

(m) Typed DBLP

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	1	1	1
SemSum+	1	1	1	1	0.79	0.88	0.94

(n) Untyped DBLP

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.79	0.88	0.94
SemSum+	1	1	1	1	0.96	0.98	0.99

(o) Typed Linkedct

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	1	1	1
SemSum+	1	0.93	0.96	1	0.73	0.84	0.89

(p) Untyped Linkedct

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	0.37	0.54	0.77
SemSum+	1	0.91	0.95	1	0.86	0.92	0.935

(q) Typed Lobid

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	$F1_p$	F1
ExpLod	1	1	1	1	1	1	1
Campinas et al	1	1	1	1	1	1	1
SemSum+	1	0.89	0.94	1	0.77	0.88	0.91

(r) Untyped Lobid

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
Campinas et	1	1	1	1	0.06	0.36	0.68
SemSum+	0.92	0.73	0.81	0.89	0.61	0.72	0.76

(s) Typed DBpedia

Algorithm	R_c	P_c	$F1_c$	R_p	P_p	F_p	F1
Campinas et al	1	1	1	1	1	1	1
SemSum+	1	0.91	0.95	1	0.86	0.92	0.93

(t) Untyped DBpedia

Table 7.4 – Precision, Recall and F-Measure at the instance level. The R_c column reports the instance class Recall $InstanceRec_{ClassAll}$. The P_c column reports the instance class precision $InstancePrec_{ClassAll}$. The $F1_c$ column reports the instance class F-measure $InstanceF1_c$. The R_p column reports the instance property Recall $InstanceRec_{PropertyAll}$. The P_p column reports the instance property precision $InstancePrec_{PropertyAll}$. The $F1_p$ column reports the instance property F-measure $InstanceF1_p$. The F1 column reports the overall instance F-Measure $InstanceF1$

examples from the different computed metrics because visualized as charts they offer more details.

We can note from Table 7.3 that the schema property recall, schema property precision and the schema property F-Measure, reported in columns R_p, P_p and F_p respectively, are always equal to 1 for the ExpLOD and the Campinas et al algorithms over all the presented datasets. The same is true for the schema class recall reported in column R_c. We can also note from the right part of the 7.3 that the values of the previously mentioned measures are equal to 1. This is because the ExpLOD and Campinas et al algorithms depend on the notion of the forward bisimulation that groups the original nodes based on classes and/or predicates, hence they are no missed properties or types (and of course nothing new is added), thus the schema class recall values will be always 1 for the ExpLOD and the Campinas et al. A predicates-based grouping is necessary for the Campinas et al algorithm when the entities' nodes do not have a class definition, hence they are no missed properties for the *untyped* case, which explains why the values for these measures have not changed for the *untyped* datasets. This also explains why we have the same measures' values for the ExpLOD and Campinas et al for the *untyped* datasets. For our RDF summarization algorithm, although it depends on the approximation type selected, if we exclude the *linkedct* dataset the values for measures mentioned previously are also equal to 1 for the *typed* and *untyped* datasets, which means that we successfully summarize the KBs, despite the fact that by construction the algorithm uses *approximate* pattern mining to detect the classes and properties available and thus some could have been possibly missed.

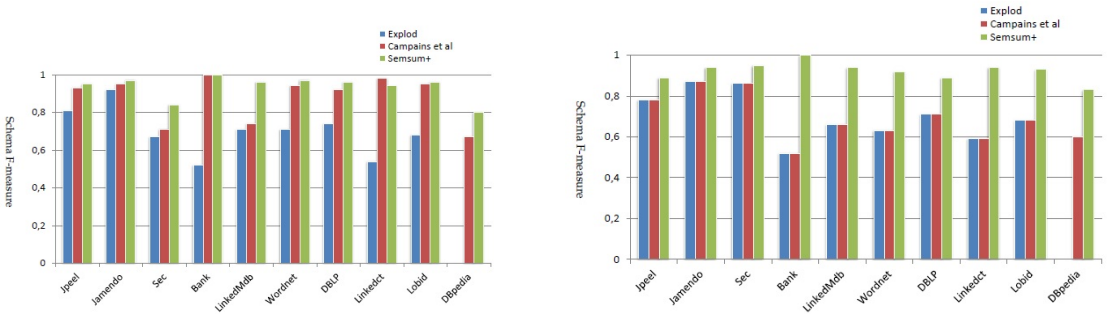
Another notable observation from the Table 7.3g and the Figure 7.1b, is that for the Bank dataset and for the overall schema F-Measure the perfect value (equals to 1) is reported for our algorithm and for Campinas et al algorithm. This is because the Bank dataset is a fully *typed* and homogeneous dataset (each subject of this dataset has at least one typeof link/property) and as we explained earlier, the Campinas et al algorithm groups the original nodes based only on their types when types exist, hence they are no missed or added properties in this case.

For the Sec dataset, the table 7.3e shows that the values of schema class precision reported in column P_c and depicted in Figure 7.2a are low for the three discussed algorithms. This is because that the ground truth schema of the Sec dataset contains a lot of inheritance relationships and as none of three algorithms deals explicitly with inheritance, the three algorithms end up with a lot of overlapping patterns (some properties which belong to the subclasses are assigned to the patterns which represent

the superclasses). Even though, we can easily note that the value of this measure for our algorithm is twice the value for the other two algorithms.

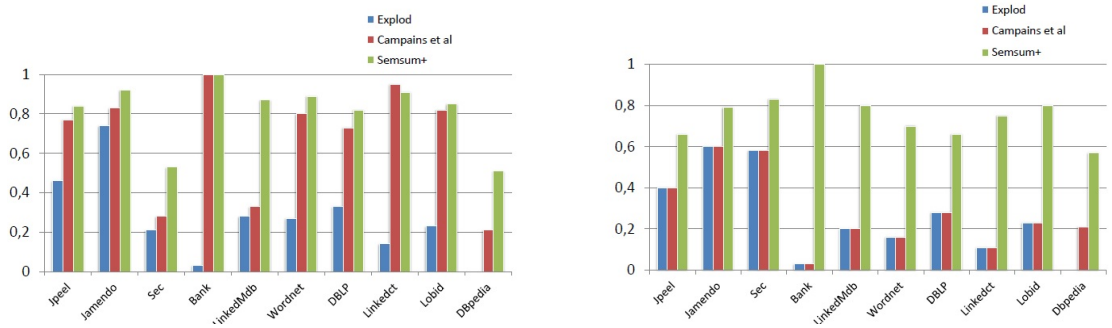
Tables 7.3s, 7.3t report metrics' values at the schema level for our summary and the generated RDF summary of the Campinas et al over the DBpedia dataset for the *typed* and *untyped* cases. We do not report results for the ExpLOD algorithm because ExpLOD's implementation was bound to datasets that fit in main memory and DBpedia could not fit in main memory. We notice from these tables that the values of the schema class precision reported in column P_c are low for the two summaries. This is because the DBpedia KB contains a lot of entities/resources having multiple classes/types and a lot of classes carry subsumption (inheritance) relationships. Actually, on average an entity has four types associated with it, and as apparently none of the two mentioned algorithms deals adequately with multiple classification, the two algorithms end up with a lot of overlapping patterns (some properties which belong to class A are assigned to the patterns which represent class B in the multiple classification case or some properties which belong to the subclasses are assigned to the patterns which represent the superclasses in inheritance case). An additional reason to have a poorer precision for the Campinas et al summary is that the type definitions are missing of a quite large number of DBpedia KB's instances. As already discussed, in this case, the Campinas et al groups the nodes based on the properties and this makes it generate a summary where a lot of the ideal summary classes are represented by several knowledge patterns. To summarize, for the DBpedia the difference of the precision values for the typed and the untyped cases between our algorithm and Campinas et al is noticable (almost twice).

Table 7.3 shows well that algorithms like ExpLOD do not provide quality summaries in extreme cases like the Bank dataset (where we have only one class) or in heterogeneous datasets like LinkedMDB, Linkedct and DBLP, where they report very low class precision values, because instances of the same class in these cases have quite different properties and they cannot be grouped together by ExpLod. This is because the ExpLod algorithm depends on the notion of the forward bisimulation [51] that groups the original nodes based on the existence of common typeof and property links. In other words, two nodes v and u are bisimilar and will end-up in the same equivalent class (pattern) if they have *exactly* the same set of types and properties. Thus, it might generate a summary where many ideal summary classes are represented by several knowledge patterns. For example, in the Bank dataset case, which contains only one class in the ideal summary, ExpLOD generated 79 knowledge patterns. And, as we already mentioned in section 6.2.1, we have included in our framework a way



(a) Typed datasets (b) Untyped datasets

Figure 7.1 – F-Measure results for typed/untyped presented datasets at the schema Level



(a) Typed datasets (b) Untyped Datasets

Figure 7.2 – Class precision results for typed/untyped presented datasets at the schema Level

to penalize these cases by introducing the W(c) exponential function (see equation 6.7).

Table 7.3 and Figures 7.2 and 7.1 also demonstrate that our algorithm gives better results, when compared with the other two algorithms, over all the presented datasets, and it shows cases that it works well with heterogeneous datasets like the LinkedMdb, unlike the ExpLod and Campinas et al that give a low class precision with the heterogeneous datasets.

By comparing the results for the *typed* datasets case depicted in Figure 7.2a and the *untyped* datasets depicted in Figure 7.2b. We can easily observe that the behavior of our algorithm and of the ExpLOD algorithm in the case of the *untyped* cases is the same as in the case of the *typed* datasets, which means that the quality of the summary is not affected by the presence (or not) of schema information in the KB. While we can easily observe the significant impact the absence of typeof schema information had for the Campinas et al algorithm.

Dataset	ExpLod	Campinas et al	Zneika et al
Jpeel	25	1	1
Jamendo	31	1	1
Sec	6	1	1
LinkedMDB	8464	1	1
Bank	11	1	1
Wordnet	778	1	1
DBLP	108	1	1
Linkedct	5699	1	1
Lobid	9786	1	1

Table 7.5 – Connectivity Metric results

As the discussion showed, the distinction of our algorithm for the different measures, it provides some insights on how we can use the proposed Quality Framework to assess the quality of the summaries produced by the different algorithms. Since we are looking at comparing the quality of the computed summary to a ground truth summary provided by an expert in general we can observe that:

- the summarization algorithms usually capture correctly the properties involved in the data but miss at different levels (and for different reasons) some of the classes. The Quality Framework provides enough resolution to clearly identify the algorithms that provide a better summary in turn of the classes reported and the quality of this report (e.g. are all properties reported, is the class present as one entity in the computed summary, etc.).

- the summarization algorithms do not capture well cases where the data (instances) are multiply classified or where there are quite widespread subsumption relationships.

- the summarization algorithms could have quite a few differences when reporting on the contents of the KB and the quality of the summaries could greatly vary and this is mostly because of the differences in the precision of reporting the classes in the summary, including penalizing verbose descriptions (like those reported by Explod). So actually we can capture even fine differences where for example a single class in the ground truth is represented by two in the computed summary.

7.3.1.2 Results for instance level metrics

Table 7.4 reports the precision, the recall and the F-Measure of RDF summaries at the instance level, based on the same datasets and algorithms as before. The left part of Table 7.3 shows the results for the *typed* datasets while the right part shows the results

for *untyped* datasets. For each dataset, we report the precision, the recall and the F-measure values at class and property level. We note that ExpLOD produces the best results (actually perfect ones, always 1) since it is not missing any property or class instance because ExpLOD works by grouping even two instances if they have the same set of attributes and types, thus does not add any false positives. We can also note that the instance class precision and the instance recall precision reported in columns P_c and R_c are always equal to 1 for Campinas et al algorithm over all the presented datasets, while the property instance precision reported in column P_p is low in most presented datasets. This is because the Campinas et al algorithm works by grouping two instances if they have the same set types, thus it does not add any false positives at the class level but maybe it will assign some properties to subjects/instances which do not actually have these properties at the KB (false positive at the property level). This explain why it is important to take into consideration quality metrics at the property and class levels.

Table 7.4 shows also that the behavior of SemSum+ and ExpLOD algorithms in the case of the *untyped* datasets is the same or approximately the same as in the case of the *typed* datasets, which means that the quality of the summary with regard to the coverage of the instances is not affected by the presence (or not) of schema information in the KB for these two algorithms. On the other hand, we can easily observe the great positive impact left by the absence of typeof schema information for the Campinas et al algorithm.

Also, tables 7.4s, 7.4t report metrics values at the instance level for the generated RDF summaries of the Campinas et al algorithm and ours over the DBpedia dataset for the *typed* and *untyped* cases respectively. From the table 7.4t, we can note that the Campinas et al algorithm produces the perfect results since it is not missing any property or class instance because for the untyped case, Campinas et al works by grouping instances if they have the same set of properties, regardless of how many they are; thus does not add any false positives. On the contrary, the table 7.4s shows that Campinas et al produces a very poor value for the instance level property precision reported in column P_p because with the presence of the class definition for the entities in the KB, works by grouping instances based only on the types they carry and ignores, e.g., how many they are. Thus with a very heterogeneous KB like DBpedia, the Campinas el al algorithm ends up with a lot of extra property instances since for all the properties the same number of property instances is assumed, since the algorithm looks only at the type information.

From this discussion, we can observe that the summarization algorithms provide results of good quality concerning the coverage of the instances in the KB. The proposed quality metrics clearly show that relying only on this metric is not adequate to judge the quality of a summary since a lot of the algorithms report perfect scores in all measures. But still we have cases where we can distinguish the quality among the results based on the instances covered by the computed summary, especially when algorithms use approximative methods to compute the summary (one algorithm in our case). It is worth noting here that our Quality Framework can capture both under-coverage (when not all instances are represented in the final result) and over-coverage (when some instances are represented more than once or some fictitious instances are included) of instances. With the metrics at the instance level we can capture these fine differences for covering correctly or not and how much the instances in the KB.

7.3.1.3 Results for the Connectivity

One final metric to be considered is whether the final graph is connected or not and appears as more than one connected components. This might mean that the summarization algorithm while captures correctly the important properties and classes in the KB fails to provide at the end a connected graph. This is important because this might signify whether the summary graph is usable or not for answering, for example, SPARQL queries. Table 7.5 reports the connectivity metric values for the summaries produced by the three discussed algorithms over all the datasets described in table 7.1. It shows that the ExpLod has always high values for this metric which means it provides a disconnected summary, while our algorithm and Campinas et al algorithm have always 1, which means that these two algorithms provide a connected summary (at least as connected as the ideal summary; fully connected in our examples).

7.3.1.4 Results combining schema- and instance-level metrics

By comparing the results in both cases, it becomes clear why it is important to take into consideration quality metrics that capture information both at the instance and the conceptual level. Otherwise behaviors like the one demonstrated by ExpLod cannot be captured and summaries that are flawed might be indistinguishable from better ones. Overall, we could argue that the Quality Framework introduced in chapter 6 is adequate for capturing the fine differences in quality of the summaries produced by the three algorithms. We can also see that with a closer look at the results we can gain or verify insights on how specific algorithms work and the quality of the summaries they produce. So measuring the quality at the schema level, the

instance level and the connected components of the graph can give us a detailed view of the strengths and weaknesses of a summary and decide whether to use it or not depending on the potential use and application. We avoided combining all the measures together because this might blur the final picture. The idea is not to necessarily prove an algorithm as better or worse (we can do this to a great extend through the different *F-measures*) but mainly to help the user understand the different qualities of the summaries and choose the best one for the different needs of the diverse use cases.

7.4 Summary

In this chapter we presented an extended experimental evaluation using a set of different and diverse datasets representing different RDF KBs. The goal that these experiments served was on the one hand to validate that the SemSum+ algorithm we proposed provides usable summaries and then to show that these summaries are better or comparable with the summaries generated of various state-of-the-art algorithms that exist in the literature. At the same time, we used the same experiments to demonstrate that the Quality Framework that we proposed for assessing the quality and compare different summaries of RDF KBs, captures adequately and with sufficient detail the differences among the computed summaries. This work was also part of the publication at [120].

Chapter 8

Conclusions

8.1 Summary of Contributions

The explosion of size, complexity and number of available RDF Knowledge Bases (KBs) and the emergence of Linked Open Data led to the necessity of providing lighter and smaller structures that would properly represent the KBs, while at the same time will be easy to query or visualize. This introduces the need to have algorithms that provide concise summaries of RDF KBs, namely of RDF knowledge graphs. In this thesis we introduced, proposed and evaluated experimentally such a method. More precisely:

In Chapter 4 we presented SemSum+, our RDF graph summarization method for RDF KBs, which is based on representing the RDF graph using the (best) top-k approximate RDF graph patterns. It extracts from the RDF graph, an RDF summary that describes the actual contents of the KB, which is not necessarily the complete schema of the KB but the used/active schema of the KB, usually subset of the original full schema. Some statistical information (number of class and property instances per pattern) is included in our summary graph, which allows us to estimate a query's expected results' size. The proposed algorithm carries all the desired properties, as they were described in the introduction. It produces a summary that is always a valid RDF/S graph, does not require the presence of a schema (but takes advantage the schema information if present) and works equally well with homogeneous and heterogeneous KBs. We evaluated experimentally our method and we showed that SemSum+ works equally well with the presence or absence of schema, on homogeneous and heterogeneous RDF KBs and always produces an RDF graph as a summary. This constitutes the first contribution of this thesis.

While developing the experimental evaluation of SemSum+, we realized that having only a memory based algorithm will not be sufficient in cases of summarizing

large KBs, meaning KBs that do not fit in memory, which is not an extreme case anymore. Thus, we looked for a scalable solution and we proposed a parallelization method that can be actually used by any approximate pattern mining algorithm if it uses a binary matrix for the input data and a cost function to heuristically decide which patterns to preserve and when to stop. We proposed such a scalable solution in Chapter 5 and we used this to allow the calculation of summaries of KBs regardless of size. And this constitutes the second contribution of this thesis.

By looking at the literature, we also realized that while there are many algorithms proposed in the literature for RDF graph summarization, there was no way or even any comprehensive discussion on how we could evaluate those different summaries and understand their different quality characteristics. Overall this means that we need a way to comprehensively understand (and possibly measure) the quality of the produced summaries. And we need to be able to do this in a way that is independent of the algorithm, the status of the KB (homogeneous or heterogeneous), the contents of the RDF KB and the summary itself. We introduced our comprehensive Quality Framework for RDF Graph Summarization in Chapter 6 and used our experiments in Chapter 7 to evaluate the quality of various summaries produced by different algorithms over diverse RDF KBs. Through the experiments we demonstrated that the proposed metrics, that operate at the schema and the instance level, manage to capture subtle differences between the algorithms and at all levels of detail. Given also that the experimental evaluation included diverse and different datasets, we also demonstrated that the Quality Framework captures the different quality aspects in the whole spectrum of RDF data. And this constitutes the third contribution of this thesis.

So in this thesis, we had the opportunity to tackle three important problems in the area of RDF graph summarization, which are: how do we compute representative semantic summaries from RDF KBs, how do we make this solutions scalable so that they work over large knowledge bases and how do we evaluate the results of such efforts.

8.2 Future work

Despite the progress done during this thesis, there are still interesting problems that can be tackled and possibly extend our work. In the area of graph summarization methods, we would like to extend our SemSum+ method to take into account:

- hierarchical (subsumption) relationships like *subClassOf* and *subPropertyOf*; those properties are easy to identify when having schema information but much more difficult to understand in its absence. This is actually a known problem in the semantic web literature and part of the many of the schema extraction methods proposed. The main difference here is that we do not try to extract the whole schema, which complicates things. Imagine the case when the summary might not finally contain the superclass and while we could identify a subsumption relationship, we might not be able finally to include it in the summary.
- the importance of the linking relationships like e.g. *sameAs*. Statistical methods tend to ignore the relationships that are not statistically significant but the important semantics of the linking relationships should be taken into account when creating the summary.

In the area of the quality evaluation of the produced summaries, there is also a need to extend the quality metrics to take into account the aforementioned two points: subsumption and linking relationships. This would allow our Quality Framework to compute additional quality aspects for the summary. Furthermore, we would like to experiment with additional datasets, maybe beyond the RDF world, since the Quality Framework could be used to evaluate other types of summaries, e.g. based on OWL.

Moving back to the summarization discussion, one aspect almost completely missing in the literature is how to *update* the produced RDF graph summary. It is quite common nowadays that the RDF KBs get updated, more at the instance level but also at the schema level (although admittedly less frequently). The research question that rises from this point is whether we can update the summary without having to recompute it from the beginning or at least to identify the turning point (if any) when the computed summary becomes obsolete and a new one must be computed. This is a quite complex problem, since the summary is by definition an approximation and we need to define when them approximation is not good/representative enough for the KB.

An additional aspect that can be investigated is the ability to provide personalized summaries, depending on various characteristics that might interest the user, like the size the complexity, the extend (coverage) of the summary and so on. In SemSum+ this can be done if we are able to express as a cost function the different aspects the concern the user and then use this cost function to substitute the original one used in the algorithm. This is not trivial at all, starting from the mere capacity of one to construct such a function.

From an application perspective, we would also like to be able to use the our summarization method in a real-world use cases and test how we could facilitate and improve *federated query evaluation* or evaluation over linked datasets and evaluate the feasibility of querying the summary only or in combination with additional information from the KB. Of course, real-world applications might also require the ability to update the extracted RDF graph summary.

As said in the introduction as well, the problem of reducing large datasets in a way that we retain their main attributes and semantics, but still making possible to understand and query them, is a very interesting problem in the area of Big Data. This problem is tightly associated with the problem of using approximation to avoid extensive searching and providing approximate answers instead of all the details that might not be so interesting in the specific context. In any case, either by providing summaries from our data or by approximating the answers we provide, we work towards solving the problem of dealing with large and difficult to manage datasets.

Bibliography

[1] About World Bank Linked Data: World Bank Finances. http://worldbank.270a.info/about.html#about-datasets/. Accessed: 2017-03-30.

[2] DBLP Bibliography Database in RDF Datahub. https://datahub.io/dataset/fu-berlin-dblp. Accessed: 2017-03-30.

[3] Jamendo DBTune home,. http://dbtune.org/jamendo. Accessed: 2017-03-30.

[4] John Peel DBTune home,. http://dbtune.org/bbc/peel. Accessed: 2017-03-30.

[5] LinkedCT Datahub. https://datahub.io/dataset/linkedct/. Accessed: 2017-03-30.

[6] LinkedMDB home,. http://www.linkedmdb.org/. Accessed: 2017-03-30.

[7] lobid-Bibliographic Resources. https://datahub.io/dataset/lobid-resources. Accessed: 2017-03-30.

[8] WordNet RDF home. http://wordnet-rdf.princeton.edu. Accessed: 2017-03-30.

[9] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Acm sigmod record*, volume 22, pages 207–216. ACM, 1993.

[10] Mir Sadek Ali, Mariano P Consens, Shahan Khatchadourian, and Flavio Rizzolo. Describex: interacting with axpre summaries. In *2008 IEEE 24th International Conference on Data Engineering*, pages 1540–1543. IEEE, 2008.

[11] Mir Sadek Ali, Mariano P Consens, and Birger Larsen. Representing user navigation in xml retrieval with structural summaries. In *European Conference on Information Retrieval*, pages 719–723. Springer, 2009.

[12] Anas Alzogbi and Georg Lausen. Similar structures inside rdf-graphs. In *LDOW*, 2013.

[13] Anas Alzogbi and Georg Lausen. Similar structures inside rdf-graphs. In *Proceedings of the WWW2013 Workshop on Linked Data on the Web, Rio de Janeiro, Brazil, 14 May, 2013*, 2013.

[14] Samur Araujo, Jan Hidders, Arjen P de Vries, and Daniel Schwabe. Serimi: resource description similarity, rdf instance matching and interlinking. In *Proceedings of the 6th International Conference on Ontology Matching-Volume 814*, pages 246–247. CEUR-WS. org, 2011.

[15] Nikolaos Athanasis, Vassilis Christophides, and Dimitris Kotzinos. Generating on the fly queries for the semantic web: The ICS-FORTH graphical RQL interface (GRQL). In *The Semantic Web - ISWC 2004: Third International Semantic Web Conference,Hiroshima, Japan, November 7-11, 2004. Proceedings*, pages 486–501, 2004.

[16] Cosmin Basca and Abraham Bernstein. Avalanche: Putting the spirit of the web back into semantic web querying. In *Proceedings of the 2010 International Conference on Posters & Demonstrations Track-Volume 658*, pages 177–180. CEUR-WS. org, 2010.

[17] T. J. Berners-Lee, R. Cailliau, and J.-F. Groff. The world-wide web. *Comput. Netw. ISDN Syst.*, 25(4-5):454–459, November 1992.

[18] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far. *International journal on semantic web and information systems*, 5(3):1–22, 2009.

[19] Stefan Blom and Simona Orzan. A distributed algorithm for strong bisimulation reduction of state spaces. *International Journal on Software Tools for Technology Transfer (STTT)*, 7(1):74–86, 2005.

[20] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[21] Paolo Boldi and Sebastiano Vigna. Axioms for centrality. *Internet Mathematics*, 10(3-4):222–262, 2014.

[22] Tim Bray, Dave Hollander, and Andrew Layman. Namespaces in xml. *World Wide Web Consortium Recommendation REC-xml-names-19990114. http://www. w3. org/TR/1999/REC-xml-names-19990114*, 1999.

[23] Dan Brickley, Ramanathan V Guha, and Brian McBride. Rdf schema 1.1. *W3C recommendation*, 25:2004–2014, 2014.

[24] Damian Bursztyn, François Goasdoué, and Ioana Manolescu. Efficient query answering in dl-lite through FOL reformulation (extended abstract). In *Proceedings of the 28th International Workshop on Description Logics, Athens,Greece, June 7-10, 2015.*, 2015.

[25] Stéphane Campinas, Renaud Delbru, and Giovanni Tummarello. Efficiency and precision trade-offs in graph summary algorithms. In *Proceedings of the 17th International Database Engineering & Applications Symposium*, pages 38–47. ACM, 2013.

[26] Stephane Campinas, Thomas E Perry, Diego Ceccarelli, Renaud Delbru, and Giovanni Tummarello. Introducing rdf graph summary with application to assisted sparql formulation. In *Database and Expert Systems Applications (DEXA), 2012 23rd International Workshop on*, pages 261–266. IEEE, 2012.

[27] Šejla Čebirić, François Goasdoué, Haridimos Kondylakis, Dimitris Kotzinos, Ioana Manolescu, Georgia Troullinou, and Mussab Zneika. Summarizing semantic graphs: a survey. *The VLDB Journal*, pages 1–33, Dec 2018.

[28] Šejla Čebirić, François Goasdoué, and Ioana Manolescu. Query-oriented summarization of RDF graphs. *PVLDB*, 8(12):2012–2015, 2015.

[29] Šejla Čebirić, François Goasdoué, and Ioana Manolescu. Query-oriented summarization of RDF graphs. In *BICOD*, 2015.

[30] Šejla Čebirić, François Goasdoué, and Ioana Manolescu. Query-Oriented Summarization of RDF Graphs. Research Report RR-8920, INRIA Saclay ; Université Rennes 1, June 2017. https://hal.inria.fr/hal-01325900v4.

[31] Gong Cheng, Cheng Jin, and Yuzhong Qu. HIEDS: A generic and efficient approach to hierarchical dataset summarization. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 3705–3711, 2016.

[32] Mariano P Consens, Valeria Fionda, Shahan Khatchadourian, and Giuseppe Pirro. S+ epps: construct and explore bisimulation summaries, plus optimize navigational queries; all on existing sparql systems. *Proceedings of the VLDB Endowment*, 8(12):2028–2031, 2015.

[33] Mariano P Consens, Flavio Rizzolo, and Alejandro A Vaisman. Axpre summaries: Exploring the (semi-) structure of xml web collections. In *ICDE*, volume 8, pages 1519–1521, 2008.

[34] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[35] Marek Dudás, Vojtech Svátek, and Jindrich Mynarz. Dataset summary visualization with lodsight. In *The Semantic Web: ESWC 2015 Satellite Events - ESWC 2015 Satellite Events Portorož, Slovenia, May 31 - June 4, 2015, Revised Selected Papers*, pages 36–40, 2015.

[36] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[37] Lorena Etcheverry and Mariano P Consens. Summary-based comparison of data quality across public mage-ml genomic datasets. *Journal of Information and Data Management*, 2(1):3, 2011.

[38] Wenfei Fan, Jianzhong Li, Xin Wang, and Yinghui Wu. Query preserving graph compression. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 157–168. ACM, 2012.

[39] Birte Glimm, Yevgeny Kazakov, Thorsten Liebig, Trung-Kien Tran, and Vincent Vialard. Abstraction refinement for ontology materialization. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference*, pages 180–195, 2014.

[40] François Goasdoué, Konstantinos Karanasos, Julien Leblay, and Ioana Manolescu. View selection in semantic web databases. *PVLDB*, 5(2), 2011.

[41] François Goasdoué, Ioana Manolescu, and Alexandra Roatiş. Efficient query answering against dynamic RDF databases. In *EDBT*, 2013.

[42] Roy Goldman and Jennifer Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 436–445, 1997.

[43] Jeremy J. Carroll Graham Klyne. Resource Description Framework (RDF): Concepts and Abstract Syntax. `https://www.w3.org/TR/rdf-concepts/`, 2004. [Online; accessed 27-May-2008].

[44] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data mining and knowledge discovery*, 8:53–87, 2004.

[45] Monika Rauch Henzinger, Thomas A. Henzinger, and Peter W. Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, 1995.

[46] Katja Hose and Ralf Schenkel. Towards benefit-based RDF source selection for SPARQL queries. In *Proceedings of the 4th International Workshop on Semantic Web Information Management, SWIM 2012, Scottsdale, AZ, USA, May 20, 2012*, page 2, 2012.

[47] Katja Hose and Ralf Schenkel. Towards benefit-based rdf source selection for sparql queries. In *Proceedings of the 4th International Workshop on Semantic Web Information Management*, page 2. ACM, 2012.

[48] Xiaowei Jiang, Xiang Zhang, Feifei Gao, Chunan Pu, and Peng Wang. Graph compression strategies for instance-focused semantic mining. In *Linked Data and Knowledge Graph - 7th Chinese Semantic Web Symposium and 2nd Chinese Web Science Conference, CSWS 2013, Shanghai, China, August 12-16, 2013. Revised Selected Papers*, pages 50–61, 2013.

[49] Amit Krishna Joshi, Pascal Hitzler, and Guozhu Dong. Towards logical linked data compression. In *Proceedings of the Joint Workshop on Large and Heterogeneous Data and Quantitative Formalization in the Semantic Web, LHD+ SemQuant2012, at the 11th International Semantic Web Conference, ISWC2012*. Citeseer, 2012.

[50] Amit Krishna Joshi, Pascal Hitzler, and Guozhu Dong. Logical linked data compression. In *Extended Semantic Web Conference*, pages 170–184. Springer, 2013.

[51] Raghav Kaushik, Philip Bohannon, Jeffrey F Naughton, and Henry F Korth. Covering indexes for branching path queries. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 133–144. ACM, 2002.

[52] Kenza Kellou-Menouer and Zoubida Kedad. Discovering types in rdf datasets. In *International Semantic Web Conference*, pages 77–81. Springer, 2015.

[53] Kenza Kellou-Menouer and Zoubida Kedad. Schema discovery in RDF data sources. In *ER*, 2015.

[54] Kifayat Ullah Khan, Waqas Nawaz, and Young-Koo Lee. Set-based approximate approach for lossless graph summarization. *Computing*, 97(12):1185–1207, 2015.

[55] Shahan Khatchadourian and Mariano Consens. Explod: Summary-based exploration of interlinking and rdf usage in the linked open data cloud. *The Semantic Web: Research and Applications*, pages 272–287, 2010.

[56] Shahan Khatchadourian and Mariano P Consens. Exploring rdf usage and interlinking in the linked open data cloud using explod. In *LDOW*, 2010.

[57] Shahan Khatchadourian and Mariano P Consens. Understanding billions of triples with usage summaries. *Semantic Web Challenge*, 2011.

[58] Shahan Khatchadourian and Mariano P Consens. Constructing bisimulation summaries on a multi-core graph processing framework. In *Proceedings of the GRADES'15*, page 8. ACM, 2015.

[59] Mathias Konrath, Thomas Gottron, and Ansgar Scherp. Schemex–web-scale indexed schema extraction of linked open data. *Semantic Web Challenge, Submission to the Billion Triple Track*, pages 52–58, 2011.

[60] Mathias Konrath, Thomas Gottron, Steffen Staab, and Ansgar Scherp. Schemex–efficient construction of a data catalogue by stream-based indexing of linked data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 16:52–58, 2012.

[61] Aapo Kyrola, Guy E Blelloch, Carlos Guestrin, et al. Graphchi: Large-scale graph computation on just a pc. In *OSDI*, volume 12, pages 31–46, 2012.

[62] Davide Lanti, Martin Rezk, Guohui Xiao, and Diego Calvanese. The NPD benchmark: Reality check for OBDA systems. In *EDBT*, 2015.

[63] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014.

[64] Amine Louati, Marie-Aude Aufaure, Yves Lechevallier, and France Chatenay-Malabry. Graph aggregation: Application to social networks. In *HDSDA*, pages 157–177, 2011.

[65] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. Mining top-k patterns from binary datasets in presence of noise. In *SDM*, pages 165–176. SIAM, 2010.

[66] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. A unifying framework for mining approximate top-k binary patterns. *IEEE TKDE*, 26:2900–2913, 2014.

[67] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. Supervised evaluation of top-k itemset mining algorithms. In *Big Data Analytics and Knowledge Discovery*, pages 82–94. Springer, 2015.

[68] P. Miettinen, T. Mielikainen, A. Gionis, G. Das, and H. Mannila. The discrete basis problem. *IEEE TKDE*, 20(10):1348–1362, Oct. 2008.

[69] Pauli Miettinen and Jilles Vreeken. Model order selection for boolean matrix factorization. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 51–59, 2011.

[70] Tova Milo and Dan Suciu. Index structures for path expressions. In *International Conference on Database Theory*, pages 277–295. Springer, 1999.

[71] Sandy Moens, Emin Aksehirli, and Bart Goethals. Frequent itemset mining for big data. In *Proceedings of the 2013 IEEE International Conference on Big Data, 6-9 October 2013, Santa Clara, CA, USA*, pages 111–118, 2013.

[72] Enrico Motta, Paul Mulholland, Silvio Peroni, Mathieu d'Aquin, José Manuél Gómez-Pérez, Victor Mendez, and Fouad Zablith. A novel approach to visualizing and navigating ontologies. In *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, pages 470–486, 2011.

[73] Jindrich Mynarz, Marek Dudás, Paolo Tomeo, and Vojtech Svátek. Generating examples of paths summarizing RDF datasets. In *Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems (SEMANTiCS2016)*, 2016.

[74] Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. Graph summarization with bounded error. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 419–432. ACM, 2008.

[75] Robert Paige and Robert E Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[76] Matteo Palmonari, Anisa Rula, Riccardo Porrini, Andrea Maurino, Blerina Spahiu, and Vincenzo Ferme. ABSTAT: linked data summaries with abstraction and statistics. In *ESWC Workshops*, 2015.

[77] Jeff Z Pan, José Manuel Gómez Pérez, Yuan Ren, Honghan Wu, Haofen Wang, and Man Zhu. Graph pattern based rdf data compression. In *Semantic Technology*, pages 239–256. Springer, 2014.

[78] Alexandros Pappas, Georgia Troullinou, Giannis Roussakis, Haridimos Kondylakis, and Dimitris Plexousakis. Exploring importance measures for summarizing RDF/S kbs. In *The Semantic Web - 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 - June 1, 2017, Proceedings, Part I*, pages 387–403, 2017.

[79] Heiko Paulheim and Christian Bizer. Type inference on noisy RDF data. In *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, pages 510–525, 2013.

[80] Silvio Peroni, Enrico Motta, and Mathieu d'Aquin. Identifying key concepts in an ontology, through the integration of cognitive principles with statistical and topological measures. In *The Semantic Web, 3rd Asian Semantic Web Conference, ASWC 2008, Bangkok, Thailand, December 8-11, 2008. Proceedings*, pages 242–256, 2008.

[81] François Picalausa, Yongming Luo, George H. L. Fletcher, Jan Hidders, and Stijn Vansummeren. A structural approach to indexing triples. In *ESWC*, 2012.

[82] Carlos Eduardo Pires, Paulo Sousa, Zoubida Kedad, and Ana Carolina Salgado. Summarizing ontology-based schemas in pdms. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 239–244. IEEE, 2010.

[83] Carlos Eduardo S. Pires, Paulo Orlando Queiroz-Sousa, Zoubida Kedad, and Ana Carolina Salgado. Summarizing ontology-based schemas in PDMS. In *Workshops Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, pages 239–244, 2010.

[84] Valentina Presutti, Lora Aroyo, Alessandro Adamou, Balthasar A. C. Schopman, Aldo Gangemi, and Guus Schreiber. Extracting core knowledge from linked data. In *Proceedings of the Second International Workshop on Consuming Linked Data (COLD2011), Bonn, Germany, October 23, 2011*, 2011.

[85] Paulo Orlando Queiroz-Sousa, Ana Carolina Salgado, and Carlos Eduardo S. Pires. A method for building personalized ontology summaries. *JIDM*, 4(3):236–250, 2013.

[86] Matteo Riondato, Justin A. DeBrabant, Rodrigo Fonseca, and Eli Upfal. PARMA: a parallel randomized algorithm for approximate association rules mining in mapreduce. In *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pages 85–94, 2012.

[87] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[88] Alexander Schätzle, Antony Neu, Georg Lausen, and Martin Przyjaciel-Zablocki. Large-scale bisimulation of rdf graphs. In *Proceedings of the Fifth Workshop on Semantic Web Information Management*, page 1. ACM, 2013.

[89] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *International Semantic Web Conference*, pages 601–616. Springer, 2011.

[90] Qi Song, Yinghui Wu, and Xin Luna Dong. Mining summaries for knowledge graph search. In *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*, pages 1215–1220, 2016.

[91] Blerina Spahiu, Riccardo Porrini, Matteo Palmonari, Anisa Rula, and Andrea Maurino. ABSTAT: ontology-driven linked data summaries with pattern minimalization. In *SumPre*, 2016.

[92] Blerina Spahiu, Riccardo Porrini, Matteo Palmonari, Anisa Rula, and Andrea Maurino. ABSTAT: ontology-driven linked data summaries with pattern minimalization. In *The Semantic Web - ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 - June 2, 2016, Revised Selected Papers*, pages 381–395, 2016.

[93] Giorgio Stefanoni, Boris Motik, and Egor V. Kostylev. Estimating the Cardinality of Conjunctive Queries over RDF Data Using Graph Summarisation. Research report, University of Oxford, 2017.

[94] Giorgio Stefanoni, Boris Motik, and Egor V Kostylev. Estimating the cardinality of conjunctive queries over rdf data using graph summarisation. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 1043–1052. International World Wide Web Conferences Steering Committee, 2018.

[95] Markus Stocker, Andy Seaborne, Abraham Bernstein, Christoph Kiefer, and Dave Reynolds. SPARQL basic graph pattern optimization using selectivity estimation. In *WWW*, 2008.

[96] Yan Sun, Kongfa Hu, Zhipeng Lu, Li Zhao, and Ling Chen. A graph summarization algorithm based on rfid logistics. *Physics Procedia*, 24:1707–1714, 2012.

[97] Yuanyuan Tian, Richard A Hankins, and Jignesh M Patel. Efficient aggregation for graph summarization. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 567–580. ACM, 2008.

[98] Yuanyuan Tian and Jignesh M Patel. Interactive graph summarization. In *Link Mining: Models, Algorithms, and Applications*, pages 389–409. Springer, 2010.

[99] Thanh Tran, Günter Ladwig, and Sebastian Rudolph. Managing structured and semistructured RDF data using structure indexes. *IEEE TKDE*, 25(9), 2013.

[100] Georgia Troullinou, Haridimos Kondylakis, Evangelia Daskalaki, and Dimitris Plexousakis. RDF digest: Efficient summarization of RDF/S kbs. In *The Semantic Web. Latest Advances and New Domains - 12th European Semantic Web Conference, ESWC 2015*, pages 119–134, 2015.

[101] Georgia Troullinou, Haridimos Kondylakis, Evangelia Daskalaki, and Dimitris Plexousakis. RDF digest: Ontology exploration using summaries. In *Proceedings of the ISWC 2015 Posters & Demonstrations Track*, 2015.

[102] Georgia Troullinou, Haridimos Kondylakis, Evangelia Daskalaki, and Dimitris Plexousakis. Ontology understanding without tears: The summarization approach. *SWJ*, 2017.

[103] Leslie G Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[104] W3C. Owl 1 web ontology language. https://www.w3.org/TR/owl-features/, 2012.

[105] W3C. Owl 2 web ontology language. https://www.w3.org/TR/owl2-overview/, 2012.

[106] Zhichun Wang. A semi-supervised learning approach for ontology matching. In *Chinese Semantic Web and Web Science Conference*, pages 17–28. Springer, 2014.

[107] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 4th edition, 2015.

[108] Gang Wu, Juanzi Li, Ling Feng, and Kehong Wang. Identifying potentially important concepts and relations in an ontology. In *The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008. Proceedings*, pages 33–49, 2008.

[109] Yang Xiang, Ruoming Jin, David Fuhry, and Feodor F. Dragan. Summarizing transactional databases with overlapped hyperrectangles. *Data Min. Knowl. Discov.*, 23(2):215–251, September 2011.

[110] Xifeng Yan, Philip S Yu, and Jiawei Han. Graph indexing: a frequent structure-based approach. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 335–346. ACM, 2004.

[111] Mohammed J. Zaki and Ching-Jui Hsiao. Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE TKDE*, 17(4):462–478, April 2005.

[112] Haiwei Zhang, Yuanyuan Duan, Xiaojie Yuan, and Ying Zhang. Assg: adaptive structural summary for rdf graph data. ISWC, 2014.

[113] Ning Zhang, Yuanyuan Tian, and Jignesh M Patel. Discovery-driven graph summarization. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 880–891. IEEE, 2010.

[114] Zhigang Zhang, Genlin Ji, and Mengmeng Tang. Mreclat: An algorithm for parallel mining frequent itemsets. In *International Conference on Advanced Cloud and Big Data, CBD 2013, Nanjing, China, December 13-15, 2013*, pages 177–180, 2013.

[115] Peixiang Zhao, Jeffrey Xu Yu, and Philip S Yu. Graph indexing: tree+ delta¡= graph. In *Proceedings of the 33rd international conference on Very large data bases*, pages 938–949. VLDB Endowment, 2007.

[116] Weiguo Zheng, Lei Zou, Wei Peng, Xifeng Yan, Shaoxu Song, and Dongyan Zhao. Semantic SPARQL similarity search over RDF knowledge graphs. *PVLDB*, 9(11):840–851, 2016.

[117] Mussab Zneika, Claudio Lucchese, Dan Vodislav, and Dimitris Kotzinos. RDF graph summarization based on approximate patterns. In *Information Search, Integration, and Personalization - 10th International Workshop, ISIP 2015, Grand Forks, ND, USA, October 1-2, 2015, Revised Selected Papers*, pages 69–87, 2015.

[118] Mussab Zneika, Claudio Lucchese, Dan Vodislav, and Dimitris Kotzinos. Summarizing linked data RDF graphs using approximate graph pattern mining. In *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016.*, pages 684–685, 2016.

[119] Mussab Zneika, Dan Vodislav, and Dimitris Kotzinos. Quality metrics for RDF graph summarization. In *34ème Conférence sur la Gestion de Données – Principes, Technologies et Applications (BDA 2018), Bucarest, 22-26 octobre 2018*.

[120] Mussab Zneika, Dan Vodislav, and Dimitris Kotzinos. Quality metrics for RDF graph summarization. *Semantic Web Journal*, 10(3):555–584, 2019.