Novel Node Importance Measures to Improve Keyword Search over RDF Graphs

Elisa Souza Menendez

Tese (Doutorado em Informática). Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2019.

Pontifícia Universidade Católica
do Rio de Janeiro

Elisa Souza Menendez

Novel Node Importance Measures to Improve Keyword Search over RDF Graphs

TESE DE DOUTORADO

Thesis presented to the Programa de Pós-Graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Ciências - Informática

Advisor: Prof. Marco Antonio Casanova

Rio de Janeiro
February 2019

Elisa Souza Menendez

Elisa Souza Menendez holds a master in computer science degree from Pontifical Catholic University of Rio de Janeiro (PUC-Rio), and a bachelor degree in Information Systems from Federal University of Sergipe (UFS). Her main research topics are Semantic Web and Information Retrieval.

Bibliographic data

Menendez, Elisa Souza

Novel Node Importance Measures to Improve Keyword Search over RDF Graphs / Elisa Souza Menendez; advisor: Marco Antonio Casanova. – 2019.

91 f. : il. ; 30 cm

Tese (Doutorado em Informática)–Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2019.

Inclui bibliografia

1. Informática – Teses. 2. Ranqueamento. 3. Busca por palavras-chave. 4. RDF. 5. SPARQL. 6. PageRank. I. Casanova, Marco Antonio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

First, the most special thank you to the best advisor I could ask for, Prof. Marco Antonio Casanova. For sure, his fully support and wisdom were key contributors in my academic achievements. I feel inspired by Prof. Casanova to become a professor myself, hoping someday, a student can admire me as much as I admire him.

I would like to thank my second advisor, Prof. Mohand Boughanem and all the team from the Institut de Recherche en Informatique de Toulouse (IRIT). They made my year in France very pleasant and culturally enriching.

I can't forget to thank the team from Tecgraf/K2, which were a key part in the early years of this project. Best regards, Yenier, Grettel, Fred and Kaka. Of course, I would like to extend my appreciation and gratitude to my classmates, professors and staff from the Department of Informatics of PUC-Rio.

This study was financed in part by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), and by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

Last, but not least important, my deep gratitude to my parents, Gracinha and Angel, for their support and encouragement during all these years of study. I love you, thanks.

Abstract

Menendez, Elisa Souza; Casanova, Marco Antonio (advisor). **Novel Node Importance Measures to Improve Keyword Search over RDF Graphs.** Rio de Janeiro, 2019. 91p. Tese de Doutorado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A key contributor to the success of keyword search systems is a ranking mechanism that considers the importance of the retrieved documents. The notion of importance in graphs is typically computed using centrality measures that highly depend on the degree of the nodes, such as PageRank. However, in RDF graphs, the notion of importance is not necessarily related to the node degree. Therefore, this thesis addresses two problems: (1) how to define importance measures for RDF graphs; (2) how to use these measures to help compile and rank results of keyword queries over RDF graphs. To solve these problems, the thesis proposes a novel family of measures, called InfoRank, and a keyword search system, called QUIRA, for RDF graphs. Finally, this thesis concludes with experiments showing that the proposed solution improves the quality of the results in two keyword search benchmarks.

Keywords

Ranking; Keyword Search; RDF; SPARQL; PageRank.

Resumo

Menendez, Elisa Souza; Casanova, Marco Antonio (orientador). **Novas Medidas de Importância de Vértices para Aperfeiçoar a Busca por Palavras-chave em Grafos RDF.** Rio de Janeiro, 2019. 91p. Tese de Doutorado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Um ponto importante para o sucesso de sistemas de busca por palavras-chave é um mecanismo de ranqueamento que considera a importância dos documentos recuperados. A noção de importância em grafos é tipicamente computada usando medidas de centralidade, que dependem amplamente do grau dos nós, como o PageRank. Porém, em grafos RDF, a noção de importância não é necessariamente relacionada com o grau do nó. Sendo assim, esta tese aborda dois problemas: (1) como definir uma medida de importância em grafos RDF; (2) como usar essas medidas para ajudar a compilar e ranquear respostas a consultas por palavras-chave sobre grafos RDF. Para resolver estes problemas, esta tese propõe uma nova família de medidas, chamada de InfoRank, e um sistema de busca por palavras-chave, chamado QUIRA, para grafos RDF. Esta tese é concluída com experimentos que mostram que a solução proposta melhora a qualidade dos resultados em benchmarks de busca por palavras-chave.

Palavras-chave

Ranqueamento; Busca por palavras-chave; RDF; SPARQL; PageRank.

Table of Contents

List of Figures

List of Tables

1
Introduction

1.1. Context and Motivation

Keyword search is a well-known and convenient way for users to query large amounts of data, whether in Web pages or databases. The user simply types some terms, called *keywords*, and it is up to the system to retrieve the documents that best match the list of keywords. Search engines for Web pages popularized this kind of search. More recently, some of the Information Retrieval techniques used by Web search engines were adapted to query databases to hide from users unfriendly SQL queries.

In the last decade, RDF emerged as a data model that represents data as a set of triples, which in turn induces a graph. This kind of modeling adds flexibility to describe resources and follows W3C standardized formats and ontologies. Considering that RDF graphs are interesting sources of knowledge that are also queried with unfriendly SPARQL queries, keyword search over RDF graphs (or briefly *RDF-KwS*) becomes a relevant research topic.

In Web Information Retrieval there are two main tasks: (1) matching keywords with indexed documents; (2) ranking the retrieved documents by order of relevance. RDF graphs present a further challenge, when compared to the Web, since the information that a user needs may not be in a single triple, but rather it is distributed over the graph. Hence, an answer for a keyword query over an RDF graph is better formalized as a minimal subgraph of the RDF graph that covers the keywords.

Summarizing, there are three main tasks in *RDF-KwS*: (1) finding pieces of information in the RDF graph; (2) assembling the retrieved pieces of information to compose complete answers; (3) ranking the complete answers. The main motivation of this work is how to construct an *RDF-KwS* system that covers these three tasks.

To achieve a good ranking mechanism, typical information retrieval systems rank the documents based not only on how well they match the keyword

query, but also based on how important the documents are. The notion of importance for Web pages is typically computed using centrality measures for graphs created using the hyperlink structure of the Web. PageRank (Brin & Page 1998) and HITS (Kleinberg 1999) are some of the most popular centrality measures used in Web Information Retrieval. Their main idea is to assign high scores to pages that are referenced by many other important pages.

Returning to the RDF environment, the majority of the related work test their strategies using some RDF graph that reflects Web pages and their links, such as DBpedia[1] (Kasneci et al. 2008; Franz et al. 2009; Harth et al. 2009; Mirizzi et al. 2010; Le et al. 2014; Ngomo et al. 2017), or using some dataset about co-authorships of research papers, with data from DBLP[2] (Balmin et al. 2002; Franz et al. 2009; Wei et al. 2011), for example. We argue that PageRank or HITS variations work well for these types of RDF graphs because the incoming or outgoing edges actually indicate the relevance of a resource. In the Web, it is reasonable that a Web page (or node) with several incoming edges is more important than a Web page with a few incoming edges. Likewise, in an RDF graph about research publications, an author with many accepted papers is usually more important than an author of few accepted papers.

However, RDF-KwS operates over full RDF graphs, where the incoming or outgoing edges of a node do not necessarily indicate the node's importance with respect to any existing node relationship or, at least, it may be hard to detect which relationships would express the notion of importance. Thus, traditional measures may fail to compute the importance of a node. As an example, in an RDF graph representation of IMDb[3], instances of "common classes" (e.g. Genre, Language, Country, Company) have a high number of incoming edges. Hence, a traditional PageRank algorithm will assign scores to these common instances that are higher than the scores of popular movies and actors. Of course, we could manually assign weights to the object properties in order to capture their semantics, and use a Weighted PageRank or HITS Algorithm, as in (Balmin et al. 2002; Ding et al. 2004;

[1] http://dbpedia.org/sparql
[2] http://dblp.uni-trier.de
[3] www.imdb.com

Park et al. 2011). However, one may argue that the manual assignment of weights is bothersome and subjective. Thus, other works focused on strategies to learn weights based on user feedback (Nie et al. 2005; Agarwal et al. 2006; Komamizu et al. 2017). In addition to the difficulty of detecting relationships that express the importance of a graph node, it would be interesting to eliminate unwanted relationships that would distort traditional importance measures.

1.2. Goal and Contributions

Summarizing, the problems addressed in this work are: (1) how to define importance measures in RDF graphs in which the importance of a node is not directly related to its degree; (2) how to use these measures to help compute and rank answers of keyword queries over RDF graphs.

To solve these problems, the first and key contribution of this thesis is a novel family of importance measures, collectively called *InfoRank*, for RDF graphs. The proposed importance measures are combinations of three intuitions: (I) *"important things have lots of information about them"*; (II) *"important things are surrounded by other important things"*; (III) *"few important relations (e.g. friends) are better than many unimportant relations (e.g. acquaintances)"*. They require neither the manual assignment of weights to object properties nor a training dataset to use as input to a learning algorithm.

The second contribution is an *RDF-KwS* system, named *QUIRA* (*QUerying with InfoRAnk*), which uses InfoRank: to narrow the retrieved pieces of information; to choose the best paths to connect the resources (nodes) in the graph; and to rank the retrieved answers.

Finally, the third contribution of this thesis consists of two enriched RDF datasets, IMDb and MusicBrainz[4], along with keyword search benchmarks adapted to the RDF enviroment. We use these datasets in our experiments to assess the correctness and the performance of the importance measures and the translation algorithm.

[4] https://musicbrainz.org

1.3. Structure of the Thesis

The remainder of this thesis is structured as follows. Chapter 2 introduces background concepts about Linked Data and Keyword Search. It also discusses related work that brings together both worlds, that is, strategies that are focused on performing Keyword Search over Linked Data, and related work aimed at measuring node important in a graph. Chapter 3 presents the proposed measures to capture node importance in RDF graphs, called InfoRank. Chapter 4 shows how to use these measures in a strategy for keyword search over RDF graphs. Chapter 5 describes the architecture and interface of QUIRA, a system that implements the strategy proposed in Chapter 4. Chapter 6 covers experiments to assess the InfoRank importance measures and the QUIRA system. Finally, Chapter 7 concludes the thesis and indicates directions for future work.

2
Background and Related Work

In this chapter we present background concepts and related work. Section 2.1 defines Linked Data, RDF, SPARQL, the Web of Data and its principles, while Section 2.2 describes the Information Retrieval processes and some popular importance measures, such as PageRank. Finally, Section 2.3 presents the related work about keyword search over structured databases (relational and RDF) and some of the importance measures used in these works.

2.1. Linked Data

2.1.1. RDF Graphs and SPARQL

The Resource Description Framework (RDF)[5] is a framework for expressing information about resources. Resources can be anything (people, objects, concepts, etc.) and are described using triples. A *triple* is a statement that has a subject, a predicate and an object. Informally, an instance of a statement can be "The Mona Lisa was created by Leonardo Da Vinci", in which the subject is "The Mona Lisa", the predicate is "was created by" and the object is "Leonardo Da Vinci". The combination of the statements forms a graph, as shown in Figure 1.

Formally, in RDF, the subject and the predicate of the triple have to be represented as an URI, and the object can be a URI or a literal. URI stands for "Uniform Resource Identifier" and is a global identifier that allows different people to reuse the URI to identify the same thing. For instance, the dataset DBpedia uses the URI <http://dbpedia.org/resource/Mona_Lisa> to denote the Mona Lisa painting described by the corresponding Wikipedia article. Additionally, DBpedia

[5] https://www.w3.org/TR/rdf11-primer/

uses the URI <http://dbpedia.org/ontology/author> to represent the predicate “was created by” and the URI <http://dbpedia.org/resource/Leonardo_da_Vinci> to represent the object “Leonardo Da Vinci”.

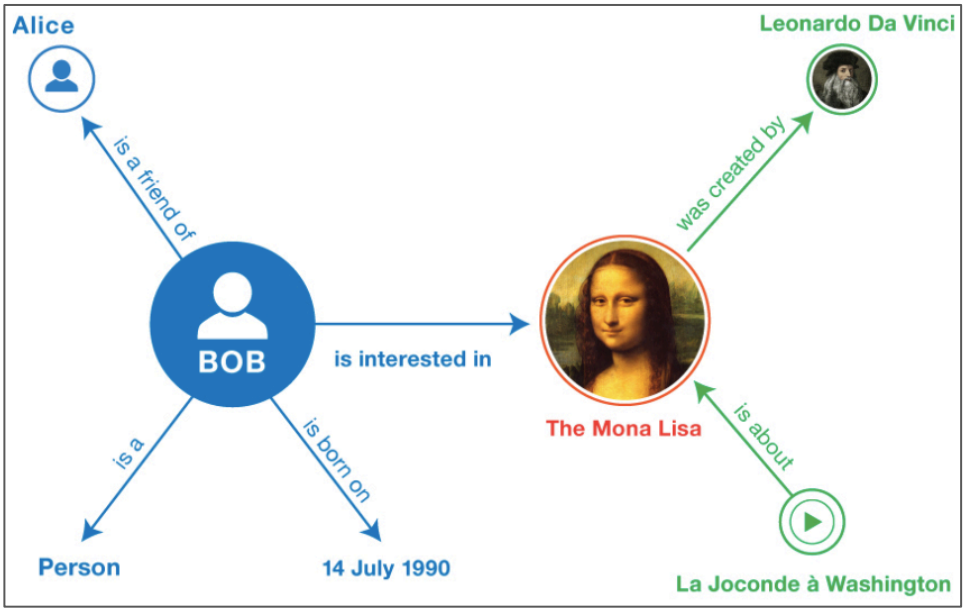


Figure 1: Informal RDF Graph

Finally, a literal is a basic value that is not a URI, such as a string, a number, a date, etc. For instance, DBpedia denotes the following triple, in which the object is literal:

(<http://dbpedia.org/resource/Mona_Lisa>,
<http://dbpedia.org/property/otherTitle>,
"La Joconde")

In practice, RDF is used in combination with vocabularies that provide semantic information about the resources. Examples of popular vocabularies are:

- RDF Schema: defines the basic idea of classes and properties. For example, one can state that the URI `http://www.example.org/friendOf` can be used as a property and that the subjects and objects of this predicate must be resources of class `http://www.example.org/Person`. Then, one can say that the resources Bob and Mary are of the type Person, and that Bob is a friend of Mary.

- OWL (Web Ontology Language): extends the expressivity of RDF Schema with additional primitives, such as *equivalent class*, *equivalent property*, *different from*, *same as*, etc.
- FOAF (Friend of a Friend): describes people, their activities and their relations to other people.
- Dublin Core: defines general attributes such as *title*, *creator*, *date* and *subject.*

The SPARQL query language[6] can be used to express queries over RDF graphs. A simple example of a SPARQL query is shown below, which returns the URIs of all people named "Mary".

```
SELECT ?subject
WHERE { ?subject rdf:type foaf:Person .
        ?subject foaf:name "Mary" }
```

The SELECT clause identifies the variables that will appear in the result (in this case, ?subject). The *where* clause contains the graph pattern that is matched with a RDF graph. The pattern in this example is a single triple, but SPARQL also supports aggregation, subqueries, negation, filters, etc.

Another important feature of SPARQL is the possibility to update RDF datasets, and to insert or delete triples. For instance, the *modify* operation can be used to remove or add triples based on bindings for a query pattern specified in a *where* clause, as in:

```
DELETE { ?person foaf:firstName "Bill" }
INSERT { ?person foaf:firstName "William" }
WHERE  { ?person foaf:firstName "Bill" .
         ?person foaf:lastName "Smith" }
```

2.1.2.Principles and The Web of Data

Tim Bernes-Lee introduced a set of best practices for publishing and interlinking structured data on the Web, known as Linked Data (Berners-Lee, 2006). There are four main principles that define Linked Data:

[6] http://www.w3.org/TR/sparql11-query/

- Use URIs as names for things.
- Use HTTP URIs so that people can look up those names.
- When someone looks up a URI, provide useful information using the standards (RDF, SPARQL).
- Include links to other URIs, so that they can discover more things.

The idea of the first principle is to extend the classic Web and use URIs (Uniform Resource Identifiers) to identify not only documents, but also any object or concept of the real world. URIs can identify concrete things, such as people, places, and cars, or abstract concepts, such as feelings and relations (Heath & Bizer, 2011).

Once there is a URI defining something, it needs to be combined with the HTTP protocol in order to enable the URI to be *dereferenced*, that is, to provide access to the description of objects and concepts.

The third principle promotes the use of standard content format to enable different applications to process Web content. The structured data can be represented and shared using a simple graph-based model, known as RDF (Resource Description Framework), described in section 2.1.1.

Finally, the fourth promotes the use of RDF triples to describe relationships between resources. Such triples are often referred to as *links*. For instance, to connect a person with a place, one may use the relationship "works".

Moreover, links should also be created between different datasets in order to create a global data space, called the Web of Data, which forms a large graph connecting RDF datasets from all sorts of topics, such as locations, people, publications, music, movie, etc. The idea of the Web of Data started to gain force in 2007 with the *LOD - Linked Open Data*[7] project. The aim of this project was to identify existing datasets available under open licenses and to publish them in RDF, according to the Linked Data Principles (Heath & Bizer, 2011). Subsequently, several individuals and organizations were stimulated to publish their data in the LOD using the Linked Data principles. Figure 2 shows the LOD graph for the datasets published until June 2018.

[7] https://lod-cloud.net

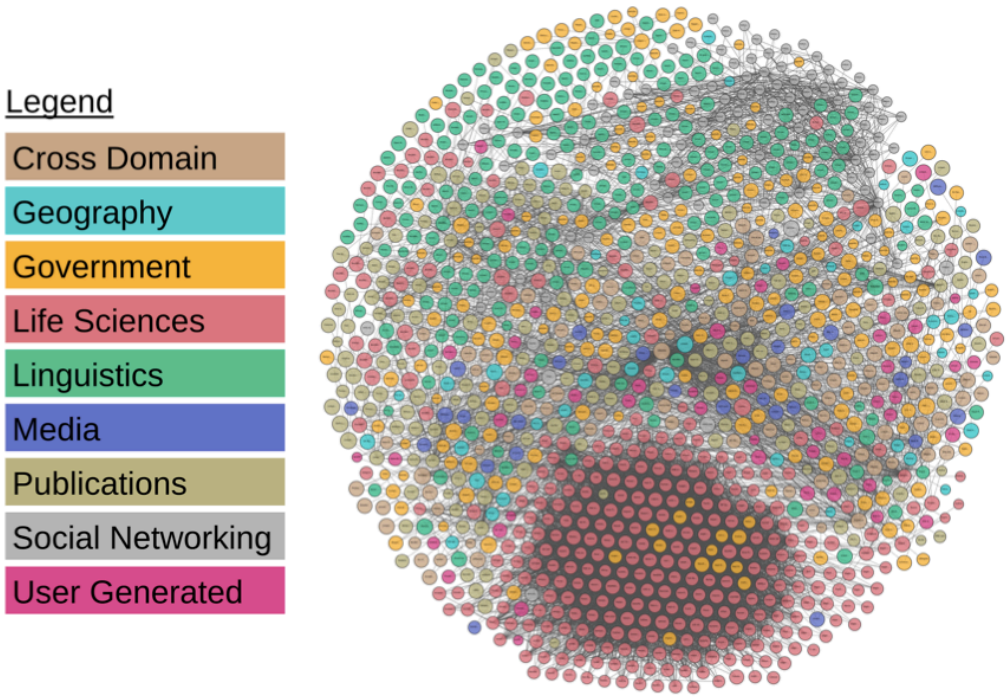


Figure 2: The LOD Cloud Diagram

2.2. Information Retrieval and Keyword Search

2.2.1.Overview

An information retrieval (IR) system is a software program that manages documents and helps users find the information they need. The documents that satisfy the user information need are called *relevant documents*. A *perfect* retrieval system would retrieve only relevant documents (that is, it would have 100% *precision*) and would retrieve all such documents (that is, it would have 100% *recall*). However, perfect retrieval systems do not exist since search statements are incomplete and relevance depends on the subjective opinion of users.

There are three basic processes an IR system has to support: the representation of the content of the documents, the representation of the user information need, and the comparison of the two representations. The processes are visualized in Figure 3, in which squared boxes represent data and rounded boxes represent processes (Hiemstra 2009).

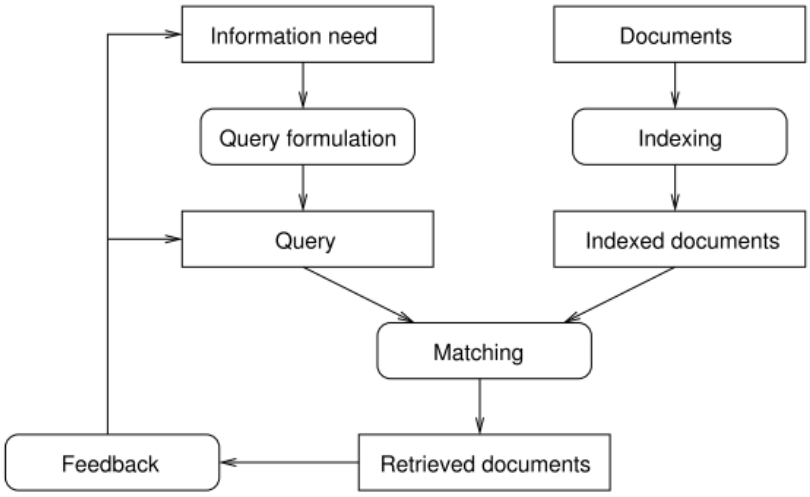


Figure 3: Information Retrieval Process

Representing the documents is usually called the *indexing process* and it takes place offline, that is, before the user accesses the IR system. The indexing process may include the actual storage of the document in the system, but often documents are only partially stored, for instance, only the title and the abstract, plus information about the actual location of the document. The process of representing the user information need is often referred to as the *query formulation process*. In a broad sense, query formulation might denote the complete interactive dialogue between the system and the user, leading not only to a suitable query but, possibly, also to the user better understanding his information need; this is denoted by the feedback process in Figure 3.

The comparison of the query against the document representations is called the *matching process*. This process usually results in a ranked list of documents. Users will walk down this document list in search of the information they need. Ranked retrieval will hopefully put the relevant documents at the top of the ranked list, minimizing the time the user has to invest in reading the documents. Simple but effective ranking algorithms use the frequency distribution of terms over documents, but also statistics over other information, such as importance measures over the graph created using the hyperlink structure of the Web.

Section 2.2.2 presents some definitions of popular importance measures, such as PageRank, while Section 2.2.3 presents some basic definitions of how to evaluate the returned list of ranked documents.

2.2.2. Importance Measures

Importance measures have as goal to identify the most important or central node in a graph, depending on what importance means. A simple way to compute the importance of a node is just to analyze its degree. However, this returns a local measure of importance, whereas in some contexts a global analysis of the graph is preferable. For instance, the *Betweenness Centrality* counts the number of shortest paths going through a node; hence it is able to identify important connectors in a graph. The *Closeness Centrality* measures the average distance from a node to all other nodes, hence the more central a node is, the closer it is to all other nodes.

Other types of importance measures try to capture the idea that "it is not about what you know, but who you know". That is, the notion of importance is given by how well-connected a node is to other important nodes. PageRank (Brin & Page 1998) is the most popular importance measure of this type. Using the hyperlink structure of the Web, the basic idea is that, if a Web page has links from other high-quality Web pages, then that is an indication that it is likely to be worth looking at the page.

PageRank can be computed using an iterative strategy, named the *Power Iteration* method. Let $G = (V, E)$ be a directed graph and $PR(v, i)$ be the PageRank score calculated at iteration i. First, we initialize all scores with the same value; then, for $0 < i < x$, we iterate until the computation of the centrality score converges or exceeds the maximum number of possible iterations x.

PageRank can be computed using an iterative method, called *Power Iteration*. Let $G = (V,E)$ be a directed graph and $PR(r,i)$ be the PageRank score of a node $r \in V$ calculated at iteration i. First, the method initializes all scores with the same value:

$$PR(r, 0) = 1/N \tag{1}$$

where N is the total number of nodes in G. Then, for $0<i<x$, it iterates until the computation of the score converges or exceeds x, the maximum number of iterations:

$$PR(r, i) = \frac{1-\alpha}{N} + \alpha \sum_{s \in M_I(r)} \frac{PR(s,i-1)}{d_O(s)} \tag{2}$$

where α is a dumping factor (usually set to 0.85), $M_I(r)$ is the set of nodes that have a link to r and $d_O(s)$ is the number of outgoing links from s.

One variant of PageRank uses link weights to give more importance to certain types of links. The *Weighted PageRank* PR_W is defined as:

$$PR_W(r,0) = 1/N \tag{3}$$

$$PR_W(r,i) = \frac{1-\alpha}{N} + \alpha \sum_{s \in M_I(r)} \frac{PR_W(s,i-1)}{d_O(s)} * w(r,s) \tag{4}$$

where $w(r,s)$ is a weight between 0 and 1 of edge $(r,s) \in E$.

Other importance measures that can be computed using the Power Iteration method are the HITS Authorities and Hubs (Kleinberg 1999). In the Web, a good Hub is page that works like a catalog or a directory of other pages, that is, a page that points to many other pages, whereas a good Authority is page that is referenced by many Hubs. HITS depends on a mutual recursion, hence, in the first step, we initialize all hubs scores with the same value:

$$Hub(r,0) = 1/N \tag{5}$$

Then, for each iteration $0<i<x$, we first update the authorities using the initialized hubs:

$$Auth(r,i) = \sum_{s \in M_I(r)} Hub(s,i-1) \tag{6}$$

Then, we update the hubs using the authorities:

$$Hub(r,i) = \sum_{s \in M_O(r)} Auth(s,i-1) \tag{7}$$

where $M_O(r)$ is the set of nodes that r links to.

As the last step of the iteration, we normalize the Authorities and Hubs scores by dividing them by the respective maxinum score. The algorithm stops when the computation of the score converges or exceeds the maximum number of iterations.

2.2.3. Evaluation of Information Retrieval Systems

When applied to a graph, a centrality or importance measure induces a ranked list L of the set of nodes of the graph, which can be compared to a *golden standard* list S. In this section, we therefore recall the definitions of precision and average precision for the ranked lists, which we will use to compare the measures.

Let *S* be a list of documents, considered as the *golden standard*, and let *R* be the set of all documents in *S*. A document *d* is *relevant* iff $d \in R$. Let *L* be a list of documents.

The *precision at position k* of *L* with respect to *S* is defined as:

$$P(k) = \frac{|\ R \cap retrieved\ (k)\ |}{|\ retrieved\ (k)\ |}$$

where *retrieved*(*k*) is the set of all documents in *L* until position *k*.

The *average precision* of *L*, denoted AP_L, with respect to *S* is defined as:

$$AP_L = \frac{1}{|R|} \sum_{k=1}^{n} relevance(k) * P(k)$$

where *relevance*(*k*) is an indicator function that returns *1*, if the document at position *k* is relevant, and *0*, otherwise. Notice that the average precision of the golden standard S is $AP_S = 1$, which is the target performance of a centrality measure.

2.3. Related Work

2.3.1. Keyword Search Over Structured Databases

Tools that implement keyword-based queries over relational databases and RDF datasets have been investigated for some time. Since both fields have similar challenges, we discuss them together.

We may distinguish between tools that are *schema-based*, in the sense that they use information about the conceptual schema to compile a keyword-based query into an SQL or SPARQL query, from those that are *graph-based*, which operate directly on the data.

BANKS (Bhalotia et al. 2002) and BLINKS (He et al. 2007) are examples of relational graph-based tools, and Sindice (Oren et al. 2008) and *Structured LM* (Elbassuoni & Blanco, 2011) are examples of RDF graph-based tools.

Relational schema-based tools explore the foreign keys declared in the relational schema to compile a keyword-based query into an SQL query with a minimal set of join clauses, based on the notion of *candidate networks* (CNs). This approach was first proposed in DISCOVER (Hristidis & Papakonstantinou 2002)

and DBXplorer (Agrawal et al. 2006) and adopted in a quite a few tools, including recent ones (De Oliveira et al. 2015).

Schema-based tools gained more attention since they take advantage of the query processor of the database (or triple store), instead of processing row by row (or triple by triple) as in graph-based tools. Hence, we continue our discussion with keyword search tools over RDF that are schema-based, that is, tools that translate keyword queries into SPARQL queries.

SPARK (Zhou et al. 2007) is an early RDF schema-based tool, which, given a keyword query, returns a ranked list of SPARQL queries as a result of the translation. In the first step of the process, the algorithm tries to match query terms with resources. In the second step, it tries to find missing relations that connect these terms to synthesize the SPARQL queries. Finally, in the third step, it uses a probabilistic ranking model to rank possible queries. However, the ranking model only considers the keywords matched and some statistics about the knowledge base, it does not consider the importance of nodes when ranking the queries.

QUICK (Zenz et al. 2009) is an RDF schema-based tool that tries to circumvent the problem of generating multiple possible queries by executing the translation process incrementally based on user feedback. QUICK also does not consider the importance of the nodes when synthesizing SPARQL queries.

Hermes (Tran et al. 2009) combines the idea of generating summary graphs for the original RDF graph, using the class hierarchy, to generate and rank candidate SPARQL queries. To capture the idea of importance, Hermes uses simple metrics over the summary graph.

The *QUIOW* tool, our first implementation (García et al. 2017; Izquierdo et al. 2018), is schema-based and supports both the RDF and the relational environments by translating keyword queries into SPARQL or SQL queries. Although the tool proved efficient for an industrial dataset about petroleum, it had a poor performance for an RDF graph representation of IMDb due to the large size and ambiguity of the domain. The importance measures introduced in this thesis remediate these problems, as shown in Section 5.

2.3.2. Importance Measures for Structured Databases

ObjectRank (Balmin et al. 2004) was one of the first proposals to compute a global importance score for database entities using PageRank. The authors transformed the structure of a relational database (RDB) into a graph, using foreign keys as links between entities, and them applied PageRank with manual weight assignment to different types of links. The authors evaluated their strategy using the DBLP dataset.

In RDF, other works that manually assign weights to use with PageRank are: Swoogle (Ding et al. 2004), which evaluated their strategy using documents crawled from the Web; Park et al. (2011), which performs evaluation using their own small research dataset; and Beagle++ (Chirita et al. 2006), which adapted ObjectRank to an RDF Graph about activity metadata in desktops.

NAGA (Kasneci et al. 2008) is a semantic search engine that uses a PageRank-like algorithm to capture the provenance of information and use the confidence to rank answers of a query. The authors performed the evaluation using TREC[8] datasets.

Harth et al. (2009) use the concept of Authority of HITS to rank RDF data, considering the provenance of the information through the naming authorities. The authors evaluate their strategy using two different RDF datasets crawled from the Web and comparing with PageRank. The authors justify that they did not compared with ObjectRank because the manual assignment of weights to the thousands of properties in the crawled datasets would be unfeasible.

TripleRank (Franz et al. 2009) represented an RDF graph as a tensor. Then, it used the PARAFAC decomposition of the tensor to induce groups of properties and resources, with authority and hub scores for the particular latent aspect (topic) the group represents. It showed how to use the result of the PARAFAC decomposition to guide a faceted browsing application. Finally, it tested the application in several experiments over RDF datasets with 5 to 55 thousand triples. PARAFAC decomposition proved interesting for faceted browsing exactly because it induces groups of properties and resources, together with authority and hub scores. However, it is not clear how to extend this strategy to the context of keyword

[8] https://trec.nist.gov

search, not to mention the problem of computing the PARAFAC decomposition of tensors with 200+ million non-zero entries, as in the experiments described in Section 5. Finally, the authors tested their strategy using DBpedia and DBLP datasets.

DBpediaRanker (Mirizzi et al. 2010) focus their strategy on ranking resources from DBpedia to generate ad-hoc tag clouds given a query. They use domain experts to identify key nodes of the domain and then use similarity algorithms to retrieve all the nodes within the same domain.

RareRank (Wei et al. 2011) is an algorithm for ranking entities in the scientific research domain. It adapts PageRank to combine two different scores: link information (e.g., a citation between two publications), and the content information (e.g., provided by the links between document-topic and topic-topic).

HARE (Ngomo et al. 2017) is another adaptation of PageRank that allows the simultaneous computation of ranks for RDF triples, resources, properties and literals using bi-partite graphs. In the evaluation they show that HARE is up to 2 orders of magnitude faster than PageRank and the ranking quality is comparable regarding DBpedia classes.

FORK (Komamizu et al. 2017) adapted ObjectRank to Linked Data. The main contribution of the work is a learning algorithm for property weights based on user's relevance feedbacks, instead of the manual assignment of weights. The authors evaluated their strategy using DBpedia and results showed that FORK achieves the best ranking method when compared to baseline approaches. Similary, DBtrends (Marx et al. 2016a) uses query logs to improve their ranking function.

As mentioned in the introduction, DBpedia and DBLP (or other research datasets) are highly influenced by link semantics: DBLP through authorship links, and DBpedia through links derived from Wikipedia, such as, *wikiPageRedirects, wikiPageDisambiguates, primaryTopic, etc*. Furthermore, in the LOD cloud, DBpedia has many incoming links from other RDF datasets.

For further references that focus on ranking strategies for degree-dependent datasets, such as DBpedia or DBLP, we refer the reader (Bast et al. 2016; Roa-Valverde et al. 2014; Yumusak et al. 2014; Marx et al. 2016b). We continue our discussion with some alternative strategies that do not highly depend on node degree.

Graves et al. (2008) propose the use of closeness centrality for undirected graphs and evaluates the strategy using three datasets, CIA Factbook, Terrorist Ontology and Wine Ontology. The authors compare their strategy with a ranking using the number of incoming edges. The problem with closeness centrality is that it is not efficient for large RDF graphs.

Although the work presented in (Kim et al. 2016) is not specific to RDF graphs, it proposes the *degree decoupled PageRank* technique that penalizes or boosts the importance of the node degree in recommendation graphs, depending on the domain characteristics. They argue that, in some contexts, the importance of the node can be inversely proportional to its degree. The authors performed an evaluation using graphs extracted from IMDb, Last.fm, DBLP and Epinions. From results for the IMDb dataset, they noticed that, for a movie recommendation graph, traditional PageRank performs better; however, for an actor recommendation graph, the node degree actually needs to be penalized. They argue that, when an actor plays in a large number of movies, he probably is a non-discriminating ("B movie") actor, whereas, when an actor is associated with relatively few movies, he may be a more discriminating ("A movie") actor.

2.4. Chapter Conclusion

In this chapter we showed the main concepts about Linked Data and Information Retrieval. In the related work section, we showed the combination of both worlds with strategies to solve the problem of keyword search over structured databases, either relational or RDF. We divided the works into two classes. The first class contains popular works that focus on solving the main problems of this field. The second class includes works that specifically deal with the ranking problem using importance measures. We showed that most of the works use some adaptation of PageRank or HITS and test their strategies with degree-dependent datasets, such as DBpedia and DBLP.

3
The InfoRank Importance Measures

This chapter introduces a novel family of importance measures, collectively called *InfoRank*, for RDF graphs. The proposed importance measures are combinations of three intuitions: (I) *"important things have lots of information about them"*; (II) *"important things are surrounded by other important things"*; (III) *"few important relations (e.g. friends) are better than many unimportant relations (e.g. acquaintances)"*. These intuitions are discussed in Section 3.1 along with formal definitions. Moreover, Section 3.2 presents a running example and details of the implementation.

3.1. Discussion and Formal Definitions

Following the intuition that *"important things have lots of information about them"* and observing the way that RDF graphs are modeled, we notice that more important nodes are usually associated with more literals (information) through datatype properties than less important nodes. This comes from the nature of RDF datasets versus normalized relational databases. A decent triplification of a normalized relational database would include a de-normalization step and would filter out null values, which would result in resources with a different number of datatype property values and object properties. Hence, the more "complete" the profile of an actor or a film, say, the more important that actor or film would be.

As example, consider some different trivia extracted from IMDb of a movie with international projection, such as *Titanic* (1997):

```
imdb:Titanic imdb:trivia "A 1/8 scale model of the ship's
                          stern was also used." .
imdb:Titanic imdb:trivia "Was the first film to be filmed at Fox
                          Studios Baja."
imdb:Titanic imdb:trivia "The \"ale\" in the below decks party
                          was actually root beer."
imdb:Titanic imdb:trivia "Tom Cruise was considered for the role
                          of Jack Dawson."
```

```
imdb:Titanic imdb:trivia "The most-voted-for film on IMDb that
                          is not on the Top 250 List."
imdb:Titanic imdb:trivia "Most of the ocean which extras were
                          jumping into was 3 feet deep."
imdb:Titanic imdb:trivia "Macaulay Culkin was considered for the
                          role of Jack Dawson."
imdb:Titanic imdb:trivia "For some wreck interior shots, a set
                          was constructed and submerged."
imdb:Titanic imdb:trivia "Barbra Streisand was considered for
                          the role of Molly Brown."
imdb:Titanic imdb:trivia "British newspapers alleged that
                          Michael Caine refused a role."
...
```

Additionally, in IMDb, *Titanic* has a total of 205 triples with trivia, 134 triples with quotes said by the characters, 180 triples with tags, and so on. In total there are 1297 literals describing the movie *Titanic*. Now, considering a movie with only national projection, such as the Brazilian movie *O Auto da Compadecida*, which has only 70 literals describing it.

Continuing the example, we can also analyze a multilingual dataset, such as DBpedia, in which *Titanic* has the following labels:

```
:Titanic(1997) rdfs:label "Titanic (1997 film)"@en .
:Titanic(1997) rdfs:label ")1997 فيلم( تيتانيك"@ar .
:Titanic(1997) rdfs:label "Titanic (1997)"@de .
:Titanic(1997) rdfs:label "Titanic (película de 1997)"@es .
:Titanic(1997) rdfs:label "Titanic (film, 1997)"@fr .
:Titanic(1997) rdfs:label "Titanic (film 1997)"@it .
:Titanic(1997) rdfs:label "タイタニック (1997年の映画)"@ja .
:Titanic(1997) rdfs:label "Titanic (1997)"@nl .
:Titanic(1997) rdfs:label "Titanic (film 1997)"@pl .
:Titanic(1997) rdfs:label "Titanic (1997)"@pt .
:Titanic(1997) rdfs:label "Титаник (фильм, 1997)"@ru .
:Titanic(1997) rdfs:label "泰坦尼克号 (1997年电影)"@zh .
```

and *O Auto da Compadecida* has the following ones:

```
:A_Dog's_Will rdfs:label "O Auto da Compadecida (filme)"@pt .
:A_Dog's_Will rdfs:label "A Dog's Will"@en
```

Note that, in a relational database, different labels or trivia of the same movie would need to be modeled in a different table according to the *First Normal Form* (1NF) of relational databases. However, in RDF, this can be modeled as literals connected directly to the resources. Hence, resources that are more "complete" (i.e. with more information) have a higher number of literals, instead of a higher degree through foreign keys.

The second intuition that we follow is inspired by PageRank and says that *"important things are surrounded by other important things"*. For instance, the movie "Titanic" has links through object properties with actors "Kate Winslet" and "Leonardo Dicaprio", which are also important nodes in the graph. As in (Graves et al. 2008), we agree that, in RDF graphs, the direction of an object property does not have the same meaning as a Web hyperlink since a property is often found in its inverse form (e.g. directedBy/directs). Given that, we treat an RDF graph as undirected and consider all neighbors of a node (i.e. all other nodes that have an object property linked to it) when propagating the importance with PageRank.

We further improve this intuition by introducing a third one that says "few friends are better than many acquaintances". As discussed in the introduction, the typical centrality measures are highly dependent on the degree of the node. In our work, we do not want to boost (or penalize) the degree importance, but we focus on a strategy that favors the quality of relations, rather than their quantity, that is, we prefer an approach that captures the notion that *"few important relations (e.g. friends) are better than many unimportant relations (e.g. acquaintances)"*.

Formally, let T be a set of RDF triples. Assume that T contains schema information and that it is possible to identify the set C of *classes* defined in T, the set P of *object properties* defined in T, the set D of *datatype properties* defined in T, the set L of *literals* defined in T, and the set R of *(class) instances* defined in T, i.e., $r \in R$ iff there is a triple $(r,rdf:type,c) \in T$ such that $c \in C$. For simplicity, we assume that there are no blank nodes.

Instance Informativeness. The level of "informativeness" of a resource measures how informative the resource is. As previously discussed, information is represented as literals in RDF graphs. However, data resources (instances) usually have more literals than metadata resources (classes and properties). Hence, we first focus our strategy on the informativeness of instances.

The *informativeness* of an instance $r \in R$, denoted $IW(r)$, is defined as the number of triples of the form $(r,p,v) \in T$, where $v \in L$.

Ranking Schema Elements. Continuing our strategy based on instance informativeness, we say that "important classes usually have informative instances" and "important properties are usually those connecting informative instances".

The *absolute informativeness* of a class $c \in C$ is defined as the maximum value of *IW(r)* of all instances *r* of class *c*. The *informativeness* of a class *c*, denoted *IR(c)*, is the absolute informativeness of *c*, divided by the maximum absolute informativeness value of all classes in *C*. That is, we normalize the values of *IR(c)* by their maximum value. We will rank classes by descending order of *IR(c)*.

Likewise, the *absolute informativeness* of an object property $p \in P$ is defined as the maximum value of *IW(r)+IW(s)* of all triples of the form $(r,p,s) \in T$. The *informativeness* of *p*, denoted *IR(p)*, is the absolute informativeness of *p*, divided by the maximum absolute informativeness value of all object properties in *P*. We will rank object properties by descending order of *IR(p)*.

On the other hand, the *absolute informativeness* of a datatype property $d \in D$ is defined as the number of distinct literals, i.e. distinct values of $v \in L$ of all triples of the form (*r*,*d*,*v*). The *informativeness* of *d*, denoted *IR(d)*, is the absolute informativeness of *d*, divided by the maximum absolute informativeness value of all datatype properties in *D*. We will rank datatype properties by descending order of *IR(d)*.

Ranking Data. Note that we used only *Intuition I* in our strategies to rank metadata resources. However, we propose a combination of the three intuitions to rank instances.

Let $r,s \in R$ and $p \in P$. Assume that $(r,p,s) \in T$ or $(s,p,r) \in T$, that is, ignore the direction of the object property *p*. The *normalized weight* of (*r*,*p*), denoted *W*(*r*,*p*), is defined as:

$$W(r,p) = IR(p) / \sum_{q \in P \text{ and } ((r,q,t) \in T \text{ or } (t,q,r) \in T)} IR(q) \qquad (8)$$

Note that the normalized weight *W*(*r*,*p*) does not depend of "who" the neighbors of *r* are, but it depends only on how they are connected to *r*, that is, it considers the InfoRank scores of properties *p* and *q*, for all properties *q* whose domain includes *r*.

Again, we initialize all scores with the same value:

$$PR_W(r,0) = 1/N \qquad (9)$$

where *N* is the total number of nodes in *G*. Then, we compute PageRank using *W*(*r*,*p*) as the edge weights:

$$PR_W(r,i) = \frac{1-\alpha}{N} + \alpha \sum_{(r,p,s)\in T \text{ or } (s,p,r)\in T} PR_W(s,i-1) * W(s,p) \quad (10)$$

where α is a dumping factor (usually set to 0.85).

The InfoRank score of an instance *r*, denoted *IR*(*r*), is the final PageRank score of *r* after *x* iterations, $PR_W(r,x)$, weighted by the informativeness of *r*, *IW*(*r*):

$$IR(r) = PR_W(r,x) * IW(r) \quad (11)$$

3.2. Implementation and Example

3.2.1. Running Example

In order to exemplify the implementation steps, consider the simple graph shown in Figure 4 with IRIs denoted in oval and literals denoted in dashed boxes.

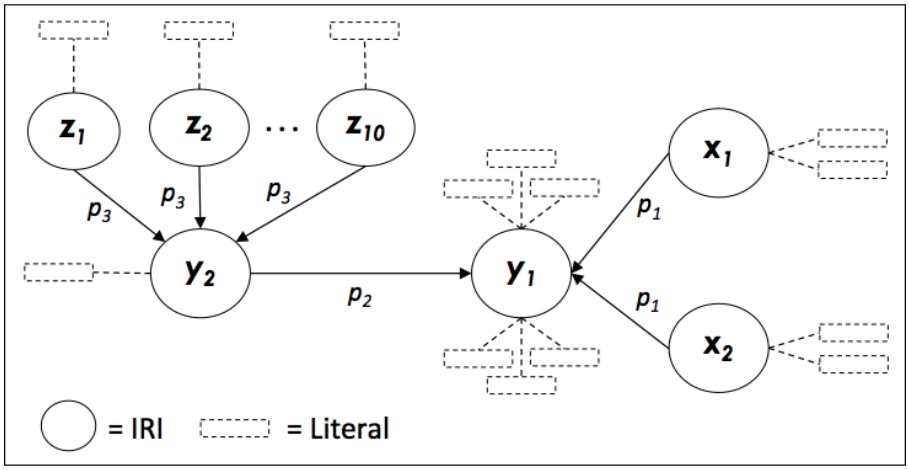


Figure 4: Graph Example

Furthermore, consider that the graph also includes the following triples:

(*X*, *rdf:type*, *rdfs:Class*), (*Y*, *rdf:type*, *rdfs:Class*), (*Z*, *rdf:type*, *rdfs:Class*)

(x_1, *rdf:type*, *X*), (x_2, *rdf:type*, *X*), (y_1, *rdf:type*, *Y*), (y_2, *rdf:type*, *Y*),

(z_1, *rdf:type*, *Z*), (z_2, *rdf:type*, *Z*), … (z_{10}, *rdf:type*, *Z*)

3.2.2.Computing Informativeness

In the first step to achieve the final InfoRank score, we need to compute the informativeness of instances by counting the number of literals as described in Section 3.1. This can be computed with the following SPARQL query.

Instances

```
select ?r ?iw
where
{ ?r rdf:type/rdf:type rdfs:Class .
  OPTIONAL
  { select ?r (count(?o) as ?inf)
    where { ?r ?p ?o .
            filter ( isLiteral(?o) )  }
    group by ?r
  }
  BIND(if(bound(?inf), ?inf, xsd:integer("0")) AS ?iw)
}
```

Note that, since our graph contains schema information, we apply a graph pattern that requires that variable *?r* be bound only to instances of some class (recall that we assume that are no blank nodes). Furthermore, since some instances may not have literals, we need to make the count of literals optional, so if there is no bound, the value returned is 0.

In the second step, we calculate the informativeness of the classes considering the informativeness of the instances to capture the idea that "important classes usually have informative instances". The following SPARQL query shows how to compute the absolute informativeness of classes.

Classes

```
select ?c (max(?iw) as ?abs)
where
{  ?r rdf:type ?c .
   ?c rdf:type rdfs:Class .
   OPTIONAL
   { select ?r (count(?o) as ?inf)
     where { ?r ?p ?o .
             filter ( isLiteral(?o) ) }
     group by ?r
   }
   BIND(if(bound(?inf), ?inf, xsd:integer("0")) AS ?iw)
}
group by ?c
```

In the third step, we calculate the informativeness of object properties, also considering the informativeness of the instances to capture the idea that "important properties are usually those connecting informative instances". The following SPARQL query shows how to compute the absolute informativeness of object properties.

Object Properties

```
select ?p (max(?iw_r + ?iw_s) as ?abs)
where
```

```
{ ?r ?p ?s .
  ?r rdf:type/rdf:type rdfs:Class .
  ?s rdf:type/rdf:type rdfs:Class .
  OPTIONAL
  { select ?r (count(?o) as ?inf_r)
    where { ?r ?p2 ?o .
    filter ( isLiteral(?o) )  }
    group by ?r
  }
  OPTIONAL
  { select ?s (count(?o) as ?inf_s)
    where { ?s ?p2 ?o . filter ( isLiteral(?o) )  }
    group by ?s
  }
  BIND(if(bound(?inf_r), ?inf_r, xsd:integer("0")) AS ?iw_r)
  BIND(if(bound(?inf_s), ?inf_s, xsd:integer("0")) AS ?iw_s)
}
group by ?p
```

Moreover, we can compute the informativeness of datatype properties using the following query.

```
                        Datatype Properties
select ?d (count(distinct ?o) as ?abs)
where
{ ?r rdf:type/rdf:type rdfs:Class .
  ?r ?d ?o . filter ( isLiteral(?o) )
}
group by ?d
```

Finally, Table 1 presents the result of the queries to calculate the informativeness of instances, classes and object properties for our running example. In this example, we assume that there is just one datatype property.

Table 1: Informativeness of resources for the running example.

Instances		Classes		Object Properties	
?r	***?iw***	***?c***	***?abs***	***?p***	***?abs***
x_1 , x_2	2	X	2	p_1	8
y_1	6	Y	6	p_2	7
y_2 , $z_1 \ldots z_{10}$	1	Z	1	p_3	2

3.2.3. Computing InfoRank

Given that we use Oracle 12c as the triple store for our RDF dataset and that it would not be possible to load the graph matrix in memory to compute PageRank or InfoRank, we used an iterative strategy with SQL queries to simulate the Power Iteration method.

As a pre-processing step, in addition to the informativeness of instances, we need to compute their degrees, which can be done with the following SPARQL query.

```
select ?r as ?instance ((?in_dgr + ?out_dgr) as ?degree)
where
{ ?r rdf:type/rdf:type rdfs:Class .
  OPTIONAL { select ?r (count(?o) as ?out_d)
             where{ ?r ?p ?o .
                    ?o rdf:type/rdf:type rdfs:Class }
             group by ?r
           }
  OPTIONAL { select ?r  (count(?o) as ?in_d)
             where { ?o ?p ?r }
             group by ?r
           }
  BIND(if(bound(?out_d), ?out_d, xsd:integer("0")) AS ?out_dgr )
  BIND(if(bound(?in_d), ?in_d, xsd:integer("0")) AS ?in_dgr )
}
```

Now, we are able to materialize a relational table, named *NODES*, with the informativeness and the degrees of each instance. Table 2 shows how this table would look like when considering our running example.

Table 2: Table *NODES* for the running example.

instance	*in_dgr*	*out_dgr*	*degree*	*info*
x_1	0	1	1	2
x_2	0	1	1	2
y_1	3	0	3	6
y_2	10	1	11	1
z_1	0	1	1	1
…	…	…	…	…

We materialize another table, named *EDGES,* to represent the instances and their neighbors. Table 3 shows how this table would look like when considering our running example.

Table 3: Table *EDGES* for the running example.

instance	*property*	*neighbor*	*direction*	*abs*	*info*	*weight*
x_1	p_1	y_1	1	8	1.00	1.00
y_1	p_1	x_1	0	8	1.00	0.34
y_2	p_2	y_1	1	7	0.87	0.25
y_1	p_2	y_2	0	7	0.87	0.30
...	...	...	...	...	...	...

Note that column *instance* contains all instances in the graph that have a neighbor, either through direct edges (*direction* = 1) or through inverse edges (*direction* = 0). This was done for efficiency purposes. The following SPARQL query shows how we can compute the instance/neighbor relation of table *EDGES*.

```
select ?instance ?p ?neighbor ?dir
where
{ { select (?r as ?instance) ?p (?s as ?neighbor) ("1" as ?dir)
    where { ?r ?p ?s .
            ?r rdf:type/rdf:type rdfs:Class .
            ?s rdf:type/rdf:type rdfs:Class }
  }
  UNION
  { select (?s as ?instance) ?p (?r as ?neighbor) ("0" as ?dir)
    where { ?r ?p ?s .
            ?r rdf:type/rdf:type rdfs:Class .
            ?s rdf:type/rdf:type rdfs:Class . }
  }
}
```

Furthermore, we join the result with the absolute informativeness (column *abs*) of the given *property* that was previously computed (see Table 1). We also present the normalized informativeness (column *info*), as described in Section 3.1.

The next step is to compute the normalized *weight* for each edge, as defined in Equation 5. The following SQL query shows how to compute this weight, and how to recreate table *EDGES* to include the new weight column.

```
create table EDGES_2 as
select e1.*, info/sum_info as weight
from EDGES e1
join (select instance, sum(info) as sum_info
      from EDGES
      group by instance) e2
on e1.instance = e2.instance;

drop table EDGES;
rename EDGES_2 to EDGES;
```

Moreover, note that, if we want to consider the graph as directed, we only need to add a clause like *where direction = 1*, or *where direction = 0* to consider a graph with the inverse edges.

Finally, Algorithm 1 implements the Power Iteration method. We used the codes of PageRank from Python's package Networkx[9] as a base for our implementation. The inputs of the algorithm are the maximum number of iterations allowed, the tolerance for convergence and the dumping factor.

In the first step of the Power Iteration method, we run an SQL query to create a table, named *LAST*, which contains the initial scores, as defined in Equation 6. Then, for each iteration, we run an SQL query that simulates the PageRank propagation of scores, as defined in Equation 7. Again, if we want to consider the graph as directed, we only need to modify the *where* clause, as explained before.

```
Algorithm 1. Power Iteration method with SQL Queries
Input: max_iter, tol, fac
Output: a table with the final PageRank score

N = select count(*) from NODES

run ( create table LAST as
      select instance, 1/N as score
      from NODES n1 )

iter = 0

while ( iter < max_iter ):
     iter += 1
     run ( create table CURR as
           select e1.neighbor as instance,
              ((1 - fac)/N) + (fac * SUM(score*weight)) as score
           from EDGES e1
           join LAST n1 on (e1.instance = n1.instance)
           group by e1.neighbor )
     conv = run ( select SUM(ABS(t2.score - t1.score))
                  from LAST t1
                  join CURR t2 on (t1.instance = t2.instance )
     drop table LAST
     if (conv <= tol):
           return table CURR
     else:
           rename table CURR to LAST
return null
```

The result of the propagation is stored in a new table named *CURR*, which is then compared with table *LAST* to check convergence. The algorithm stops when the convergence achieves the tolerance or the number of iterations exceeds the

[9] https://networkx.github.io

maximum number of iterations allowed. In the case of convergence, table *CURR* will have the final weighted PageRank score.

Table 4 shows the PageRank scores of instances throughout the iterations using a tolerance of 1.0e-4 and a dumping factor of 0.85. Note that, the highest score alternates between instances y_1 and y_2. However, since y_2 has a higher degree, it ends up with the highest PageRank score when the algorithm achieves convergence after 57 iterations.

Table 4: Example of the Power Iteration method.

instance	*i = 0*	*i = 1*	*i = 2*	*i = 3*	*i = 4*	*i = 57*
x_1, x_2	0.071	0.032	0.054	0.071	0.053	0.072
y_1	0.071	0.148	**0.205**	0.143	**0.241**	0.206
y_2	0.071	**0.636**	0.178	**0.495**	0.234	**0.334**
$z_1 \ldots z_{10}$	0.071	0.015	0.051	0.022	0.042	0.032

Hence, to conclude our strategy, we simply join table *CURR* with table *NODES*, so we can weight the PageRank score by the instance informativeness to finally obtain the InfoRank score, as defined in Equation 8. The following SQL query shows how to achieve the final score.

```
select n1.instance, info,
       score as pagerank,
       info * score as inforank
from teste_nodes n1
join teste_curr n2 on n1.instance = n2.instance
```

The final InfoRank result of our running example is shown in Table 5. Instance y_1 finishes with the highest InfoRank score, since it has a good PageRank score and a large level of informativeness.

Table 5: InfoRank result for the running example.

instance	*info*	*pagerank*	*inforank*
x_1, x_2	2	0.072	0.143
y_1	**6**	0.206	**1.236**
y_2	1	**0.334**	0.334
$z_1 \ldots z_{10}$	1	0.032	0.032

As the last step of our strategy, we materialize the InfoRank scores as triples of the graph. Hence, the graph of our running example would also include the following triples:

(*X*, *:inforank*, "0.333"), (*Y*, *:inforank*, "1.000"), (*Z*, *:inforank*, "0.166"),
(*p*$_1$, *:inforank*, "1.000"), (*p*$_2$, *:inforank*, "0.875"), (*p*$_3$, *:inforank*, "0.250"),
(*x*$_1$, *:inforank*, "0.143"), (*x*$_2$, *:inforank*, "0.143"),
(*y*$_1$, *:inforank*, "1.236"), (*y*$_2$, *:inforank*, "0.334"),
(*z*$_1$, *:inforank*, "0.032"), …, (*z*$_{10}$, *:inforank*, "0.032")

4
The Process of Keyword Search over RDF Graphs

This chapter presents a solution that covers the three main tasks of the process of Keyword Search over RDF Graphs: (1) finding pieces of information in the RDF graph; (2) assembling the retrieved pieces of information to compose complete answers; (3) ranking the complete answers. Section 4.1 presents an overview of the problem and our strategy, Section 4.2.1 to 4.2.3 discuss in detail solutions to the these three tasks.

4.1. Discussion and Problem Definition

4.1.1. The Keyword Search Problem

Let T be a set of RDF triples, G_T be the RDF graph induced by T, and L be the set of *literals* defined in T. Assume that T contains schema information, that is, T follows an RDF schema S, with $S \subseteq T$.

A *keyword-based query* K is simply a set of literals, or *keywords*. Let *match*: $\boldsymbol{L}\times\boldsymbol{L} \rightarrow [0,1]$ be a similarity function between literals such that $match(s,t)=j$ indicates how similar s and t are: $j=1$ says that s and t are identical, and $j=0$ indicates that s and t are completely dissimilar. Let $\sigma \in [0,1]$ be a *similarity threshold*. We leave *match* and σ unspecified at this point.

The set $\boldsymbol{M}[K,T]$ of *literal matches* between K and literals of T is defined as (recall that $S \subseteq T$):

$$\boldsymbol{M}[K,T] = \{ (k,(r,p,o)) \in K\times T \,/\, (r,p,o) \in T \wedge match(k,o) \geq \sigma \}$$

An *answer* for K over T is a subset A of T such that:

1) There is a subset of K, denoted K/A, such that, for each $k \in K/A$ there is $(r,p,o) \in A$ such that $(k,(r,p,o)) \in \boldsymbol{M}[K,T]$.
2) There is no other answer B for K over T such that $(K/A < K/B)$.

We say that K/A is the set of keywords *matched* by A. Condition (1) says that k matches the literal of a triple (r,p,v) in A, in which r can be an instance, a class or a property (recall that $S \subseteq T$). Also, Condition (1) does not require that all keywords in K be matched in an answer. Indeed, we say that A is *total* iff $K/A = K$, and *partial* otherwise. Condition (2) requires that an answer must match as many keywords in K as possible.

This notion of an answer is quite liberal. In particular, it allows the RDF graph G_A induced by an answer A to be disconnected. To circumvent this problem, we define a partial order between answers, using their induced RDF graphs, as follows. Given a directed graph G, let $|G|$ denote the number of nodes and edges of G and $\#c(G)$ denote the number of connected components of G, when the direction of the edges of G is disregarded. We define a partial order "<" for graphs such that, given two graphs G and G':

$$G < G' \text{ iff } (\#c(G) + |G|) < (\#c(G') + |G'|) \text{ or}$$
$$(\#c(G) + |G|) = (\#c(G') + |G'|) \text{ and } \#c(G) < \#c(G')$$

We use the partial order "<" between graphs to compare answers. Let A and B be two answers and G_A and G_B be their RDF graphs. We say that *A is smaller than B* iff $G_A < G_B$. An answer A for K over T is *minimal* iff there is no other answer B for K over T such that $G_B < G_A$.

Recall that G_T is the RDF graph induced by T. A possible strategy to return minimal answers would be to generate an answer A such that the induced graph G_A is a minimal Steiner tree over G_T connecting the literals in G_A that match the keywords. Indeed, computing a Steiner tree avoids including unnecessary edges to connect the nodes. However, we note that the minimal Steiner tree problem is NP-Complete, albeit there are known heuristics that generate approximate solutions, especially when G_T is a large graph (Li et al. 2016).

As an example, let K={*rocky*,*sylvester*,*stallone*} and G_T be the graph shown in Figure 5. Note that two triples match with keyword *rocky*, one triple matches with the keyword *sylvester* and two triples match with the keyword *stallone*. Hence, the set of data matches is:

$$\boldsymbol{M} = \{ (rocky, (r_3, :label, Rocky)), (rocky, (r_4, :label, Rocky\ V))$$
$$(sylvester, (r_1, :label, Sylvester\ Stallone)),$$
$$(stallone, (r_1, :label, Sylvester\ Stallone)),$$
$$(stallone, (r_2, :label, Sage\ Stallone)) \}$$

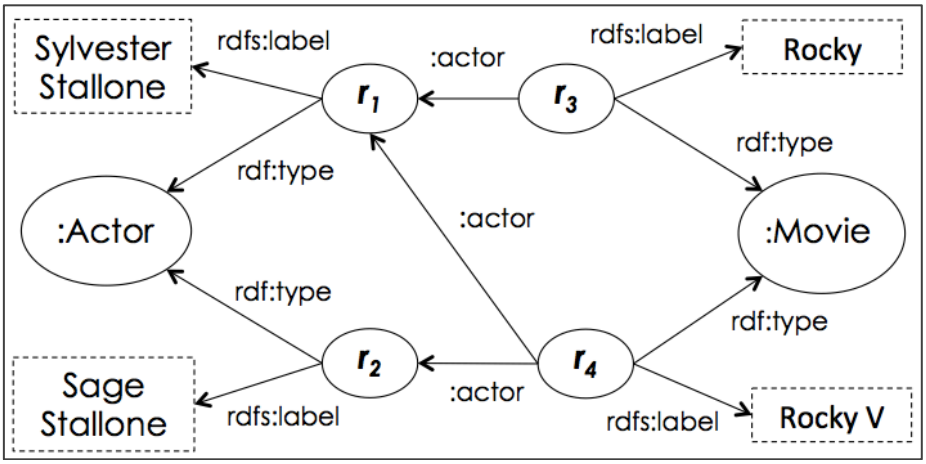


Figure 5: An RDF Graph.

Finally, Table 6 presents the set of possible answers for keyword query K={*rocky*,*sylvester*,*stallone*}. Note that A_1 to A_4 are *partial* matches because $K/A_n < K$ and A_5 to A_8 are *total* matches because $K/A_n = K$. Considering that A_5 and A_6 are smaller sets of triples than A_7 and A_8, and that there is a Steiner Tree connecting each triple in those sets, A_5 and A_6 are probably the answers the user would prefer. Then, we could use some ranking function to return the answers to the user ordered by relevance.

Table 6: Possible answers for *K*={*rocky*,*sylvester*,*stallone*}.

A_n	K/A_n
A_1 = {(r_3, *:label, Rocky*)}	*rocky*
A_2 = {(r_4, *:label, Rocky V*)}	*rocky*
A_3 = {(r_1, *:label, Sylvester Stallone*)}	*sylvester, stallone*
A_4 = {(r_2, *:label, Sage Stallone*)}	*stallone*
A_5 = {(r_3, *:label, Rocky*), (r_1, *:label, Sylvester Stallone*)}	*rocky,* *sylvester, stallone*
A_6 = {(r_4, *:label, Rocky V*), (r_1, *:label, Sylvester Stallone*)}	*rocky,* *sylvester, stallone*
A_7 = {(r_3, *:label, Rocky*), (r_1, *:label, Sylvester Stallone*), (r_2, *:label, Sage Stallone*)}	*rocky* *sylvester* *stallone*
A_8 = {(r_4, *:label, Rocky V*), (r_1, *:label, Sylvester Stallone*), (r_2, *:label, Sage Stallone*)}	*rocky,* *sylvester,* *stallone*

4.1.2.Overview of the Proposed Solution

In Section 4.1.1, we illustrated a naïve method that goes as follows: (1) generate all possible matches by searching the triples for each keyword; (2) generate all possible answers; (3) select the minimal answers; (4) generate all possible Steiner Trees; (5) rank the results. However, in a large and ambiguous dataset, this is not at all feasible. For instance, if we search IMDb looking for triples that have keyword *rocky* in some value, we find 9,600 triples, for keyword *sylvester* we find 4,237 triples and for keyword *stallone* we find 1,242 triples. This would generate billions of possible answers to be analyzed.

Therefore, we looked for strategies to minimize these problems. The first one is that we do not need to analyze the triples themselves, but we can group the matches at the level of classes and properties. By doing this, we can generate SPARQL templates that satisfy groups of classes/properties. Furthermore, we do not need to generate a Steiner tree between each collection of triples, but we can generate a Steiner tree at the schema level between collections of classes and, using the Steiner tree, synthesize SPARQL templates. By generating SPARQL templates and consequently SPARQL queries, we leave the responsibility for finding the actual instances, and paths between instances, to the RDF triple stores that were built for that. Hence, we classify the translation algorithm proposed in this research as *schema-based*.

Furthermore, we use the InfoRank measures to further reduce the number of matches of the task "finding pieces of information in the graph", as described in Section 4.2.1. Our strategy contrasts with a typical Web Information Retrieval system since it may not choose the triple that covers the maximum number of keywords as in the example of Section 4.1.1. Instead, we may prioritize exact matches with highly important instances. Furthermore, we do not treat the keywords as "bag of words", but we consider the order of appearance of the words in the query. For instance, for the keyword query K={*movie,rocky,character*} we assume that the user wants a *character* of a *movie* named *rocky*, however, if the query is K={*character,rocky,movie*}, we assume that the user wants a movie that has a *character* named *rocky*. This strategy was adopted after analyzing the queries in the keyword search benchmarks described in Section 6.

For the task “assembling the retrieved pieces of information to compose complete answers”, we use InfoRank to choose the best paths when building the Steiner Tree, as described in Section 4.2.2. Finally, for the task “ranking the complete answers”, we use the importance measure to order the SPARQL query and consider the importance of the instances retrieved in each SPARQL result, as described in Section 4.2.3.

4.2. Implementation and Examples

4.2.1. Finding Pieces of Information in a Graph

In this section, we present a greedy algorithm that takes keywords as input and returns the best set of class/property groups, as defined in Section 4.1.

In the first step of our strategy, we materialize group tables about metadata (classes and properties), and data (instances and values) that will help us finding the pieces of information. Note that these tables are computed only once, before execution time.

The group tables about classes and properties, respectively named *TMC* and *TMP*, can be computed using the following queries. Recall that we assume that the graph contains schema information and that the InfoRank scores are materialized in the graph.

Group Table for Classes - TMC

```
select ?class ?literal ?info
       (strlen(?literal) as ?length)
where
{ ?class rdf:type rdfs:Class .
  ?class rdfs:label ?literal .
  ?class :inforank ?info   }
```

Group Table for Properties - TMP

```
select distinct ?class ?property ?literal ?info
       (strlen(?literal) as ?length)
where
{  ?r rdf:type ?class .
   ?class rdf:type rdfs:Class .
   ?r ?property ?o .
   filter (?property != rdf:type)
   ?property rdfs:label ?literal .
   ?property :inforank ?info   }
```

Furthermore, we materialize two other tables about instance identifiers (i.e. *rdfs:label*, *foaf:name*, *dc:title*, etc.), named *TDI*, which can be computed with the following query.

```
                Group Table for Data Instances - TDI
select ?class ?property ?literal
       (sum(?inforank) as ?info)
       (strlen(?literal) as ?length)
       (count(*) as ?count)
where
{ ?r rdf:type ?classs .
  ?class rdf:type rdfs:Class .
  ?r ?property ?literal .
  filter ( ?property in (rdfs:label, foaf:name, dc:title … ))
  ?r :inforank ?inforank .
}
group by ?class ?property ?literal
```

Finally, the last table, named *TDV*, is about other data values (i.e. *sound mix*, *trivia*, etc.), which can be computed with the following query.

```
                 Group Table for Data Values - TDV
select ?class ?property ?literal
       (sum(?inforank) as ?info)
       (strlen(?literal) as ?length)
       (count(*) as ?count)
where
{ ?r rdf:type ?classs .
  ?class rdf:type rdfs:Class .
  ?r ?property ?literal .
  filter( ?property NOT in (rdfs:label, foaf:name, dc:title … ))
  filter( isLiteral(?literal))
  ?r :inforank ?inforank .
}
group by ?class ?property ?literal
```

Table 7 and Table 8 show examples of the group tables *TMC* and *TMP* considering an IMDb dataset. Likewise, Table 9 and Table 10 show examples of data instances and data values in the same dataset. The *count* column indicates that there are two movies named *The Sound of Music*, three actors named *Christopher Plummer*, 99,956 movies with a mono sound mix, and so on. The *info* column is the aggregation of the InfoRank scores of all resources of a given group. For instance, the two resources of group e_1 sum up to 0.00190 of InfoRank scores. The *length* column is the number of characters of the *literal* column (i.e. string length).

Table 7: Example of TMC from IMDb.

g	*class*	*literal*	*info*	*length*
c_1	*:Movie*	Movie	0.90673	5
c_2	*:Actress*	Actress	0.91476	7

Table 8: Example of TMP from IMDb.

g	*class*	*property*	*literal*	*info*	*length*
p_1	*:Movie*	*:sound_mix*	sound mix	0.00002	9
p_2	*:Movie*	*:trivia*	trivia	0.06817	6

Table 9: Example of TDI from IMDb.

g	*class*	*property*	*literal*	*info*	*length*	*count*
e_1	*:Movie*	*rdfs:label*	The Sound of Music	0.00190	18	2
e_2	*:Actress*	*rdfs:label*	Julie Andrews	0.00067	13	2
e_3	*:Actor*	*rdfs:label*	Christopher Plummer	0.00039	19	3

Table 10: Example of TDV from IMDb.

g	*class*	*property*	*literal*	*info*	*length*	*count*
v_1	*:Movie*	*:sound_mix*	mono	1.28063	4	99956
v_2	*:Movie*	*:trivia*	The gazebo used for the "Sixteen Going on Seventeen" and "Something Good" scenes can still be visited in the Salzburg area, on "Sound of Music" tours. However, the public had to be excluded from the interior because film fans who were considerably older than "sixteen going on seventeen" were injuring themselves while trying to dance along the seats. The gazebo in Austria was only used for exterior shots, the actual dance by Julie Andrews and Christopher Plummer was filmed on a replica located in L.A.	0.00190	505	1

Algorithm 2 presents (in an indented style) an overview of a greedy strategy to obtain the best set of groups that satisfy a keyword query *K*.

```
Algorithm 2: Greedy Strategy to return the best set of groups
Input: A keyword query K and the group tables TMC, TMP, TDI, TDV
Output: A subset of groups M

1.  J ← all keywords in K
2.  M ← ∅
3.  while J is not empty:
4.     c ← find in TMC a class with the highest accum given J,
           use the highest info/length to disambiguate
5.     if a match c is found:
6.        add c to M, remove the keywords in literal(c) from J
7.     if J did not change:
8.        break
9.  S ← ∅  #S will contain keywords
10. L ← ∅  #L will contain list of keywords
11. for k in K:
12.    if k is NOT in a class match of M:
13.       add k to S
14.    else if S is not empty:
15.       add S to L and clear S
16. if S is not empty:
17.    add S to L and clear S
18. for S in L:
19.   J ← all keywords in S
20.   while J is not empty:
21.      p ← find in TMP a property with the highest accum given
              J, use the highest info/length to disambiguate
             (filter by class if necessary)
22.      e ← find in TDI an instance with the highest accum
              given J, use the highest info/length to
              disambiguate (filter by class if necessary)
23.      if accum(J,literal(p)) >= accum(J,literal(e)):
24.         add p to M, remove the keywords in literal(p) from J
25.         continue
26.      if all keywords in J are matched by literal(e):
27.         add e to M, remove the keywords in literal(e) from J
28.         continue
29.      v ← find in TDV a value with the highest accum score
              given J, use the highest info/length to
              disambiguate (filter by class if necessary)
30.      if ( accum(J,literal(e)) * info(e)/length(e) >=
              accum(J,literal(v)) * info(v)/length(v) ):
31.         add e to M, remove the keywords in literal(e) from J
32.      else:
33.         add v to M, remove the keywords in literal(v) from J
34.      if J did not change:
35.         break
36. eliminate redundancies from M and return
```

Let *discrete*(*j*,*o*) be a score function that returns 100 if literal *o* contains keyword *j*, and 0, otherwise. Also, let *accum*(*J*,*o*) be a function that returns the accumulated score, given a set of keywords $J = \{j_1, j_2, \ldots, j_n\}$:

$$accum(J, o) = discrete(j_1, o) + \ \ldots + discrete(j_n, o) \qquad (12)$$

As an example, consider the keyword query $K=\{sound, music\}$. Using the data in Table 7 to Table 10, the discrete scores that return 100 are:

discrete(*sound*, *The Sound of Music*) = 100

discrete(*music*, *The Sound of Music*) = 100

discrete(*sound*, *sound mix*) = 100

Hence, the accumulated non-zero scores are:

accum({*sound*,*music*}, *The Sound of Music*) = 200

accum({*sound*,*music*}, *sound mix*) = 100

Note that functions *discrete* and *accum* are based on the Oracle Text[10] scoring functions, since we used Oracle 12c database to index and search for matches.

Let $K=\{movie,sound,of,music,actress,julie\}$ be our first running example. In the first step of the algorithm (lines 3 to 8), we try to find all class matches considering first the *accum* score, and then, disambiguating with the *info/length* score. The following query in Oracle shows how to find such matches in table TMC for our running example (note that we eliminate stop words, such as *of*, when computing the *accum* score).

```
select class, literal, info, score(1) as accum
from TMC
where contains(value, 'DEFINESCORE(movie,DISCRETE) accum
                       DEFINESCORE(sound,DISCRETE) accum
                       DEFINESCORE(music,DISCRETE) accum
                       DEFINESCORE(actress,DISCRETE) accum
                       DEFINESCORE(julie,DISCRETE)', 1) > 0
order by accum desc, info/length desc
```

Hence, the algorithm starts with $J=\{movie,sound,of,music,actress,julie\}$, and in the first iteration of the first *while* loop (line 3), it chooses class *Movie* since

[10] https://docs.oracle.com/cd/B28359_01/text.111/b28303/quicktour.htm#g1011793

it has a higher *info/length* score than class *Actress* (and both have the same *accum* score).

Furthermore, note that, besides considering the importance of the group given by the *info* score, we also consider how close the literal is to *J* by using the length of the string. For instance, exact matches, such as *movie/Movie*, have a higher priority than matches that contain other words, such as *movie*/*TV Movie*. The word count could probably better identify the closeness of the match, however, due to efficiency issues, we chose the length instead of the count.

Continuing to the second iteration, the algorithm chooses class *Actress*, given that *J*={~~*movie*~~,*sound*,*of*,*music*,*actress*,*julie*}. In the third iteration, the while loop breaks since it had no more matches (i.e., *J* did not change).

In the second step (lines 9 to 17), we split the keywords in sequences separated by the class matches. This is important so we give a meaning to the order of appearance of words, instead of treating the query as a "bag of words". For our running example, the sequences generated are S_1={*sound*,*of*,*music*} and S_2={*julie*}, and *L*={S_1,S_2}.

In the third step (lines 18 to 35), for each sequence, we try to match the keywords with properties, instances and values, and solve ambiguity problems.

Hence, in the first iteration of the *for* loop (line 18), *J*={*sound*,*of*,*music*}. We proceed by trying to find property matches in table *TMP* (line 21) using the following query.

```
select class, property, literal, info, score(1) as accum
from TMP
where contains(value, 'DEFINESCORE(sound,DISCRETE) accum
                      DEFINESCORE(music,DISCRETE)', 1) > 0
      and class = ':Movie'
order by accum desc, info/length desc
```

Note that the above query considers only properties that have class *Movie* as their domain. We are able to perform such filtering since we identify that there is a class match (*movie*) before the given sequence S_1={*sound*,*of*,*music*}, which again depends on the order of words. However, if the sequence has no preceding class match, we simply do not add the class filter to the *where* clause.

We continue by searching the instances (line 22) for the same sequence in order to check whether there are ambiguities. The following query shows how to

find instance matches in table *TDI*. Again, note that the query considers only instances of class *Movie*.

```
select class, property, literal, info, length, score(1) as accum
from TDI
where contains(value, 'DEFINESCORE(sound,DISCRETE) accum
                       DEFINESCORE(music,DISCRETE)', 1) > 0
      and class = ':Movie'
order by accum desc, info/length desc
```

Continuing with the running example for J={*sound*,*of*,*music*}, we found an ambiguity for keyword *sound* between property *sound mix* in table *TMP* and instance *The Sound of Music* in the table *TDI*. However, recall that

$$accum(\{sound,music\}, \textit{The Sound of Music}) = 200$$

$$accum(\{sound,music\}, \textit{sound mix}) = 100.$$

Hence, we solve this ambiguity by choosing instance *The Sound of Music*, since it has a higher *accum* score than property *sound mix*. Since all keywords in J are matched by the chosen instance, the *while* loop stops (recall that the *continue* statement[11] skips the remainder of the loop body and continues with the next iteration of the loop).

Finally, in the second iteration of the *for* loop, where J={*julie*}, the algorithm chooses instance *Julie Andrews* and stops. The result at the end of this loop is M={c_1, c_2, e_1, e_2}. However, since e_1 (i.e. instance *The Sound of Music*) and e_2 (i.e. instance *Julie Andrews*) were already filtered by c_1 (i.e. class *Movie*) and c_2 (i.e. class *Actress*), respectively, we eliminate the redundancies and the output is actually M={e_1, e_2} (line 36).

In the first running example we showed how to solve ambiguities between properties and instances matches. However, we still need to solve ambiguities between instances and values.

Let K={*julie*,*andrews*,*christopher*,*plummer*} be our second running example. Since there are no class and property matches, the algorithm goes straight to table *TDI* and successfully finds a match with group e_2 (see Table 11). However,

[11] https://en.wikipedia.org/wiki/Control_flow#Continuation_with_next_iteration

since it did not find all keywords (line 26), the algorithm proceeds to try other values (line 29). The following query shows how to match the keywords with table *TDV* for our running example.

```
select class, property, literal, info, length, score(1) as accum
from TDV
where contains(value, 'DEFINESCORE(julie,DISCRETE) accum
                       DEFINESCORE(andrews,DISCRETE) accum
                       DEFINESCORE(christopher,DISCRETE) accum
                       DEFINESCORE(plummer,DISCRETE)', 1) > 0
order by accum desc, info/length desc
```

Table 11: Matches for *K*={*julie*,*andrews*,*christopher*,*plummer*}.

g	*class*	*property*	*literal*	*info*	*length*
e_2	*:Actress*	*rdfs:label*	Julie Andrews	0.00067	13
v_2	*:Movie*	*:trivia*	(…) The gazebo in Austria was only used for exterior shots, the actual dance by Julie Andrews and Christopher Plummer was filmed on a replica located in L.A.	0.00190	505

After searching the values table, the algorithm finds a new match, which leads us to an ambiguity between groups v_2 and e_2 (again, see Table 11). Moreover, recall that

$$accum(\{julie,andrews,christopher,plummer\}, \textit{Julie Andrews}) = 200$$
$$accum(\{julie,andrews,christopher,plummer\}, \textit{The gazebo}....) = 400$$

In order to resolve this ambiguity, we use *accum* * *info*/*length* (line 30). Note that, although value v_2 contains the four keywords (i.e. higher *accum*), the algorithm actually chooses instance e_2 instead of value v_2 due to the *info/length* scores. Hence, in the second iteration of the *while* loop (line 20), where J={*christopher*,*plummer*}, the algorithm chooses instance e_3, and the final output is $M=\{e_2, e_3\}$.

A typical Web information retrieval system would probably prioritize value v_2 in the first iteration, since it covers all keywords. However, we argue that, in an RDF environment, we need to take advantage of the explicit relations indicated by the graph modeling. Hence, we rather prioritize single instance matches, such as e_2

and e_3, so that we can later present a subgraph that shows the relation between them. In this case, the graph would show the relation between *Julie Andrews* and *Christopher Plummer* through *The Sound of Music,* that is, the movie they co-acted.

Note that, if we present the sentence "… *the actual dance by Julie Andrews and Christopher Plummer was filmed on a replica located in L.A.*" to the user, the relation between *Julie Andrews* and *Christopher Plummer* is not clear, even though this sentence is a trivia of the movie *The Sound of Music*.

Finally, with the output from Algorithm 2, we can generate SPARQL templates that satisfy the retrieved groups. The resulting templates for *K*={*julie*,*andrews*,*christopher*,*plummer}* are shown in Table 12.

Table 12: Templates generated for *K*={*julie*,*andrews*,*christopher*,*plummer}*.

template	*interpretation*

`?r1 rdf:type :Actress .`
`?r1 rdfs:label ?o1 .`
`filter(contains(?o1, 'julie andrews'))`
	Instances of class *:Actress* that contain keywords *julie andrews* in their labels

`?r2 rdf:type :Actor .`
`?r2 rdfs:label ?o2 .`
`filter(contains(?o2, 'christopher plummer'))`
	Instances of class *:Actor* that contain keywords *christopher plummer* in their labels

4.2.2. Connecting Pieces of Information in a Graph

The second task of the *RDF-KwS* process, i.e., connecting pieces of information, consists of finding a minimal Steiner tree between the classes of the groups retrieved in the first task.

The Steiner tree problem consists in finding a minimal tree that spans a given subset of nodes of a given graph. The nodes that must be spanned by the tree are called terminals. However, the solution might include other nodes to connect these terminals. Given that the Steiner tree problem is NP-Complete, the classical approximated solution consists of three steps: (1) construct the metric closure of a graph *G*, that is, a complete graph in which each edge (*m,n*) is weighted by the shortest path distance between *m* and *n*, for each pair of nodes *m* and *n* of *G*; (2) select from the metric closure a subgraph that contains only the terminal nodes; (3) compute a minimal spanning tree on the subgraph.

In our solution, the Steiner tree is computed over the schema graph, a representation of the schema as the IMDb example in Figure 6. Note that, we can pre-compute the metric closure before running time and execute only steps two and three when the user submits a keyword query.

Let $G = (V,E)$ be an undirected graph, where the nodes are the classes of the RDF graph with the respective informativeness (as defined in Chapter 3) and the edges represent the connections between the classes through object properties. Note that there might be several properties connecting two classes in the RDF graph; however, we create the schema with only one edge between nodes. Moreover, we do not consider the direction of the edge when computing the Steiner Tree.

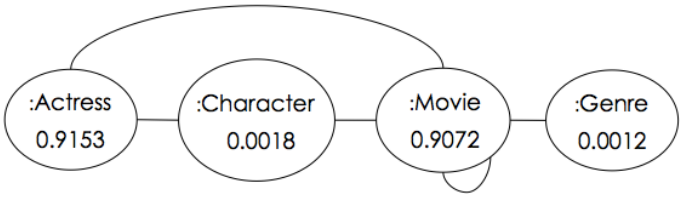


Figure 6: Example of the Schema Graph in IMDb.

The following query shows how to obtain the edges of the schema graph given an RDF graph, which are materialized in a table named *TSG*. Again, recall that we assume that the graph contains schema information and that the InfoRank scores are materialized in the graph.

Table for the Schema Graph - TSG

```
select distinct ?domain ?domain_info ?range ?range_info
where
{  ?r ?p ?o .
   ?r rdf:type ?domain .
   ?domain rdf:type rdfs:Class .
   ?o rdf:type ?range  .
   ?range rdf:type rdfs:Class .
   ?domain :inforank ?domain_info .
   ?range  :inforank ?range_info .
}
```

In the first step, we need to solve the *all pairs shortest path* problem in order to compute the metric closure of a graph *G*. This problem consists in finding the shortest paths between all pairs of nodes of the given graph. One known approach to this problem is to run Dijkstra's algorithm (Cormen et al. 2009) considering every node in the graph as source.

Algorithm 3 presents an adapted version of Dijkstra's algorithm that better fits our needs. The main difference is that we consider the informativeness of classes to choose the best path when there is a tie (line 18), instead of using edge weights as in a typical implementation of Dijkstra's algorithm.

Furthermore, in a typical implementation of Dijkstra's algorithm, the distance from the source node *s* to itself is always initialized with 0. However, in our strategy, this distance is only initialized with 0 (line 10) if the class has a self loop (e.g. class *Movie* in Figure 6). Otherwise, we initialize the distances between all neighbors of the source node with 1 (line 12). We will justify such modification with an example later on.

Algorithm 3: Dijkstra shortest path algorithm
Input: A graph *G* and a source node *s*
Output: The distance and the path from *s* to all other nodes of *G*

```
1.  Q ← ∅
2.  info ← the informativeness of the nodes in G
3.  for each node v in G:
4.     dist[v] ← ∞
5.     path[v] ← ∅
6      add v to Q
7.  for each neighbor n of s:
8.     add s and n to path[n]
9.     if n = s:
10.       dist[n] ← 0
11.    else:
12.       dist[n] ← 1
13. while Q is not empty:
14.    u ← node in Q with the smallest dist[u]
15.    remove u from Q
16.    for each neighbor v of u:
17.       alt ← dist[u] + 1
18.       if((alt < dist[v]) or
             (alt = dist[v] and info[u] > info[v])):
19.          dist[v] ← alt
20.          path[v] ← path[u] + v
21. return dist and path
```

We run Dijkstra's algorithm using every node of the graph as source. Then, we eliminate the redundancies and finally create the metric closure. Note that this is computed only once, before execution time. Table 13 presents an example of the metric closure for the schema graph in Figure 6.

In the next step to compute the Steiner tree, we select from the metric closure a subgraph with only edges that are incident to terminal nodes. Then, we

can calculate an approximation of the Steiner tree by solving the minimum spanning tree problem on the metric closure subgraph.

Table 13: Metric closure example of the classes in IMDb.

edge	*source*	*target*	*dist*	*path*
e_1	*:Genre*	*:Genre*	2	*:Genre*, *:Movie*, *:Genre*
e_2	*:Genre*	*:Movie*	1	*:Genre*, *:Movie*
e_3	*:Genre*	*:Character*	2	*:Genre*, *:Movie, :Character*
e_4	*:Genre*	*:Actress*	2	*:Genre*, *:Movie, :Actress*
e_5	*:Movie*	*:Movie*	0	*:Movie*, *:Movie*
e_6	*:Movie*	*:Character*	1	*:Movie, :Character*
e_7	*:Movie*	*:Actress*	1	*:Movie, :Actress*
e_8	*:Character*	*:Character*	2	*:Character, :Actress, :Character*
e_9	*:Character*	*:Actress*	1	*:Character, :Actress*
e_{10}	*:Actress*	*:Actress*	2	*:Actress*, *:Movie*, *:Actress*

Algorithm 4 shows how to solve this problem using Kruskal's algorithm and a data structure for disjoint sets (Cormen et al. 2009). Again, we adapted the algorithm to use the informativeness of the nodes to choose the best path (line 5).

```
Algorithm 4: Kruskal mininum spanning tree algorithm
Input: An undirected weighted graph G
Output: A mininum spanning tree
1.  function kruskal(G)
2.     X ← ∅
3.     for each node u in G:
4.        makeset(u)
5.     E = the edges of G sorted by asceding order of weight and
           descending order of informativeness
6.     for each edge {u,v} of E
7.        if find(u) ≠ find(v):
8.           add edge {u,v} to X
9.           union(u,v)
10. function makeset(x)  # create a single set containing just x
11.    π(x) ← x
12.    rank(x) ← 0
13. function find(x)     # to which set does x belong?
14.    while x ≠ π(x):
15.       x ← π(x)
16.    return x
17. function union(x,y)  # merge the sets containing x and y
18.    r_x ← find(x)
19.    r_y ← find(y)
20.    if rank(r_x) > rank(r_y)
21.       π(r_y) ← r_x
```

```
22.     else:
23.         π(r_x) ← r_y
24.         if rank(r_x) = rank(r_y):
25.             rank(r_y) ← rank(r_y) + 1
```

The algorithm starts by creating trees that contain only a single node of the graph (lines 3 and 4). Then, it repeatedly looks for the edge with lowest weight/higher informativeness that connects two different trees and, when it finds one, the two trees are merged (lines 6 to 9).

Let *K*={*julie, andrews*, *drama*, *movie*} be our first running example. The keywords *julie andrews* matches with an instance of class *Actress*, the keyword *drama* matches with an instance of class *Genre*, and keyword *movie* matches with class *Movie*. Recall that this is the output of "finding pieces of information in a graph", as described in Section 4.2.1. Hence, *L*={*:Actress*, *:Genre*, *:Movie*} are the terminal nodes that needs to be spanned in our solution. Figure 7 shows the subgraph of the metric closure of Table 13 that connects only terminal nodes in *L*.

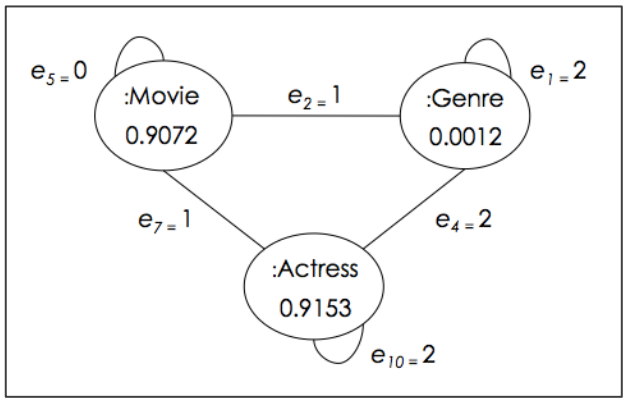


Figure 7: Metric closure sub graph for *L={:Actress,:Genre,:Movie}*.

In the first iteration of Kruskal's algorithm, the edge with the lowest weight is e_5; however, it does not connect two different trees. Then, the algorithm chooses edge e_7, since the combined informativeness of classes *:Movie* + *:Actress* is higher than the combined informativeness of classes *:Movie* + *:Genre*. Then, since edge e_7 connects two different trees, they are merged to compose a tree with nodes *:Movie* and *:Actress*. Finally, the algorithm chooses edge e_2 and stops, since all nodes were spanned.

The output of Kruskal's algorithm for terminal nodes *L* are edges e_7 and e_2. However, note that these are the edges of the metric closure; the final output of the Steiner tree requires that we "unpack" the path of each edge (see Table 13). In this

case, the output of the Steiner tree is the same as Kruskal's algorithm, given that the distances of the paths are 1, that is, there is no need to unpack the path.

Now, let *K*={*julie*, *andrews*, *anne*, *hathaway, comedy*} be our second running example. The keywords *julie* and *andrews* match with an instance of class *Actress*, keywords *anne* and *hathaway* match with another instance of class *Actress*, and keyword *comedy* matches with an instance of class *Genre*.

If we considered that the set of terminal nodes is *L1*={*:Actress*, *:Genre*} (i.e. only the distinct classes), the metric closure edges for these nodes would be those in Table 14. However, we actually want to identify different matches for different instances. Hence, the different class matches are identified with a number id, which lead us to *L2*={*:Actress/1*, *:Actress/2*, *:Genre*}, where *:Actress/1* is *Julie Andrews*, and *:Actress/2* is *Anne Hathaway*. Therefore, in order to achieve a proper solution for keyword query *K*, we expand the edges in Table 14 that have multiple class matches to compose the expanded metric closure edges for *L2*, which are presented in Table 15.

Table 14: Metric closure edges for *L1*={*:Actress*,*:Genre*}.

edge	*source*	*target*	*dist*	*path*
e_1	*:Genre*	*:Genre*	2	*:Genre*, *:Movie*, *:Genre*
e_4	*:Genre*	*:Actress*	2	*:Genre*, *:Movie, :Actress*
e_{10}	*:Actress*	*:Actress*	2	*:Actress*, *:Movie*, *:Actress*

Table 15: Metric closure edges for *L2*={*:Actress/1*,*:Actress/2*,*:Genre*}.

edge	*source*	*target*	*dist*	*path*
e_1	*:Genre*	*:Genre*	2	*:Genre*, *:Movie*, *:Genre*
$e_{4/1}$	*:Genre*	*:Actress/1*	2	*:Genre*, *:Movie, :Actress/1*
$e_{4/2}$	*:Genre*	*:Actress/2*	2	*:Genre*, *:Movie, :Actress/2*
$e_{10/1}$	*:Actress/1*	*:Actress/1*	2	*:Actress/1*, *:Movie*, *:Actress/2*
$e_{10/2}$	*:Actress/2*	*:Actress/2*	2	*:Actress/2*, *:Movie*, *:Actress/2*
$e_{10/3}$	*:Actress/1*	*:Actress/2*	2	*:Actress/1*, *:Movie*, *:Actress/2*

Figure 8 presents the graph of the edges in Table 15. Given that all edges have the same weight, Kruskal's algorithm first selects edge $e_{10/3}$, since the combined informativeness of *:Actress* + *:Actress* is higher than that of *:Actress* + *:Genre*, and the edge connects two different trees. Then, the algorithm may select either edge $e_{4/1}$ or $e_{4/2}$. Suppose that it selects $e_{4/1}$ and, hence, the output is $e_{10/3}$ and

$e_{4/1}$. In this example, we need to unpack the path of the edges in order to get the final Steiner tree. Hence, the edges generated from $e_{4/1}$ and $e_{10/3}$ are:

$$e_{4/1} \rightarrow s_1=\{:Genre, :Movie\}, s_2=\{:Movie\ :Actress/1\}$$

$$e_{10/1} \rightarrow s_3=\{:Actress/1, :Movie\}, s_4=\{:Movie\ :Actress/2\}$$

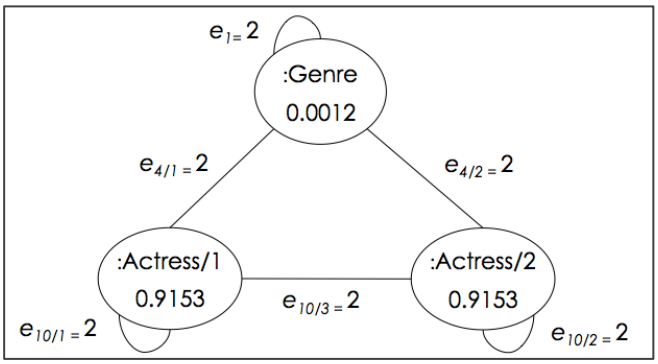


Figure 8: Metric closure sub graph for *L2={:Actress/1,:Actress/2,:Movie}*.

Note that when unpacking the paths, we include class *:Movie* (a non-terminal node) to connect the terminal nodes. This is a good example of why we need the modification in Dijkstra's algorithm. In its typical implementation, edge $e_{10/3}$ would have a distance of 0, and the path would not include class *:Movie*. This would lead to an incorrect schema representation indicating that class *:Actress* has a self loop, which consequently would generate an incorrect SPARQL template. Furthermore, recall that the metric closure is computed only once, before execution time, to avoid running Dijkstra's algorithm every time a user executes a keyword query. Finally, note that edge s_3 is a duplicate of edge s_2, given that we consider the graph as undirected. Thus, the final output of the Steiner tree is a graph with edges s_1, s_2 and s_4.

After computing the Steiner tree, we are able to create another SPARQL template indicating how the instances of classes are connected. Table 16 shows the templates generated for the running example at the end of this step.

Table 16: Templates generated for *K={julie,andrews,anne,hathaway,genre}*.

template	*interpretation*

`?r1 rdf:type :Actress .`
`?r1 rdfs:label ?o1 .`
`filter(contains(?o1, 'julie andrews'))`
	Instances of class *:Actress* that contains keywords *julie andrews* in their labels

`?r2 rdf:type :Actress .`
`?r2 rdfs:label ?o2 .`
`filter(contains(?o2, 'anne hathaway'))`
	Instances of class *:Actress* that contains keywords *anne hathaway* in their labels

`?r3 rdf:type :Genre .`
`?r3 rdfs:label ?o3 .`
`filter(contains(?o3, 'comedy'))`
	Instances of class *:Genre* that contains keyword *comedy* in their labels

`?r4 rdf:type :Movie .`	Instances of class *:Movie*

`?r4 ?p1 ?r1 .`
`?r4 ?p2 ?r2 .`
`?r4 ?p3 ?r3 .`
	How these instances are connected together

4.2.3. Ranking Information in a Graph

The third task of the Keyword Search over RDF Graphs process consists in ranking the retrieved answers of a keyword query. In our solution, this task is simple as creating new templates to get the InfoRank scores divided by the length of the string, and aggregate them in an *order by* clause. Again, recall that we materialized the scores as triples of the graph. Hence, continuing the example in Table 16, the final SPARQL query for K={*julie*, *andrews*, *anne*, *hathaway, comedy*} is presented in the following.

```
select *
where
{
   # Matches templates
   ?r1 rdf:type :Actress .
   ?r1 rdfs:label ?o1 .
   filter(contains(?o1, 'julie andrews'))
   ?r2 rdf:type :Actress .
   ?r2 rdfs:label ?o2 .
   filter(contains(?o2, 'anne hathaway'))
   ?r3 rdf:type :Genre .
   ?r3 rdfs:label ?o3 .
   filter(contains(?o3, 'comedy'))
   ?r4 rdf:type :Movie .
   ?r4 rdfs:label ?o4 .

   # Steiner Tree templates
   ?r4 ?p1 ?r1 .
   ?r4 ?p2 ?r2 .
   ?r4 ?p3 ?r3 .

   # Ranking templates
   ?r1 :inforank ?s1 .
   ?r2 :inforank ?s2 .
   ?r3 :inforank ?s3 .
   ?r4 :inforank ?s4 .
}
order by desc (?s1/strlen(?o1) +
               ?s2/strlen(?o2) +
               ?s3/strlen(?o3) +
```

```
                ?s4/strlen(?o4) )
```

4.3.Chapter Conclusion

In this chapter, we presented a solution to the Keyword Search over RDF Graphs problem. For the first task of the solution, we introduced a greedy strategy that, given a keyword query, finds matches (pieces of information) in a graph. The strategy also groups the matches at the level of classes and properties, instead of processing triple by triple, and generates SPARQL templates with patterns that satisfy these groups. Furthermore, the greedy strategy uses InfoRank to reduce the number of matches and to prioritize highly important instances when there is ambiguity. It also considers the order of appearance of the keywords, instead of treating it as a "bag of words". For the second task, we defined a strategy that, given the classes retrieved from the first task, finds a Steiner tree in a graph that represents the schema of an RDF dataset. This task uses the informativeness of classes to choose the best path when there is also an ambiguity. Likewise, it generates SPARQL templates with patterns that satisfy the Steiner tree. Finally, for the third task, we showed how to create a few more templates with InfoRank patterns to rank the matches in a SPARQL query.

5
The QUIRA Keyword Search System

This chapter presents the technical aspects of QUIRA (*QUerying with InfoRAnk*), a keyword search system that implements the strategy proposed in Chapter 4. First, we describe the architecture of QUIRA by showing the interaction between its processes and the database. Then, we present an interface with the following functions: (1) it allows the user to submit keyword queries; (2) it presents to the user an answer of a keyword query over an RDF graph; (3) it allows the user to evaluate an answer; (4) it allows the user to navigate through URIs; (5) it describes an URI.

5.1. Architecture

Figure 9 summarizes the architecture of QUIRA, in which the green squares represent the processes and the inputs are enumerated in sequence. Also, we describe the integration of QUIRA with the database, which has the RDF dataset and the pre-materialized auxiliary tables described in Section 4.2.

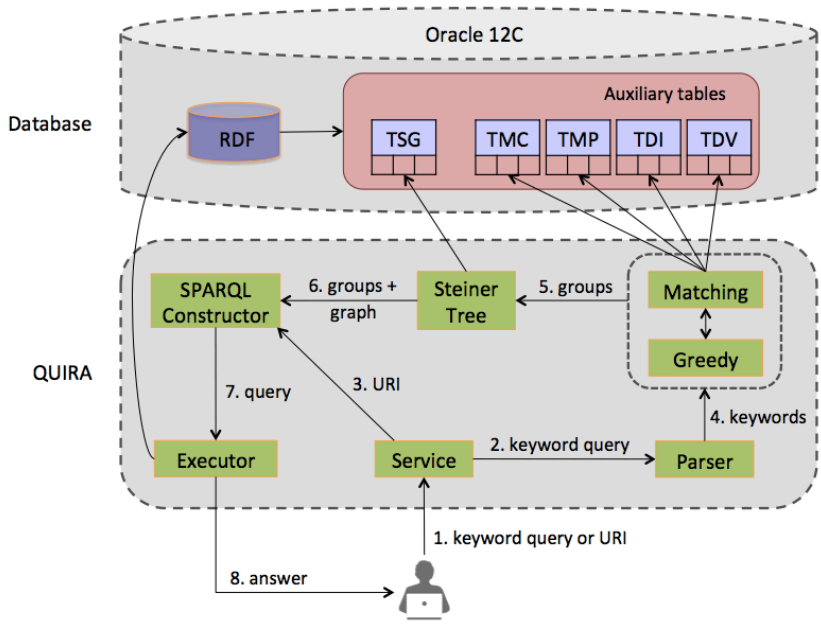


Figure 9: QUIRA's Architecture.

The execution flow goes as follows:

1. The user submits a keyword query or a URI.
2. If the input is a keyword query, the *Service* process redirects it to the *Parser* process.
3. If the input is a URI, *Service* redirects it directly to the *SPARQL Constructor*.
4. *Parser* splits the keyword query into a list of keywords and identifies which ones are stop words according to the default list of English stop words[12]. Then, the keywords are redirected to the *Greedy/Matching* processes.
5. The *Greedy* and the *Matching* processes are executed together as described in Section 4.2.1. *Matching* is responsible for accessing the auxiliary tables *TMC*, *TMP*, *TDI* and *TDV* to find the matches with the keywords. *Greedy* is responsible for handling the matches and for disambiguating the matches found, when required. The groups generated in this process are redirected to the Steiner Tree process.
6. The *Steiner Tree* process is responsible for the strategy described in Section 4.2.2. The output is a Steiner tree over the schema graph and the groups themselves, which are redirected to the *SPARQL Constructor* process.
7. According to the type of input, the *SPARQL Constructor* process is responsible for synthesizing a *SPARQL* query, which is redirected to the *Executor* process.
8. The *Executor* process is simply responsible for executing the SPARQL query in the RDF dataset and redirecting the answers to the user.

Note that, when the input of the *SPARQL Constructor* process are the groups and the schema graph, the synthesized query is similar to that presented in Section 4.2.3. However, when the input is an URI, the query is formulated to retrieve the information of that given URI. For example, the URI of the movie *The Sound of*

[12] https://www.ranks.nl/stopwords

Music is <https://www.imdb.com/title/tt0059742>; hence, if the user submits this specific URI, the generated SPARQL query is as follows:

```
select ?topic ?property ?value ?label
where
{ # Literals of the instance
  { select ("Information" as ?topic) ?property ?value ?label
    where { <https://www.imdb.com/title/tt0059742> ?p  ?value .
            ?p rdfs:label ?property .
            FILTER ( isLiteral(?value) ) }
    order by ?property ?value
  }
  union
  # Connections to other instances through outgoing edges
  { select ?topic ?property ?value ?label
    where
    { <https://www.imdb.com/title/tt0059742> ?p  ?value .
      ?value rdf:type  ?g .
      ?value rdfs:label ?label .
      ?p rdfs:label ?property .
      ?g rdfs:label ?topic .
      filter not exists {?g rdfs:subClassOf ?superClass }
    }
    order by ?topic ?property ?label
  }
  union
  # Connections to other instances through incoming edges
  { select ?topic ?property ?value ?label
    where
    { ?value ?p <https://www.imdb.com/title/tt0059742> .
      ?value rdf:type  ?g.
      ?value rdfs:label ?label .
      ?p rdfs:label ?property .
      ?g rdfs:label ?topic .
      filter not exists {?g rdfs:subClassOf ?superClass }
    }
    order by ?topic ?property ?label
  }
}
```

The next section presents the result of this SPARQL query at the interface when the user wants to describe a URI.

5.2. Interface

The main requirements of a keyword search interface are a text box where the user types keywords and a layout area to present the answers to the user. When dealing with an RDF graph, one could naturally consider presenting the results as graphs, given that answers are subgraphs of the RDF graph. However, there may be several

possible answers, and one single answer may contain several nodes and edges. Hence, we chose a table layout to present the retrieved answers, as shown in Figure 10. Note that there is an indication of the *Page* and a *Next* button so that the user can see all the results. Moreover, users are more familiar with table or list presentations than with graph presentations.

Furthermore, given that a table layout may lose information about how instances are connected in the graph, we allow the user to see the schema behind the answer by clicking on the graph icon, as shown in Figure 11.

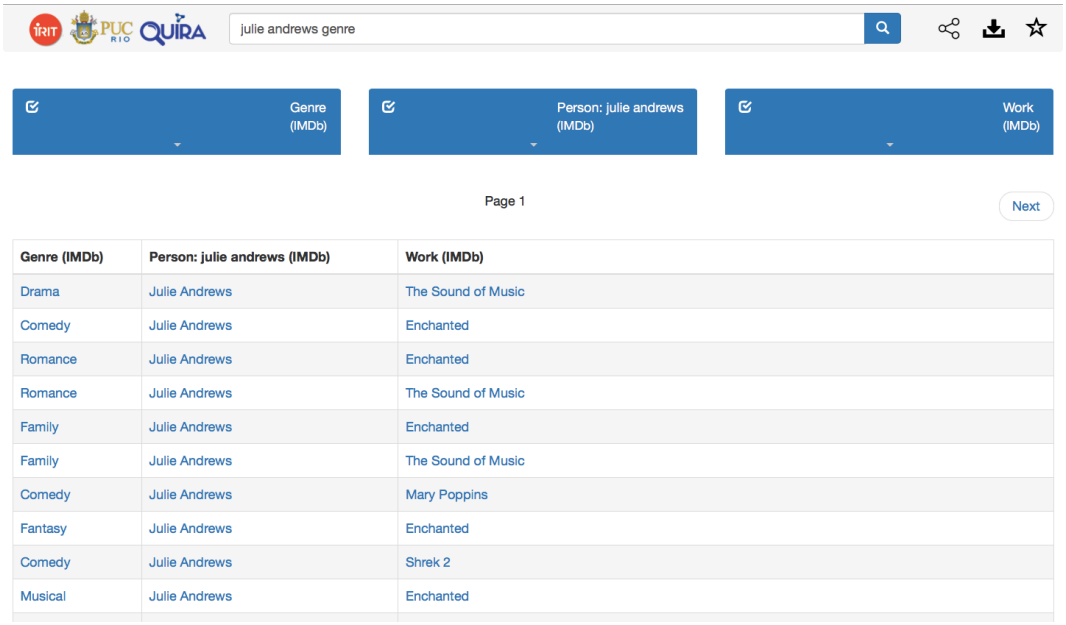


Figure 10: Query submission and answer.

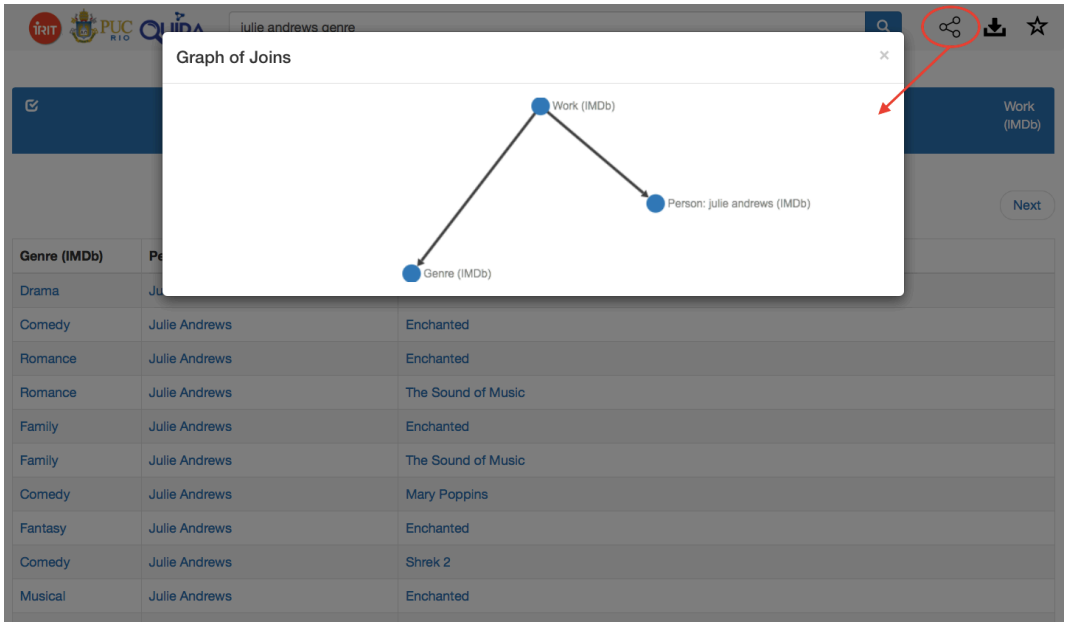
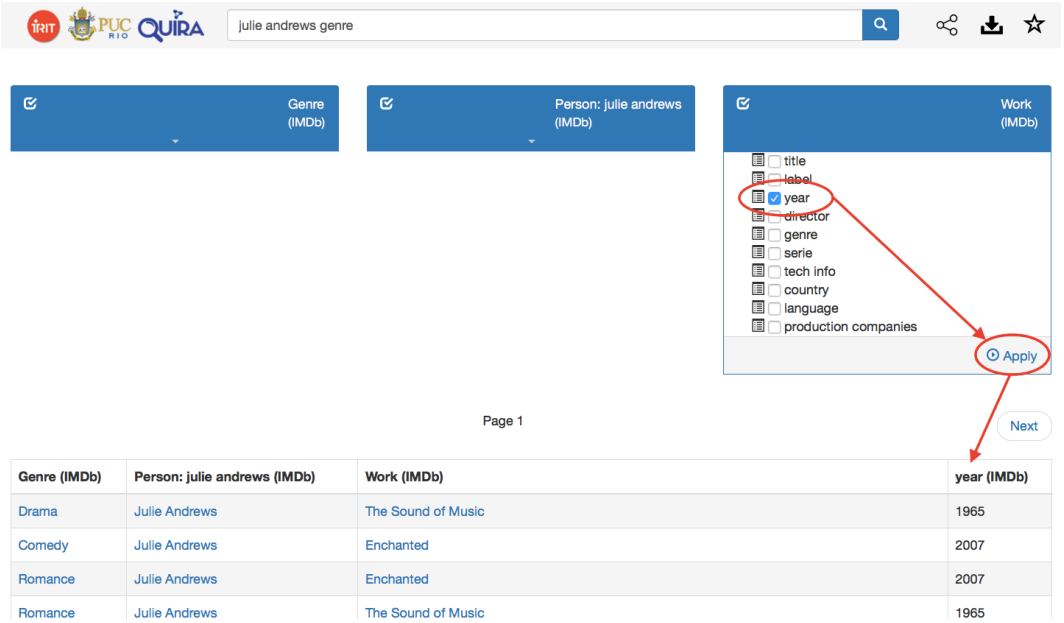


Figure 11: Graph schema for an answer.

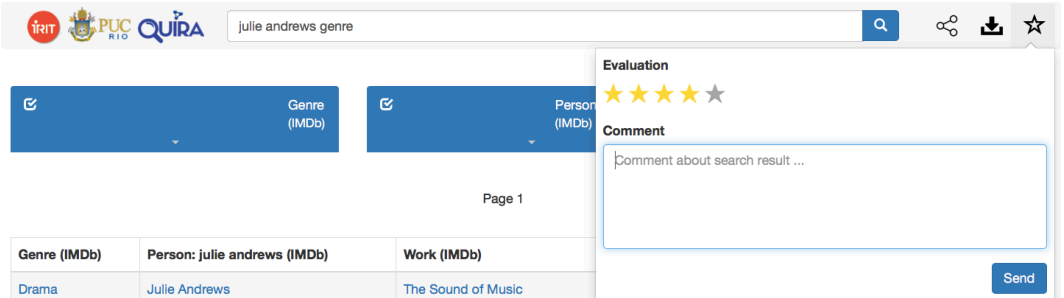
Another feature of the presentation is to show to the user the classes that compose the answers, to allow her to see the available properties of these classes, and to select those that she wants to include in the table. Figure 12 shows an example of this feature, in which a user wants to include the year of the retrieved works.

The last feature concerning presentation is the possibility of the user to evaluate the retrieved answer giving 1 to 5 starts, and also to provide a comment to help improve QUIRA's results (see Figure 13).



Genre (IMDb)	Person: julie andrews (IMDb)	Work (IMDb)	year (IMDb)
Drama	Julie Andrews	The Sound of Music	1965
Comedy	Julie Andrews	Enchanted	2007
Romance	Julie Andrews	Enchanted	2007
Romance	Julie Andrews	The Sound of Music	1965

Figure 12: Property selection.



Genre (IMDb)	Person: julie andrews (IMDb)	Work (IMDb)
Drama	Julie Andrews	The Sound of Music
Comedy	Julie Andrews	Enchanted
Romance	Julie Andrews	Enchanted
Romance	Julie Andrews	The Sound of Music
Family	Julie Andrews	Enchanted
Family	Julie Andrews	The Sound of Music
Comedy	Julie Andrews	Mary Poppins
Fantasy	Julie Andrews	Enchanted
Comedy	Julie Andrews	Shrek 2
Musical	Julie Andrews	Enchanted

Figure 13: Answer evaluation.

Besides the keyword search engine, we also allow the user to navigate to an instance by clicking on the URI, and see its data (Figure 14) and its relations with other instances (Figure 15). The user may continue navigating through other instances to discover more data. Note that the work *Everyone Says I Love* is connected to the work *The Sound of Music* through property *referenced in*; hence, the user can navigate to the former work to see more data, as shown in Figure 16.

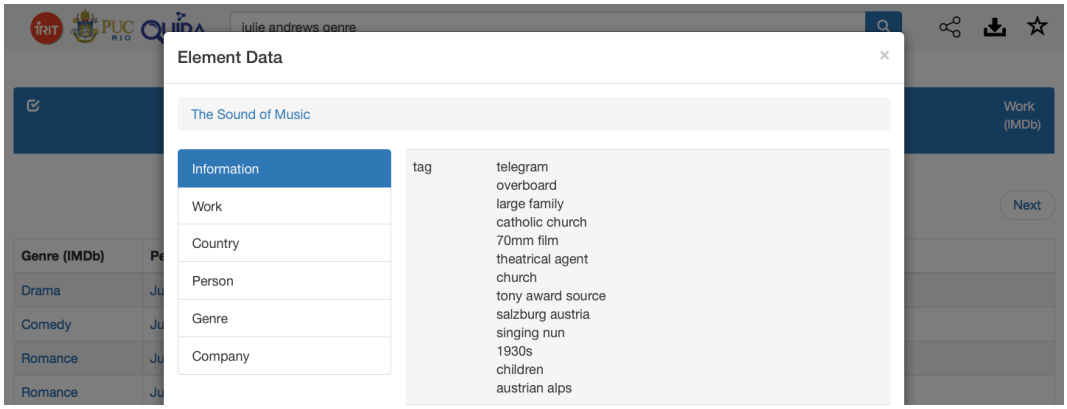


Figure 14: URI information.

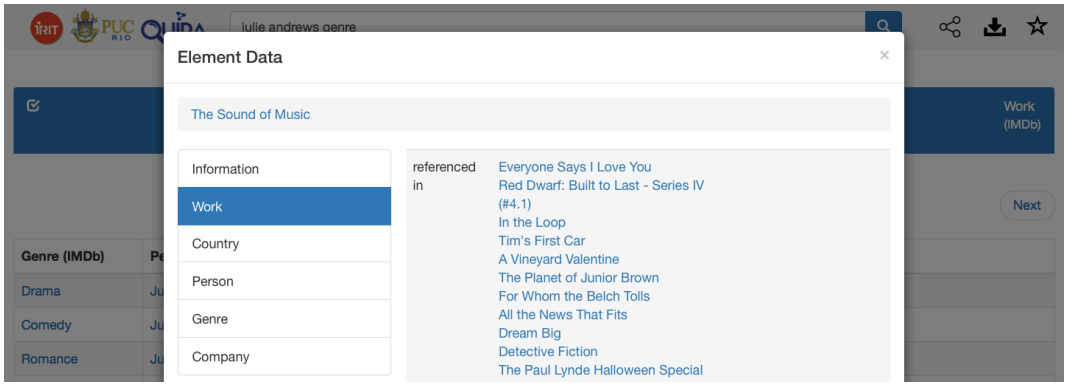


Figure 15: URI relations.

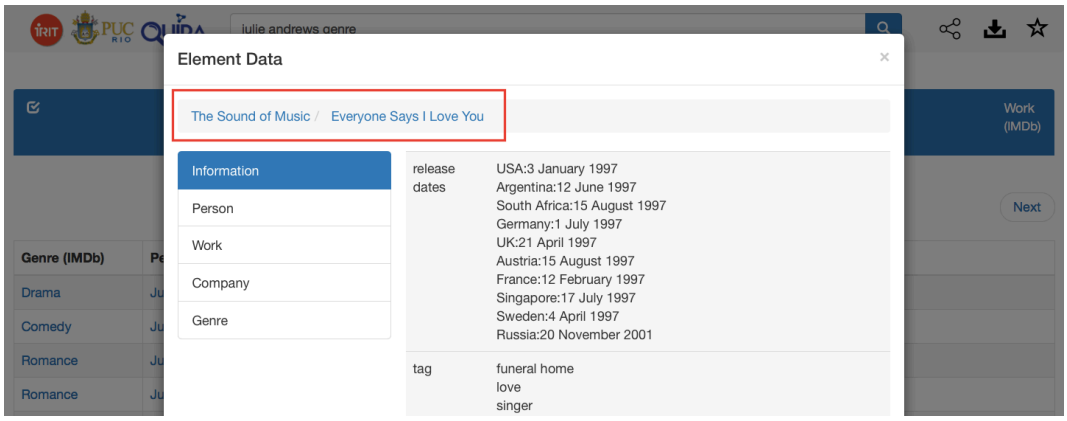


Figure 16: URI navigation.

5.3.Chapter Conclusion

In this chapter, we presented the architecture and interface of the QUIRA system. The architecture reflects the steps of the keyword search process, as defined in Chapter 4. Also, we introduced a new feature in which the user is able to submit a URI in order to get more data about the respective instance. Finally, the interface permits the user to submit queries, analyze answers and navigate through the instances of the graph.

6
Evaluation

This chapter presents an evaluation of the strategies proposed in this thesis using the IMDb and MusicBrainz RDF datasets. First, we show experiments to test the effectiveness of InfoRank as an importance measure for RDF graphs, when compared to other popular measures, such as PageRank. Second, given two popular keyword benchmarks over IMDb and MusicBrainz, we evaluate the quality of the results of QUIRA, a system for Keyword Search over RDF Graphs that uses InfoRank.

6.1. Setup

In order to evaluate our strategy, we downloaded the relational IMDb[13] dataset in MySQL and used Oracle 12c to transform it to RDF via R2RML. We used an RDF dump of MusicBrainz[14] as our second dataset and enriched it with DBpedia data, since the dump was incomplete.

Figure 17 and Figure 18 show an overview of the RDF schemas. The IMDb schema has 24 classes, 92 datatype properties, 37 object properties, and a total of 238,778,487 triples. The MusicBrainz schema has 8 classes, 13 datatype properties, 12 object properties, and a total of 212,948,635 triples.

Note that, since MusicBrainz did not include schema data, we performed a pre-processing step in order to capture such data. Hence, the following queries retrieve triples with rdfs:Class, rdf:Property, rdfs:domain and rdfs:range, as well as the rdfs:label of classes and properties generated from the respective URI. Hence, the only required schema properties are *rdf:type* and *rdfs:subClassOf*.

[13] https://sites.google.com/site/ontopiswc13/home/imdb-mo
[14] http://www.linkedbrainz.org

```
# Generates the rdfs:Class triples
insert { ?class rdf:type rdfs:Class }
where
{ select distinct ?class
  where { ?r rdf:type ?class}
}
# Generates the rdf:Property triples
insert { ?property rdf:type rdf:Property }
where
{ select distinct ?property
  where { ?r ?property ?o .
          filter (?property not in (rdf:type,rdfs:label,…)) }
}
# Generates the rdfs:domain and rdfs:range triples
insert { ?property rdfs:domain ?domain .
         ?property rdfs:range  ?range  }
where
{ select distinct ?property ?domain ?range
  where
  {  ?r ?property ?o .
     ?property rdf:type rdf:Property .
     ?r rdf:type ?domain .
     ?o rdf:type ?range  .
  }
}
# Generates the labels of classes from URI
insert { ?class rdfs:label ?label }
where
{ select ?class
         replace(replace(replace(str(?class),
                 ".*(#|/)", ""),
                 "([^_])([A-Z])","$1_$2"),
                 "_"," ") as ?label
  where { ?class rdf:type rdfs:Class }
}
# Generates the labels of properties from URI
insert { ?property rdfs:label ?label }
where
{ select ?property
         lcase(replace(replace(replace(str(?property),
                 ".*(#|/)", ""),
                 "([^_])([A-Z])","$1_$2"),
                 "_"," ")) as ?label
  where { ?property rdf:type rdf:Property }
}
```

All experiments were conducted using a RESTful Web application developed in Java. The app ran on a macOS Sierra, 1,7 GHz Intel Core i5 RAM 4 GB. To store and manage the RDF data, we used Oracle 12c, running on a 2x deca-

core Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz, 128GB RAM, 32KB Cache L1.

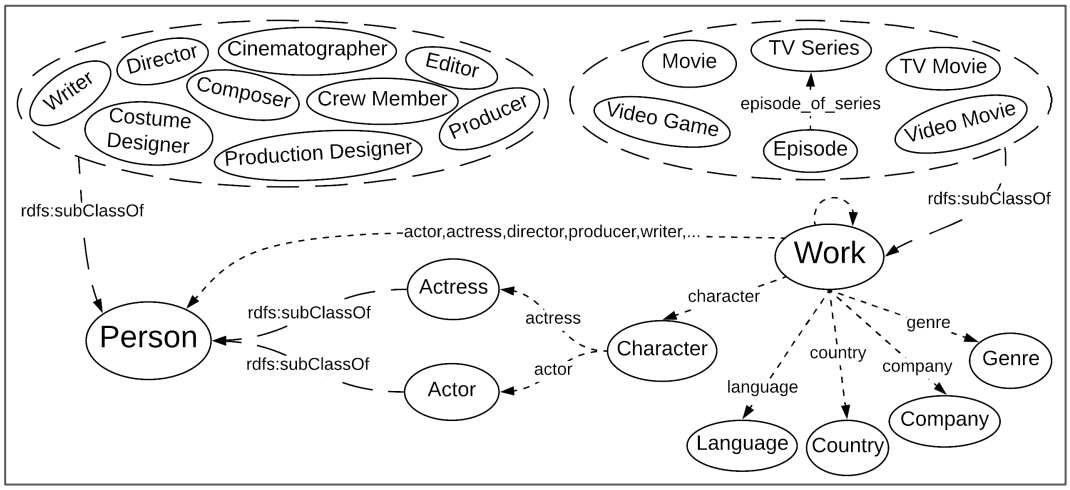


Figure 17: Overview of the IMDb Schema

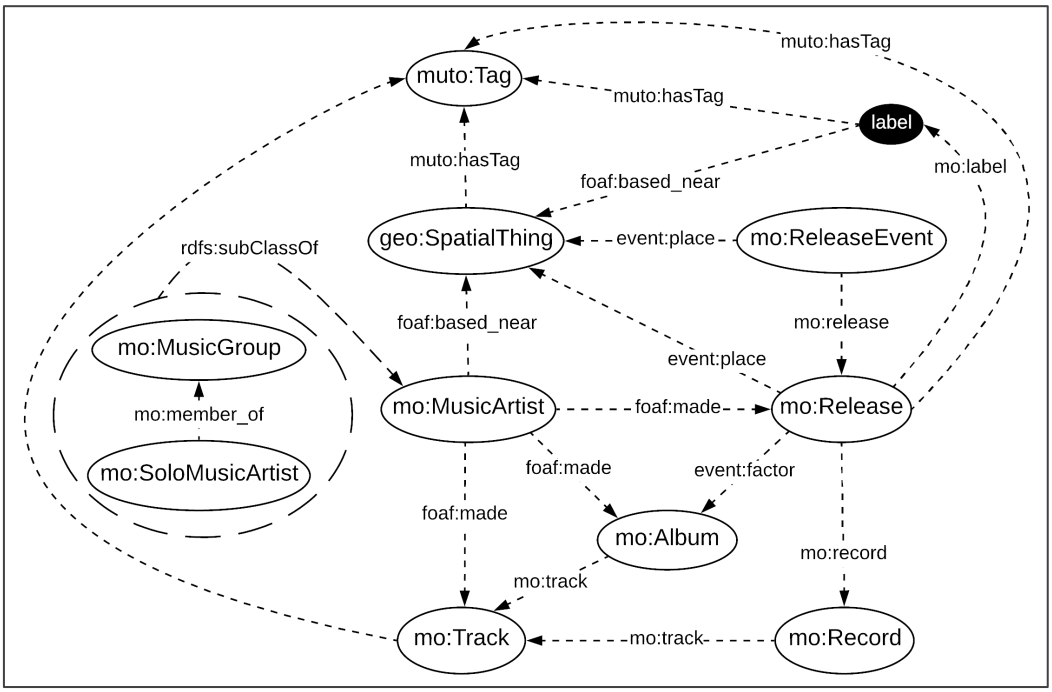


Figure 18: Overview of the MusicBrainz Schema

The datasets, benchmarks, and a detailed description of the experiments with printouts and queries are available at the QUIRA Web page[15].

[15] https://sites.google.com/view/quira/

6.2. Ranking Experiments

6.2.1. IMDb

This section presents results that assess InfoRank in the IMDb dataset to check whether it would be feasible to use these measures in a keyword search system.

Table 17 presents the ranking of the IMDb classes (super classes only) using the *absolute informativeness* (*InfoMax*) of classes defined in Section 3.1. Recall that we use the maximum informativeness of the instances of the given class. Hence, to show why we chose such metric, Table 17 also presents the summation (*InfoSum*) and the average (*InfoMax*) of the informativeness of instances. We argue that, in the IMDb dataset, the most important classes are those that represent works (movies, TV series, etc.) and people (actors, actresses, directors, etc.), which is the result that *InfoMax* and *InfoAvg* give. Note that, if we ranked classes using the node degree or *InfoSum*, the first in the ranking would be class *Character*. However, an IMDb user is probably more interested in movies (and other types of works) and movie stars, rather than in characters.

Table 18 presents the ranking of the IMDb object properties also using the *informativeness* of properties, as defined in Section 3.1. Note that the top properties are those connecting movies. For instance, the triple (:*StarWars_EpisodeII*, *:follows*, :*StarWars_EpisodeI*) indicates that the movie *Star Wars: Episode II* is a sequence of the movie *Star Wars: Episode I*. We can also find the inverse property *followed_by*, expressed by the triple (*:StarWars_EpisodeI*, *:followed_by*, *:StarWars_EpisodeII*).

Table 19 and Table 20 show the top 10 instances induced by InfoRank and PageRank, respectively. With PageRank, the top instances are highly connected nodes, such as countries, language and genres. However, we argue that, when considering a movies dataset, we would expect as top instances popular movies, series, actors, actresses, etc.

To indicate popularity, the tables also show the users' rating of works extracted from the IMDb Web site. In the case of a person, we extracted the best rated work in which she starred, directed, produced, etc. InfoRank's results show highly rated work/person, such as *Star Wars*, *The Wizard of Oz*, *Titanic* and *Morgan Freeman*. The results show some TV Series with lower rates because they have a

considerable level of informativeness (*General Hospital* – 375 literals; *Days of Our Lives* – 232 literals), and also a high degree through property *:episode_of_series,* since they have been on air for a long time. Likewise, the results show some hosts from TV Shows that also have been on air for a long time. Although InfoRank results show a few less popular works/people, we argue that InfoRank results correspond better to what users would expect in an IMDb dataset.

Table 17: IMDb class ranking computed by InfoRank.

Rank	Class	InfoMax	InfoAvg	InfoSum	Degree
1	imdb:Work	1619	11.06	24,250,376	2,410,207
2	imdb:Person	1482	4.57	13,987,247	3,913,018
3	imdb:Character	3	2.37	26,736,416	19,419,994
4	imdb:Company	3	3.00	449,942	224,971
5	imdb:Language	2	2.00	364	364
6	imdb:Country	2	2.00	46	319
7	imdb:Genre	2	2.00	319	46

Table 18: IMDb object property ranking computed by InfoRank.

Rank	Property	Info	Degree
1	imdb:follows	1.000	332,551
2	imdb:followed_by	1.000	332,548
3	imdb:edited_from	1.000	14,103
4	imdb:edited_into	1.000	14,103
5	imdb:referenced_in	0.985	223,535
6	imdb:references	0.985	223,532
	….		

Table 19: IMDb top 10 instances induced by InfoRank.

Rank	Instance	Class	Rating
1	Star Wars	imdb:Movie	8.6
2	Dolly Parton	imdb:Actress	6.8
3	Jay Leno	imdb:Actor	5.3
4	Morgan Freeman	imdb:Actor	8.6
5	The Wizard of Oz	imdb:Movie	8.0
6	General Hospital	imdb:TV Series	6.7
7	Days of Our Lives	imdb:TV Series	5.3
8	Bob Barker	imdb:Actor	7.7
9	Titanic	imdb:Movie	7.8
10	Around the World in Eighty Days	imdb:Movie	6.8

Table 20: IMDb top 10 instances induced by PageRank.

Rank	Instance	Class	Rating
1	English	imdb:Language	-
2	United States	imdb:Country	-
3	Short	imdb:Genre	-
4	Drama	imdb:Genre	-
5	Comedy	imdb:Genre	-
6	Documentary	imdb:Genre	-
7	UK	imdb:Country	-
8	Spanish	imdb:Language	-
9	German	imdb:Language	-
10	France	imdb:Country	-

6.2.2. MusicBrainz

This section presents results that assess InfoRank in the MusicBrainz dataset.

Table 21 presents the ranking of the MusicBrainz classes (super classes only) using their respective *absolute informativeness*. We argue that, in a MusicBrainz dataset, the most important classes are those representing music artists, tracks and albums, which is the result that *InfoMax* and *InfoAvg* gives. Note that, if we ranked classes using the node degree, the ranking would be *Track*, *Record*, *Release*, *Release Event*, *Album*, *Music Artist*. However, a user is probably more interested in the artists themselves than in events. Also, if we ranked using *InfoSum*, class *Record* would come before class *Album*.

Table 21: MusicBrainz class ranking computed by InfoRank.

Rank	Class	InfoMax	InfoAvg	InfoSum	Degree
1	mo:MusicArtist	1407	4.92	4,419,477	1,126,024
2	mo:Album	800	3.39	3,634,901	1,523,406
3	mo:Track	162	3.97	53,866,908	18,153,577
4	mo:Release	73	2.16	2,261,498	1,954,557
5	mo:Record	4	2.91	4,099,572	2,150,137
6	geo:SpatialThing	4	2.00	8,850	8,819
7	mo:ReleaseEvent	2	2.00	1,606,050	1,606,089
8	muto:Tag	2	2.00	65,772	65,772

Likewise, Table 22 shows the ranking of the MusicBrainz object properties also using their respective informativeness. The highest informative property is

foaf:made, which connects artists with their respective work (tracks, albums and releases). The top property considering the degree is *mo:track*, which connects tracks with their respective albums and records, and the second one is *foaf:made*. In this case, we consider that both *Info* and *Degree* return relevant results.

Table 22: MusicBrainz object property ranking computed by InfoRank.

Rank	Property	Info	Degree
1	foaf:made	1.0000	25,163,691
2	foaf:based_near	0.9979	628,359
3	muto:hasTag	0.7007	1,802,421
4	mo:track	0.5979	43,613,185
5	mo:member_of	0.5901	1,863,05
6	event:factor	0.5674	1,908,578
7	mo:label	0.0773	1,435,813
8	mo:record	0.0525	2,150,137
9	mo:release	0.0518	1,606,089
10	event:place	0.0014	1,652,066

Table 23 and Table 24 show the top 10 instances induced by InfoRank and PageRank, respectively. In PageRank, the majority of the top instances are countries from class *geo:SpatialThing*, again because they are nodes with a high degree. The *Various Artists* instance is used when a track is recorded by several artists. However, in the case of a music dataset, we argue that a user would expect famous artists, songs or albums as top instances. In the case of MusicBrainz, we extracted the number of listeners of artists from Last.fm[16] as an importance indicator. Hence, note that InfoRank returns famous musicians, such as *Elvis Presley*, *Mozart*, *Beethoven*, *Bob Dylan*, etc. Again, we argue that InfoRank shows results that correspond better to the expectation of a user searching for music related instances.

To summarize, these preliminary experiments suggested that InfoRank would provide better ranking results than PageRank when used in a keyword search system, which is confirmed in the following section.

Table 23: MusicBrainz top 10 instances induced by InfoRank.

Rank	Instance	Class	Listeners

[16] https://www.last.fm

1	Various Artists	mo:MusicArtist	-
2	Elvis Presley	mo:MusicArtist	2,4 mi
3	Johann Sebastian Bach	mo:MusicArtist	1,5 mi
4	Wolfgang Amadeus Mozart	mo:MusicArtist	1,6 mi
5	Willie Nelson	mo:MusicArtist	0,9 mi
6	Frank Sinatra	mo:MusicArtist	2,3 mi
7	Ludwig van Beethoven	mo:MusicArtist	1,8 mi
8	Bruce Springsteen	mo:MusicArtist	2,2 mi
9	Bob Dylan	mo:MusicArtist	2,6 mi
10	Pyotr Ilyich Tchaikovsky	mo:MusicArtist	0,9 mi

Table 24: MusicBrainz top 10 instances induced by PageRank.

Rank	Instance	Class	Listeners
1	United States	geo:SpatialThing	-
2	United Kingdom	geo:SpatialThing	-
3	Various Artists	mo:MusicArtist	-
4	Germany	geo:SpatialThing	-
5	Japan	geo:SpatialThing	-
6	France	geo:SpatialThing	-
7	Canada	geo:SpatialThing	-
8	Italy	geo:SpatialThing	-
9	Netherlands	geo:SpatialThing	-
10	Australia	geo:SpatialThing	-

6.3.Keyword Search Experiments

6.3.1. IMDb

To evaluate the impact of using InfoRank in a keyword search system over IMDb, we used all 50 queries (adapted to our RDF dataset) from Coffman's IMDb Benchmark (Coffman & Weaver 2010). We ran versions of *QUIRA* using a variety of ranking measures. Table 25 presents an overview of the results, with the Mean Average Precision (MAP), the total elapsed time and the number of iterations needed to compute the measures.

The measures in Table 25 include InfoRank, as defined in Section 3.1, a version of PageRank using the graph as undirected, the HITS Authorities, which

prioritizes nodes with high in-degree, and HITS Hubs, which prioritizes nodes with high out-degree. We also include the Degree-decoupled PageRank (Kim et al. 2016) with a penalization parameter of 0.5. Note that we did not compared InfoRank with any approach that uses manually weighted links due to its subjectivity, neither with approaches that learn weights from user feedback since we face the cold start problem. Moreover, we eliminated measures that are not computed efficiently in large data, such as, closeness centrality.

Furthermore, Table 26 and Table 27 show the complete results for InfoRank and PageRank. The *Create* column shows the time in seconds to transform the keyword query into a SPARQL query, column *Execute* shows the time to execute the query in Oracle 12c, and column *Build* shows the time to build the answers and present them to the user. We also included the individual Mean Average Precision (MAP) score for each query.

Table 25: IMDb results.

	Time(min)	Iterations	MAP
InfoRank	28	24	0.82
PageRank	27	30	0.76
HITS Authorities	25	12	0.73
HITS Hubs	25	12	0.30
Degree-decoupled PageRank p = 0.5	38	37	0.54

Analyzing the results, we noted that PageRank and HITS Authorities fail when choosing class Character, instead of class Work, in queries where a Steiner tree needs to be computed. As an example, consider the keyword query K={*harrison*, *ford*, *george*, *lucas*} from Coffman's Benchmark, whose expected results are movies directed by *George Lucas* and starred by *Harrison Ford*. Figure 19 and Figure 20 show the InfoRank and PageRank results, respectively. Note that, in both results, the algorithm correctly identifies Harrison Ford and George Lucas as instances of class Person. However, given that there two possible paths between two people (see Figure 17), PageRank chooses class Character as a preferable path since it has the highest degree, and InfoRank correctly chooses class Work since it is more informative.

Table 26: InfoRank results for IMDb.

Keyword Query	Create	Execute	Build	Total	AP
1. denzel washington	1.47	1.17	3.97	6.61	1.00
2. clint eastwood	1.74	60.32	0.94	63.00	1.00
3. john wayne	1.16	60.88	0.73	62.77	1.00
4. will smith	0.98	61.23	0.63	62.85	1.00
5. harrison ford	0.91	60.96	1.19	63.06	1.00
6. julia roberts	1.23	60.06	0.69	61.98	1.00
7. tom hanks	1.26	19.45	0.53	21.24	1.00
8. johnny depp	0.85	21.25	0.93	23.02	1.00
9. angelina jolie	1.24	60.84	0.62	62.71	1.00
10. morgan freeman	1.25	59.42	0.61	61.28	1.00
11. gone with the wind	1.21	60.78	0.55	62.54	1.00
12. star wars	1.67	63.08	0.63	65.38	1.00
13. casablanca	1.37	62.91	0.55	64.83	1.00
14. lord of the rings	1.30	61.55	0.83	63.69	1.00
15. the sound of music	6.74	66.78	0.65	74.17	1.00
16. wizard of oz	1.26	61.20	0.64	63.09	1.00
17. the notebook	1.25	20.79	0.63	22.67	1.00
18. forrest gump	0.76	20.94	0.90	22.59	1.00
19. the princess bride	1.38	59.41	1.13	61.91	1.00
20. the godfather	1.08	61.39	0.63	63.11	1.00
21. atticus finch movie	1.90	22.18	0.89	24.97	1.00
22. indiana jones movie	1.99	61.86	0.53	64.38	0.00
23. james bond movie	1.95	66.57	0.52	69.04	0.00
24. rick blaine movie	1.72	22.37	0.69	24.78	0.00
25. will kane movie	1.93	60.96	0.60	63.48	0.00
26. dr. hannibal lecter movie	1.87	22.31	0.65	24.83	1.00
27. norman bates movie	2.21	64.59	0.51	67.31	1.00
28. darth vader movie	1.88	24.39	0.72	26.99	1.00
29. the wicked witch of the west movie	2.00	65.64	0.63	68.27	1.00
30. nurse ratched movie	2.76	22.13	0.62	25.51	1.00
31. frankly my dear i don't give a damn	7.94	11.93	0.78	20.66	1.00
32. i'm gonna make him an offer he can't …	7.866	20.283	0.87	29.02	1.00
33. you don't understand i coulda had class …	6.763	61.03	0.55	68.35	1.00
34. toto, i've a feeling we're not in kansas…	6.262	21.99	0.549	28.805	1.00
35. here's looking at you kid	2.278	62.669	0.631	65.578	0.00
36. hamill skywalker	3.892	103.367	0.557	107.816	1.00
37. tom hanks 2004	17.157	120.755	0.534	138.446	0.00
38. henry fonda yours mine ours character	14.89	30.877	0.527	46.294	1.00
39. russell crowe gladiator character	9.508	101.688	0.744	111.94	1.00
40. brent spiner star trek character	11.512	95.765	0.627	107.904	1.00
41. audrey hepburn 1951	4.946	90.74	0.726	96.412	0.00
42. jacques clouseau actor	2.321	63.092	0.565	65.978	1.00
43. jack ryan actor	1.909	60.543	3.034	65.486	0.00
44. rocky stallone	10.459	108.386	3.127	121.972	1.00
45. terminator actor	1.861	66.645	0.823	69.329	1.00
46. harrison ford george lucas	24.257	110.807	0.629	135.693	1.00
47. sean connery fleming	11.337	105.784	0.73	117.851	1.00
48. keanu reeves wachowski	4.26	93.424	0.698	98.382	1.00
49. dean jones herbie	16.758	84.688	0.524	101.97	0.00
50. indiana jones last crusade lost ark person	9.462	32.159	0.525	42.146	1.00
MIN	0.76	1.17	0.51	6.61	0.00
AVG	4.52	57.68	0.84	63.04	**0.82**
MAX	24.26	120.76	3.97	138.45	1.00

Table 27: PageRank results for IMDb.

Keyword Query	Create	Execute	Build	Total	AP
1. denzel washington	1.25	24.19	2.28	27.71	1.00
2. clint eastwood	1.24	59.56	0.67	61.47	1.00
3. john wayne	0.97	60.39	0.58	61.95	1.00
4. will smith	1.29	61.65	0.61	63.54	1.00
5. harrison ford	0.99	60.63	0.59	62.21	1.00
6. julia roberts	1.36	59.83	0.56	61.75	1.00
7. tom hanks	1.73	19.82	1.05	22.59	1.00
8. johnny depp	1.03	23.33	0.70	25.06	1.00
9. angelina jolie	1.20	68.52	0.61	70.33	1.00
10. morgan freeman	1.17	81.29	0.64	83.10	1.00
11. gone with the wind	1.15	61.42	0.52	63.09	1.00
12. star wars	1.96	62.26	0.93	65.16	0.78
13. casablanca	1.70	61.45	0.52	63.67	1.00
14. lord of the rings	0.89	62.10	0.73	63.71	1.00
15. the sound of music	6.08	72.49	0.94	79.51	1.00
16. wizard of oz	1.12	61.49	0.63	63.24	1.00
17. the notebook	0.79	19.70	0.64	21.13	1.00
18. forrest gump	1.89	19.41	1.21	22.51	1.00
19. the princess bride	1.22	61.15	0.52	62.89	1.00
20. the godfather	1.21	61.40	0.63	63.24	1.00
21. atticus finch movie	2.14	21.99	0.58	24.71	1.00
22. indiana jones movie	2.46	66.33	1.07	69.86	1.00
23. james bond movie	1.98	61.42	0.49	63.89	0.00
24. rick blaine movie	2.27	20.93	0.69	23.89	1.00
25. will kane movie	1.86	65.75	0.50	68.11	1.00
26. dr. hannibal lecter movie	2.16	22.39	0.66	25.22	1.00
27. norman bates movie	2.14	64.42	0.62	67.18	1.00
28. darth vader movie	2.00	24.88	0.58	27.46	1.00
29. the wicked witch of the west movie	2.02	64.62	0.73	67.36	1.00
30. nurse ratched movie	2.14	22.40	0.63	25.17	1.00
31. frankly my dear i don't give a damn	8.65	20.47	0.60	29.72	1.00
32. i'm gonna make him an offer he can't …	8.76	20.74	0.75	30.25	1.00
33. you don't understand i coulda had class …	-	-	-	timeout	0.00
34. toto, i've a feeling we're not in kansas…	7.01	27.15	4.34	38.50	1.00
35. here's looking at you kid	2.27	61.11	0.60	63.99	0.00
36. hamill skywalker	4.15	106.44	0.62	111.20	1.00
37. tom hanks 2004	20.94	219.02	0.83	240.78	0.00
38. henry fonda yours mine ours character	16.67	31.32	0.63	48.62	1.00
39. russell crowe gladiator character	9.55	206.95	0.75	217.25	0.00
40. brent spiner star trek character	7.30	99.08	1.02	107.39	1.00
41. audrey hepburn 1951	3.88	85.69	0.84	90.41	0.00
42. jacques clouseau actor	2.56	65.54	0.78	68.88	1.00
43. jack ryan actor	1.42	59.84	0.51	61.76	0.00
44. rocky stallone	3.04	87.32	0.76	91.11	1.00
45. terminator actor	1.84	69.96	0.84	72.63	0.00
46. harrison ford george lucas	14.66	110.09	0.86	125.61	0.00
47. sean connery fleming	6.55	84.53	0.69	91.77	0.00
48. keanu reeves wachowski	3.27	95.73	0.65	99.65	0.00
49. dean jones herbie	8.47	86.76	0.60	95.83	0.00
50. indiana jones last crusade lost ark person	8.09	32.42	0.54	41.05	1.00
MIN	0.79	19.41	0.49	21.13	0.00
AVG	3.89	62.60	0.80	67.29	**0.76**
MAX	20.94	219.02	4.34	240.78	1.00

PageRank and HITS Authorities also fail in the ranking step for some keyword queries due to the high dependency on the degree. For example, consider the query *K*={*terminator*, *actor*} whose expected results are the *Terminator* movies stared by *Arnold Schwarzenegger*. Figure 21 and Figure 22 show the InfoRank and PageRank results, respectively. PageRank ranks first the voice actor *Jim Cummings* because his node has a high degree, since voice actors are usually casted several times, whereas InfoRank correctly returns in the top results the movies *The Terminator*, *Terminator 2: Judgment Day* and *Terminator 3: Rise of the Machines* starred by *Schwarzenegger*.

The HITS Hubs fails in all queries that refer to a person (e.g. *K*={*denzel*, *washington*}) since instances of class Person do not have outgoing edges. Furthermore, the Degree Decoupled PageRank (Kim et al. 2016), with a penalization parameter of 0.5, fails because it penalizes instances with high degree, whereas many important instances (e.g. *Star Wars*) have a high degree.

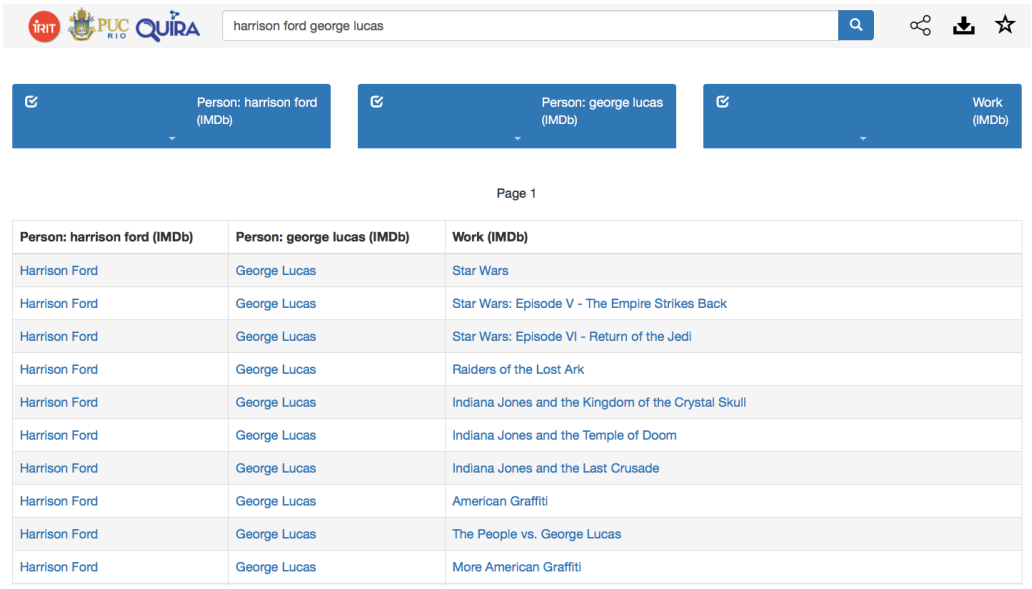


Figure 19: InfoRank result for *K*={*harrison*, *ford*, *george*, *lucas*}.

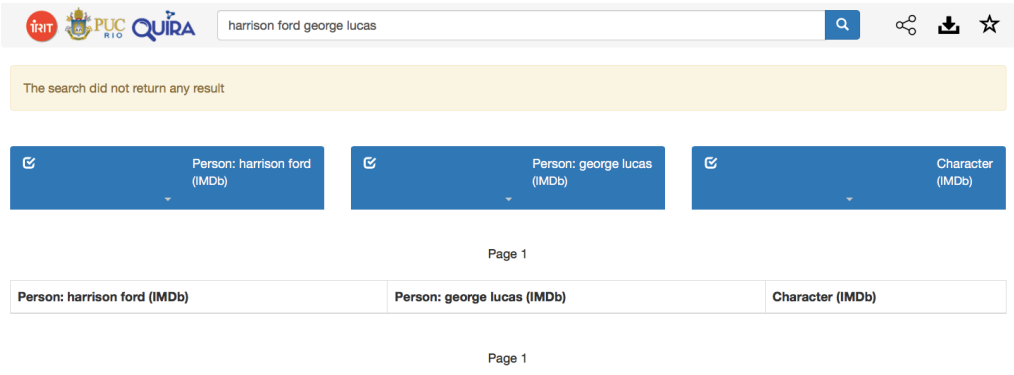
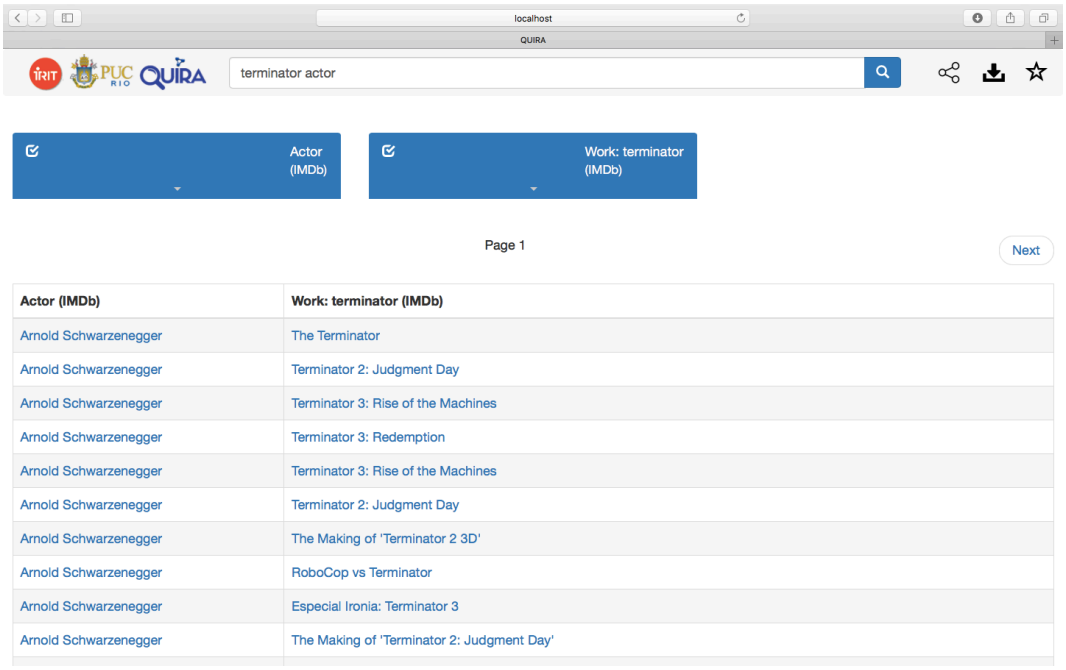
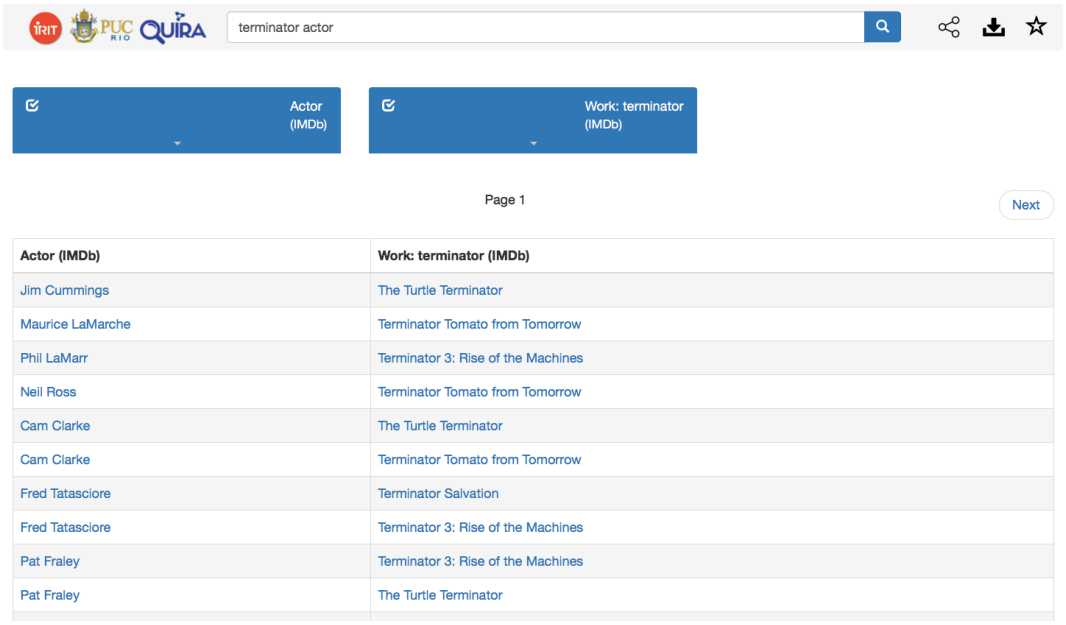


Figure 20: PageRank result for *K*={*harrison*, *ford*, *george*, *lucas*}.



Actor (IMDb)	Work: terminator (IMDb)
Arnold Schwarzenegger	The Terminator
Arnold Schwarzenegger	Terminator 2: Judgment Day
Arnold Schwarzenegger	Terminator 3: Rise of the Machines
Arnold Schwarzenegger	Terminator 3: Redemption
Arnold Schwarzenegger	Terminator 3: Rise of the Machines
Arnold Schwarzenegger	Terminator 2: Judgment Day
Arnold Schwarzenegger	The Making of 'Terminator 2 3D'
Arnold Schwarzenegger	RoboCop vs Terminator
Arnold Schwarzenegger	Especial Ironia: Terminator 3
Arnold Schwarzenegger	The Making of 'Terminator 2: Judgment Day'

Figure 21: InfoRank result for *K* = {*terminator*, *actor*}.



Actor (IMDb)	Work: terminator (IMDb)
Jim Cummings	The Turtle Terminator
Maurice LaMarche	Terminator Tomato from Tomorrow
Phil LaMarr	Terminator 3: Rise of the Machines
Neil Ross	Terminator Tomato from Tomorrow
Cam Clarke	The Turtle Terminator
Cam Clarke	Terminator Tomato from Tomorrow
Fred Tatasciore	Terminator Salvation
Fred Tatasciore	Terminator 3: Rise of the Machines
Pat Fraley	Terminator 3: Rise of the Machines
Pat Fraley	The Turtle Terminator

Figure 22: PageRank result for *K* = {*terminator*, *actor*}.

To summarize, InfoRank achieves the best MAP result in Coffman's IMDb Benchmark queries, since it successfully finds a balance between degree and informativeness. Furthermore, Table 25 indicates that this type of centrality measure, based on the Power Iteration method, can be computed in feasible time.

6.3.2. MusicBrainz

Continuing our experiments, we used 25 queries from QALD-2[17] (also adapted to our schema) to evaluate the impact of InfoRank in a keyword search system over MusicBrainz.

Table 28 and Table 29 show the results of the evaluation using InfoRank and PageRank, respectively. The results only differed in query K = {*Hardcore, Kids, duration*}, which should return the duration of track *Hardcore Kids*. Note that InfoRank (Figure 23) returns the expected result. On the other hand, PageRank (Figure 24) matches *Hardcore Kids* with an album, instead of a track, since it gives a priority to music albums that have a higher number of tracks, given that more tracks imply more links. However, we argue that the number of tracks is not necessarily related to the importance of an album.

Table 28: InfoRank results for MusicBrainz.

Keyword Query	Create	Execute	Build	Total	AP
1. Slayer track	1.58	33.61	3.95	39.14	1.00
2. David Bowie group	1.14	26.54	0.79	28.47	1.00
3. group Dover start year	1.58	22.00	0.62	24.20	1.00
4. Michael Jackson album	0.87	67.29	0.77	68.93	1.00
5. Star Wars soundtrack	1.72	76.67	0.78	79.16	1.00
6. solo artist birth date 1962-05-30	1.98	64.06	0.69	66.73	-
7. The Cure solo artist	1.199	21.792	0.672	23.663	0.80
8. Kraftwerk album	0.72	25.24	0.59	26.56	1.00
9. group Nirvana	1.18	12.78	0.68	14.64	1.00
10. Sex Pistols group end year	1.18	20.91	0.80	22.89	1.00
11. Quee MacArthur group Queen	4.01	92.24	0.90	97.15	1.00
12. Tom Waits birth date	1.30	20.81	0.61	22.72	1.00
13. solo artist birth date 1960-12-29	3.36	124.82	0.65	128.82	-
14. group end year 2010	7.33	75.29	0.67	83.30	1.00
15. BBC Symphony Orchestra album	0.70	92.29	0.67	93.66	0.00
16. Michael Stipe group	0.67	21.98	0.56	23.22	1.00
17. Amy Macdonald album	1.00	66.02	0.68	67.70	1.00
18. Michael Jackson live album	2.25	77.70	0.67	80.63	0.00
19. In Utero producer	1.30	20.99	0.66	22.95	-
20. Hardcore Kids duration	2.23	61.53	0.52	64.27	1.00
21. Kurt Cobain Nirvana	0.73	20.21	1.04	21.98	0.00
22. Aretha Franklin track	0.96	26.67	0.70	28.33	1.00
23. Millencolin group start year	1.56	21.61	0.63	23.81	1.00
24. group Trio solo artist	9.79	30.68	0.66	41.13	0.00
25. Ramones solo artist	0.73	22.01	0.71	23.45	-
MIN	0.67	12.78	0.52	14.64	0.00
AVG	2.04	45.83	0.83	48.70	**0.80**
MAX	9.79	124.82	3.95	128.82	1.00

[17] https://github.com/ag-sc/QALD

Table 29: PageRank results for MusicBrainz.

Keyword Query	Create	Execute	Build	Total	AP
1. Slayer track	1.43	35.765	2.819	40.014	1.00
2. David Bowie group	0.924	28.292	0.813	30.029	1.00
3. group Dover start year	1.54	22.915	0.82	25.275	1.00
4. Michael Jackson album	1.429	71.058	0.779	73.266	1.00
5. Star Wars soundtrack	2.11	70.522	0.712	73.344	1.00
6. solo artist birth date 1962-05-30	1.678	60.002	0.665	62.345	-
7. The Cure solo artist	1.569	21.904	1.023	24.496	0.80
8. Kraftwerk album	0.837	24.483	0.753	26.073	1.00
9. group Nirvana	7.61	19.466	0.497	27.573	1.00
10. Sex Pistols group end year	1.653	26.627	0.854	29.134	1.00
11. Quee MacArthur group Queen	6.938	186.027	0.592	193.557	1.00
12. Tom Waits birth date	0.901	23.162	0.817	24.88	1.00
13. solo artist birth date 1960-12-29	7.579	141.798	0.738	150.115	-
14. group end year 2010	13.023	129.654	0.74	143.417	1.00
15. BBC Symphony Orchestra album	0.638	83.114	0.684	84.436	0.00
16. Michael Stipe group	0.677	21.955	0.566	23.198	1.00
17. Amy Macdonald album	0.617	63.783	0.657	65.057	1.00
18. Michael Jackson live album	1.524	69.067	0.611	71.202	0.00
19. In Utero producer	1.342	23.84	0.535	25.717	-
20. Hardcore Kids duration	0.933	65.52	0.717	67.17	0.00
21. Kurt Cobain Nirvana	0.652	20.69	0.57	21.912	0.00
22. Aretha Franklin track	0.629	24.975	0.671	26.275	1.00
23. group Millencolin start year	1.556	21.188	0.864	23.608	1.00
24. group Trio solo artist	9.499	35.955	0.729	46.183	0.00
25. Ramones solo artist	0.891	20.344	0.773	22.008	-
MIN	0.62	19.47	0.50	21.91	0.00
AVG	2.73	52.48	0.80	56.01	**0.75**
MAX	13.02	186.03	2.82	193.56	1.00

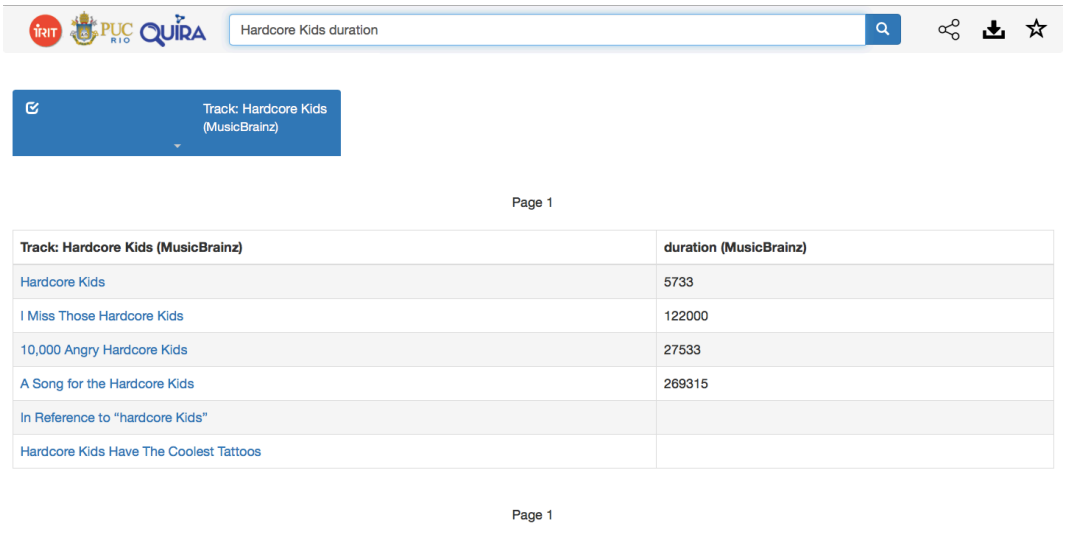
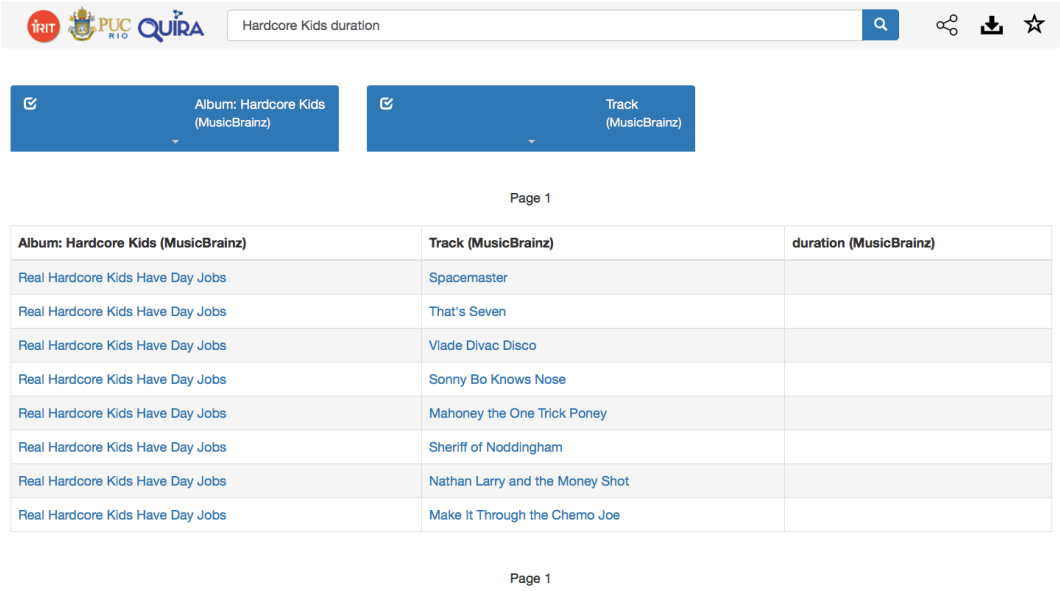


Figure 23: InfoRank result for *K* = {*Hardcore, Kids, duration*}.



Album: Hardcore Kids (MusicBrainz)	Track (MusicBrainz)	duration (MusicBrainz)
Real Hardcore Kids Have Day Jobs	Spacemaster	
Real Hardcore Kids Have Day Jobs	That's Seven	
Real Hardcore Kids Have Day Jobs	Vlade Divac Disco	
Real Hardcore Kids Have Day Jobs	Sonny Bo Knows Nose	
Real Hardcore Kids Have Day Jobs	Mahoney the One Trick Poney	
Real Hardcore Kids Have Day Jobs	Sheriff of Noddingham	
Real Hardcore Kids Have Day Jobs	Nathan Larry and the Money Shot	
Real Hardcore Kids Have Day Jobs	Make It Through the Chemo Joe	

Figure 24: PageRank result for *K* = {*Hardcore, Kids, duration*}.

6.4. Chapter Conclusion

In this chapter we presented an evaluation using the IMDb and MusicBrainz datasets. In a preliminary step, we compared rankings induced by InfoRank and PageRank regarding classes, properties and instances. While PageRank results show highly connected nodes in the first positions, InfoRank provides what we argue to be more important nodes considering the domain. Hence, the preliminary experiments suggested that InfoRank could achieve a good ranking mechanism when used in a keyword search system.

Indeed, in the second step of the evaluation, we ran two popular benchmarks over IMDb and MusicBrainz using different versions of QUIRA to compare the results provided by different importance measures. The two best results were InfoRank and PageRank. However, we showed that InfoRank outperformed PageRank in both datasets.

7
Conclusions and Future Work

7.1. Conclusions

In the last years, keyword search over RDF graphs (briefly *RDF-KwS*) became a relevant research topic with the goal of hiding from users the unfriendly SPARQL queries. Many of the proposed solutions adapted popular Information Retrieval techniques to the *RDF-KwS* world, including ranking mechanisms (e.g. PageRank) that consider the importance of the retrieved documents. However, PageRank, and other typical importance measures, highly depend on the degree of nodes, whereas the notion of importance of a node is not necessarily related to its degree in RDF graphs.

Therefore, the first contribution of this thesis, presented in Chapter 3, is a novel family of importance measures, called InfoRank, designed for degree-independent RDF Graphs. The proposed importance measures are combinations of three intuitions: (I) *"important things have lots of information about them"*; (II) *"important things are surrounded by other important things"*; (III) *"few important relations (e.g. friends) are better than many unimportant relations (e.g. acquaintances)"*.

The second contribution, presented in Chapter 4, is a strategy that translates keyword queries into SPARQL queries and that incorporates the InfoRank measure. The strategy solves the three tasks of the *RDF-KwS* problem: (1) finding pieces of information in the RDF graph; (2) assembling the retrieved pieces of information to compose complete answers; (3) ranking the complete answers. Furthermore, Chapter 5 presents the architecture and interface of QUIRA (*QUerying with InfoRAnk*), a tool that implements the strategy.

The third contribution, presented in Chapter 6, consists of two enriched RDF datasets, IMDb and MusicBrainz[18], along with keyword search benchmarks adapted to the RDF enviroment.

We used these datasets to evaluate InfoRank and QUIRA. We first presented preliminary experiments to assess the potential of InfoRank as an importance measure. Thus, for both datasets, we compared the rankings generated by InfoRank and PageRank regarding classes, properties and instances. The top instances of the PageRank rankings include highly connected nodes, such as countries (e.g. *United States* and *France*), either in IMDb and MusicBrainz. However, the InfoRank rankings return mostly popular instances according to the domain, for example, *Star Wars*, *Morgan Freeman* and *Titanic*, in IMDb, and *Elvis Presley*, *Mozart* and *Beethoven*, in MusicBrainz. Hence, these experiments indicate that InfoRank would provide better ranking results than PageRank when used in a keyword search system.

In the second part of the evaluation, we tested our tool with two popular keyword search benchmarks for IMDb and MusicBrainz, adapted to the RDF schema. We ran the benchmarks with versions of QUIRA using different importance measures, such as PageRank, Degree-decoupled PageRank, HITS, and InfoRank itself. Indeed, from the experiments, we were able to conclude that InfoRank improves the quality of results, when compared to other ranking strategies. For instance, in IMDb, PageRank fails when it gives priority to class Character, instead of class Work, in queries where a Steiner tree needs to be computed. It also fails when it ranks first supporting actors that are casted several times (i.e. nodes with high degree), instead of lead actors, such as *Arnold Schwarzenegger*. In MusicBrainz, a similar scenario happens when PageRank prioritizes music albums that have a higher number of tracks (again, nodes with high degree), instead of other popular albums or tracks.

Hence, to summarize the results, InfoRank achieved a Mean Average Precision (MAP) of 0.82 and PageRank a MAP of 0.76 in IMDb. While in MusicBrainz, InfoRank achieved a MAP of 0.80 and PageRank a MAP of 0.75.

[18] https://musicbrainz.org

7.2. Future Work

We may suggest a range of possibilities that would improve our solution.

The first one is actually a research field called Entity Linking (Moro et al. 2014), which refers to the task of linking entity references in a text to a knowledge base. This could be used to discover entities in literals of an RDF dataset. For instance, consider the trivia of *The Sound of Music* that says, "…the actual dance by Julie Andrews and Christopher Plummer was filmed on a replica located in L.A.". Recall from Section 4.2.1 that this caused an ambiguity with the label of actress *Julie Andrews*. Hence, we could perform a pre-processing step to annotate free-text literals of an RDF dataset using the labels of the same dataset. For instance, the mentioned trivia would become "…the actual dance by **Julie Andrews** and **Christopher Plummer** was filmed on a replica located in L.A.". Then, we could exclude such links when indexing literals for the keyword search process, which would eliminate the ambiguity between the trivia and label of the actress (or actor). Note that this step can also benefit from InfoRank to choose the most important entities. Furthermore, this would represent an advantage to the users, since the interface would have more links for them to follow.

The second possible path to follow is another research field, called Entity Summarization (Cheng et al. 2011; Thalhammer et al. 2012), which aims at providing meaningful descriptions of entities. This field is an extension of the Ontology Summarization field (Zhang et al. 2007; Peroni et al. 2008). We actually have a preliminary implementation of this feature in the sense that the user is able to click on an URI instance and see its data and its relations with other instances, as described in Chapter 5. However, we can improve the implementation with techniques from the Entity Summarization field to show more meaningful information. Furthermore, we can also benefit from InfoRank to show the most important instances first.

Other possible future work is to test QUIRA with datasets with larger schemas. Recall from Section 4.2.2 that we compute an approximated Steiner tree over the schema graph. However, in a dataset with thousands of classes the computation of the Steiner tree could be very inefficient. Furthermore, we could

also experiment different ranking functions in the final SPARQL query. Recall from Section 4.2.3 that we simply sum the InfoRank scores of all instances involved in the query. However, we could use some strategy that gives more weight to instances that are more important to the query. Finally, another option could be to adopt domain knowledge, such as user rating of IMDb, in the ranking function.

Other more obvious possibilities to improve our keyword search system are the well-known fields of Natural Language Processing/Question Answering (Unger et al. 2012; Freitas et al. 2013), and Machine Learning algorithms that take advantage of user feedback (Komamizu et al. 2017).

8
Bibliography

AGARWAL, Alekh et al. Learning to rank networked entities. In: **Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. ACM Press, 2006, p. 14.

AGRAWAL, Sanjay et al. DBXplorer: a system for keyword-based search over relational databases. In: **Proceedings of the 18th IEEE International Conference on Data Engineering (ICDE)**. IEEE Comput. Soc, 2002, p. 5-16.

BALMIN, Andrey et al. Objectrank: Authority-based Keyword Search in Databases. In: **Proceedings of the 13th International Conference on Very Large Data Bases (VLDB)**. VLDB Endowment, 2004, p. 564-575.

BAST, Hannah et al. Semantic Search on Text and Knowledge Bases. **Foundations and Trends® in Information Retrieval**, v. 10, n. 2-3, p. 119-271, 2016.

BERNERS-LEE, Tim. Linked Data - Design Issues. 2006. Available at: <http://www.w3.org/DesignIssues/LinkedData.html>. Accessed January 2, 2019.

BHALOTIA, Gaurav, et al. Keyword searching and browsing in databases using BANKS. In: **Proceedings of the 18th IEEE International Conference on Data Engineering (ICDE)**. IEEE Comput. Soc, 2002, p. 431-440.

BRIN, Sergey; & PAGE, Lawrence. The anatomy of a large-scale hypertextual Web search engine. **Computer Networks and ISDN Systems**, v. 30, n. 1-7, p. 107-117, 1998.

CHENG, Gong et al. Relin: relatedness and informativeness-based centrality for entity summarization. In: **International Semantic Web Conference**. Springer, Berlin, Heidelberg, 2011. p. 114-129.

CHIRITA, Paul-Alexandru et al. Beagle++ : Semantically Enhanced Searching and Ranking on the Desktop. In: **Proceedings of the 3rd European Semantic Web Conference (ESWC)**. Springer, Berlin, Heidelberg, 2006. p. 348-362.

COFFMAN, Joel; & WEAVER, Alfred C. A Framework for Evaluating Database Keyword Search Strategies. In: **Proceedings of the 19th ACM International Conference on Information and Knowledge Managemen (CIKM)**. ACM, 2010, p. 729–38.

CORMEN, Thomas et al. **Introduction to Algorithms**. MIT Press, 2009.

DE OLIVEIRA, Pericles et al. Ranking Candidate Networks of Relations to Improve Keyword Search over Relational Databases. In: **Proceedings of the 31th IEEE International Conference on Data Engineering (ICDE)**. IEEE, 2015, p. 399–410.

DING, Li et al. Swoogle: A Search and Metadata Engine for the Semantic Web. In: **Proceedings of the 13th ACM International Conference on Information and Knowledge Management (CIKM)**. ACM, 2004, p. 652.

ELBASSUONI, Shady; & BLANCO, Roi. Keyword search over RDF graphs. In: **Proceedings of the 20th ACM international conference on Information and knowledge management**. ACM, 2011. p. 237-242.

FRANZ, Thomas et al. TripleRank: Ranking Semantic Web Data by Tensor Decomposition. In: **Proceedings of the 8th International Semantic Web Conference (ISWC)**. Springer, Berlin, Heidelberg, 2009, p. 213–28.

FREITAS, André et al. Querying linked data graphs using semantic relatedness: A vocabulary independent approach. **Data & Knowledge Engineering**, v. 88, p. 126-141, 2013.

GARCÍA, Grettel et al. RDF Keyword-Based Query Technology Meets a Real-World Dataset. In: **Proceedings of the 20th International Conference on Extending Database Technology (EDBT)**. EDBT, 2017, p. 656–67.

GRAVES, Alvaro et al. A Method to Rank Nodes in an RDF Graph. In: **Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC)**. CEUR-WS.org, 2008, p. 84–85.

HARTH, Andreas et al. Using Naming Authority to Rank Data and Ontologies for Web Search. In: **Proceedings of the 8th International Semantic Web Conference (ISWC)**. Springer, Berlin, Heidelberg, 2009, p. 277–92.

HE, Hao et al. BLINKS: Ranked Keyword Searches on Graphs. In: **Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data**. ACM, 2007, p. 305.

HEATH, Tom; & BIZER, Christian. Linked Data: Evolving the Web into a Global Data Space. **Synthesis Lectures on the Semantic Web: Theory and Technology**, v. 1, n. 1, p. 1–136, 2011.

HIEMSTRA, Djoerd. Information Retrieval Models. In: **Information Retrieval: Searching in the 21st Century**, p. 2–19, 2009.

HOGAN, Aidan et al. ReConRank: A Scalable Ranking Method for Semantic Web Data with Context. In: **Proceedings of the 2nd International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS)**. 2006.

HRISTIDIS, Vagelis; & PAPAKONSTANTINOU, Yannis. Discover: Keyword Search in Relational Databases. In: **Proceedings of the 28th International Conference on Very Large Databases (VLDB)**. Elsevier, 2002, p. 670–81.

IZQUIERDO, Yenier et al. QUIOW: A Keyword-Based Query Processing Tool for RDF Datasets and Relational Databases. In: **Proceedings of the 29th International Conference on Database and Expert Systems Applications (DEXA)**. Springer, Cham, 2018. p. 259-269.

KASNECI, Gjergji et al. NAGA: Searching and Ranking Knowledge. In: **Proceedings of the 24th IEEE International Conference on Data Engineering (ICDE)**. IEEE, 2008, p. 953–62.

KIM, Jung Hyun et al. PageRank Revisited: On the Relationship between Node Degrees and Node Significances in Different Applications? In: **GraphQ: 5th International Workshop on Querying Graph Structured Data (Satellite Event at EDBT/ICDT)**. CEUR-WS, 2016, p. 1–8.

KLEINBERG, Jon M. Authoritative Sources in a Hyperlinked Environment. **Journal of the ACM**, v. 46, n. 5, p. 604–32, 1999.

KOMAMIZU, Takahiro et al. FORK: Feedback-Aware ObjectRank-Based Keyword Search over Linked Data. In: **Asia Information Retrieval Symposium (AIRS)**. Springer, Cham, 2017, p. 58–70.

LE, Wangchao et al. Scalable keyword search on large RDF data. **IEEE Transactions on knowledge and data engineering**, v. 26, n. 11, p. 2774-2788, 2014.

LI, Rong-Hua et al. Efficient and progressive group steiner tree search. In: **Proceedings of the 2016 International Conference on Management of Data**. ACM, 2016. p. 91-106.

MARX, Edgard et al. DBtrends: Exploring Query Logs for Ranking RDF Data. In: **Proceedings of the 12th International Conference on Semantic Systems**. ACM, 2016a, p. 9-16.

MARX, Edgard et al. DBtrends: Publishing and Benchmarking RDF Ranking Functions. In: **2nd International Workshop on Summarizing and Presenting Entities and Ontologies, Co-located with the 13th Extended Semantic Web Conference**. SumPre@ ESWC, 2016b.

MIRIZZI, Roberto et al. **Ranking the Linked Data: The Case of DBpedia**. In: **Proceedings of the 10th International Conference on Web Engineering (ICWE)**. Springer, Berlin, Heidelberg, 2010, p. 337–54.

MORO, Andrea; RAGANATO, Alessandro; NAVIGLI, Roberto. Entity linking meets word sense disambiguation: a unified approach. **Transactions of the Association for Computational Linguistics**, v. 2, p. 231-244, 2014.

NGOMO, Ngonga et al. Holistic and Scalable ranking of RDF data. In: **Big Data (Big Data), 2017 IEEE International Conference on**. IEEE, 2017. p. 746-755.

NIE, Zaiqing et al. Object-Level Ranking. In: **Proceedings of the 14th International Conference on World Wide Web (WWW)**. ACM Press, 2005, p. 567.

OREN, Eyal et al. Sindice. com: a document-oriented lookup index for open linked data. **International Journal of Metadata, Semantics and Ontologies**, v. 3, n. 1, p. 37-52, 2008.

PARK, Hyunjung et al. A Link-Based Ranking Algorithm for Semantic Web Resources. **Journal of Database Management**, v. 22, n. 1, p. 1–25, 2011.

PERONI, Silvio et al. Identifying key concepts in an ontology, through the integration of cognitive principles with statistical and topological measures. In: **Asian Semantic Web Conference**. Springer, Berlin, Heidelberg, 2008. p. 242-256.

ROA-VALVERDE, Antonio J. et al. A Survey of Approaches for Ranking on the Web of Data. **Information Retrieval**, v. 17 n. 4, p. 295–325, 2014.

THALHAMMER, Andreas et al. Evaluating entity summarization using a game-based ground truth. In: **International Semantic Web Conference**. Springer, Berlin, Heidelberg, 2012. p. 350-361.

TRAN, Thanh et al. Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped RDF Data. In: **Proceedings of the 25th IEEE**

International Conference on Data Engineering (ICDE). IEEE, 2009, p. 405–16.

TURPIN, Andrew & SCHOLER, Falk. User Performance versus Precision Measures for Simple Search Tasks. In: **Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval**. ACM Press, 2006, p. 11.

UNGER, Christina et al. Template-based question answering over RDF data. In: **Proceedings of the 21st international conference on World Wide Web**. ACM, 2012. p. 639-648.

WEI, Wang et al. Rational Research Model for Ranking Semantic Entities. **Information Sciences**, v. 181, n.13, p. 2823–40, 2011.

YU, Jeffrey et al. Chang. 2010. **Keyword Search in Databases**. Morgan & Claypool.

YUMUSAK, Semih et al. A Short Survey of Linked Data Ranking. In: **Proceedings of the 2014 ACM Southeast Regional Conference**. ACM Press, 2014, p. 1–4.

ZENZ, Gideon et al. From Keywords to Semantic Queries—Incremental Query Construction on the Semantic Web. **Journal of Web Semantics: Science, Services and Agents on the World Wide Web**, v. 7, n. 3, p. 166–76, 2009.

ZHANG, Xiang et al. Ontology summarization based on rdf sentence graph. In: **Proceedings of the 16th international conference on World Wide Web**. ACM, 2007. p. 707-716.

ZHENG, Weiguo et al. Semantic SPARQL Similarity Search over RDF Knowledge Graphs. **Proceedings of the VLDB Endowment**, v. 9, n. 11, p. 840–51, 2016.

ZHOU, Q et al. SPARK: Adapting Keyword Query to Semantic Search. In: **Proceedings of the 6th International Semantic Web Conference (ISWC)**. Springer, Berlin, Heidelberg, 2007, p. 694–707.