Pontifícia Universidade Católica
do Rio de Janeiro

Angelo Batista Neves Júnior

Automatic Generation of Benchmarks for Evaluating Keyword and Natural Language Interfaces to RDF Datasets

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
Co-advisor: Prof. Luiz André Portes Paes Leme

Rio de Janeiro
September 2022

Pontifícia Universidade Católica
do Rio de Janeiro

Angelo Batista Neves Júnior

Automatic Generation of Benchmarks for Evaluating Keyword and Natural Language Interfaces to RDF Datasets

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. Approved by the Examination Committee:

Prof. Marco Antonio Casanova
Advisor
Departamento de Departamento de Informática – PUC-Rio

Prof. Luiz André Portes Paes Leme
Co-advisor
UFF

Prof. Antonio Luz Furtado
Departamento de Informática – PUC-Rio

Profa. Melissa Lemos Cavaliére
Instituto Tecgraf – PUC-Rio

Profa. Vânia Maria Ponte Vidal
UFC

Prof. Geraldo Bonorino Xexéo
UFRJ

Rio de Janeiro, September 9th, 2022

Angelo Batista Neves Júnior

Angelo Batista Neves Júnior graduated in Computer Science at the Federal Institute of Education Science and Technology of Southeast Minas Gerais - Campus Rio Pomba. He holds a master's degree in Systems and Information Engineering from Universidade Federal Fluminense. His line of research focuses on the Semantic Web, in which he studies keyword recommendation and search techniques in semantic databases. He is currently a fellow at Instituto Tecgraf / PUC-Rio.

Bibliographic data

Neves Júnior, Angelo Batista

Automatic Generation of Benchmarks for Evaluating Keyword and Natural Language Interfaces to RDF Datasets / Angelo Batista Neves Júnior; advisor: Marco Antonio Casanova; co-advisor: Luiz André Portes Paes Leme. – 2022.

83 f: il. color. ; 30 cm

Tese (doutorado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Departamento de Informática, 2022.

Inclui bibliografia

1. Informática – Teses. 2. Benchmark. 3. Interface em Linguagem Natural. 4. datasets. I. Casanova, Marco A.. II. Portes Paes Leme, Luiz. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Departamento de Informática. IV. Título.

CDD: 004

To my family and my friends for their support and motivation.

Acknowledgments

First, thank God for giving me health during this research; without Him, I would not have gotten anywhere.

For my advisor, Prof. Marco Antonio Casanova. He helped with everything I needed during my doctorate. He is the most intelligent person and didactic that I know.

For my co-advisor, Prof. Luiz André Portes Paes Leme. He always helped during my master's degree and now my doctorate. He always was willing to take my questions. I always will be grateful for all the knowledge acquired by him.

Thanks my parents, Angelo Batista Neves and Vânia da Costa Pacheco Neves. They have always supported, encouraged, and done everything for me.

I also thank my sister Mariana Pacheco Neves, who also always supported and accompanied me during my doctorate.

To my best friends, Fillipe Flores, Rodrigo Abrão, Maria Clara, and Laís Brandão, for always supporting me during my doctorate.

My Tecgraf friends Melissa, Yenier, Grettel, Javier, and Bruno for the knowledge acquired.

To all the professors at PUC-Rio that I had the opportunity to be a student, for having I learn more and more.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. Thank you for the aids granted, without which this work does not could have been accomplished.

Last but not least, I would like to thank everyone who directly and indirectly helped me in this research.

Abstract

Neves Júnior, Angelo Batista; Casanova, Marco A. (Advisor); Portes Paes Leme, Luiz (Co-Advisor). **Automatic Generation of Benchmarks for Evaluating Keyword and Natural Language Interfaces to RDF Datasets**. Rio de Janeiro, 2022. 83p. Tese de Doutorado – Departamento de Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Text search systems provide users with a friendly alternative to access Resource Description Framework (RDF) datasets. The performance evaluation of such systems requires adequate benchmarks, consisting of RDF datasets, text queries, and respective expected answers. However, available benchmarks often have small sets of queries and incomplete sets of answers, mainly because they are manually constructed with the help of experts. The central contribution of this thesis is a method for building benchmarks automatically, with larger sets of queries and more complete answers. The proposed method works for both keyword and natural language queries and has two steps: query generation and answer generation. The query generation step selects a set of relevant entities, called inducers, and, for each one, heuristics guide the process of extracting related queries. The answer generation step takes the queries and computes solution generators (SG), subgraphs of the original dataset containing different answers to the queries. Heuristics also guide the construction of SGs, avoiding the waste of computational resources in generating irrelevant answers.

Keywords

Benchmark; Natural Language Interface; datasets.

Resumo

Neves Júnior, Angelo Batista; Casanova, Marco A.; Portes Paes Leme, Luiz. **Geração Automática de Benchmarks para Avaliar Interfaces Baseadas em Palavras-Chave e Linguagem Natural para Datasets RDF**. Rio de Janeiro, 2022. 83p. Tese de Doutorado – Departamento de Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Os sistemas de busca textual fornecem aos usuários uma alternativa amigável para acessar datasets RDF (Resource Description Framework). A avaliação de desempenho de tais sistemas requer benchmarks adequados, consistindo de datasets RDF, consultas e respectivas respostas esperadas. No entanto, os benchmarks disponíveis geralmente possuem poucas consultas e respostas incompletas, principalmente porque são construídos manualmente com a ajuda de especialistas. A contribuição central desta tese é um método para construir benchmarks automaticamente, com um maior número de consultas e com respostas mais completas. O método proposto aplica-se tanto a consultas baseadas em palavras-chave quanto em linguagem natural e possui duas partes: geração de consultas e geração de respostas. A geração de consultas seleciona um conjunto de entidades relevantes, chamadas de indutores, e, para cada uma, heurísticas orientam o processo de extração de consultas relacionadas. A geração de respostas recebe as consultas produzidas no passo anterior e computa geradores de solução (SG), subgrafos do dataset original contendo diferentes respostas às consultas. Heurísticas também orientam a construção dos SGs evitando o desperdiço de recursos computacionais na geração de respostas irrelevantes.

Palavras-chave

Benchmark; Interface em Linguagem Natural; datasets.

Table of contents

List of figures

List of tables

List of Abreviations

AE – *Autoencoding*

AR – *Autoregressive*

BERT – *Bidirectional Encoder Representations from Transformers*

SG – *Solution Generators*

IRI – *International Resource Identifier*

OWL – *Web Ontology Language*

POS – *part-of-speech*

RDF – *Resource Description Framework*

RDF-KwS – *RDF - Keyword Search systems*

RDF-NLIB – *Natural Language Interfaces to RDF Datasets*

RDF-TS – *RDF Textual Search*

RDFS – *Resource Description Framework Schema*

RFC – *Request for Comments*

When science offers an answer, that answer is universal. Humans do not go to war over it; they rally around it.

Dan Brown, *Origin*.

1
Introduction

Keyword search is a popular information discovery method because it allows naive users to retrieve information without knowing schema details or query languages. The user specifies a few terms, called *keywords* and it is up to the system to retrieve the documents, such as Web pages, that best match the keywords. Following this trend, keyword search systems designed for Resource Description Framework (RDF) datasets have emerged. The latest advances in this area go beyond using keywords as search terms and accept queries expressed as natural language sentences. One usually refers to the former category as RDF Keyword Search systems (RDF-KwS), while to the latter as Natural Language Interfaces to RDF Datasets (RDF-NLID). For simplicity, we refer to both categories as *RDF Textual Search (RDF-TS) systems*. Likewise, we use *text queries* to refer to both keyword and natural language queries.

RDF-TS systems have three main tasks: (1) retrieve nodes in the RDF graph that the query specify; (2) discover how they are interrelated to compose complete answers; and (3) rank these answers [2]. Hence, answers are not just sets of nodes but sets of nodes and paths between them, i.e., subgraphs of the dataset.

RDF-TS systems are evaluated using benchmarks with sets of information needs and their respective lists of expected answers, possibly ordered, for a given dataset. Despite the many existing benchmarks for structured data [3, 4, 5, 6], these benchmarks have at least four limitations when it comes to RDF-TS: (1) they are frequently built for relational data; (2) they are incomplete in the sense that they do not cover many reasonable answers; (3) they are not always publicly available; (4) they are small.

To remedy the first limitation, some authors [7] adapted benchmarks developed for relational databases. However, the adaptation requires the triplification of relational databases and benchmark data, leading to problems when comparing different systems using different triplifications.

As an example of the incompleteness of existing benchmarks, consider the keyword query "Mauritius India", which is Query 43 for the Mondial dataset in Coffman's benchmark [8]. The list of expected answers in the benchmark covers just the organizations that both countries participate in. However, other

answers that express geographical relationships between Mauritius Islands and India should have been included in the list of expected answers. Incompleteness in this sense is a serious problem, which is difficult to overcome in manually constructed benchmarks.

Benchmark size is also an issue. Many RDF-TS systems are based on machine learning techniques and require a large volume of examples for training.

This thesis addresses the problem of constructing benchmarks for evaluating RDF-TS systems in a holistic approach, i.e., from the definition of queries to the computation of expected answers. This is a challenging problem for three fundamental reasons: (1) the set of queries included in the benchmark should be large and contain a broad range of query patterns; (2) RDF-TS tools vary widely and compute different – albeit correct – answers for the same query; (3) computing the set of all expected answers for a given query leads to an explosive combinatorial problem.

The main contributions of this thesis are then as follows. First, it introduces a method that, given an RDF dataset, automatically defines keyword queries, as well as natural language queries, and their respective expected answers. Therefore, one can use the method to create benchmarks to evaluate both keyword-based and natural language-based systems. Second, the thesis outlines an implementation of the method. Third, the thesis describes five benchmarks constructed with the proposed method and based on three real datasets, DBpedia, IMDb, and Mondial, and two synthetic datasets, LUBM and BSBM. Finally, it compares the constructed benchmarks with keyword search benchmarks published in the literature.

The rest of this thesis is organized as follows. Chapter 2 covers related work. Chapter 3 contains the required definitions. Chapter 4 describes the proposed method for the automatic generation of benchmarks for evaluating keyword and natural language interfaces to RDF Datasets. Chapter 5 evaluates the proposed benchmark generation method. Finally, Chapter 6 contains the conclusions and suggestions for future work.

2
Related Work

This chapter describes related work on benchmarks, keyword search systems, and natural language interfaces to database systems.

2.1
Benchmarks

A crucial aspect of keyword search systems is their evaluation. In recent years, the research community concentrated on evaluating keyword search systems over relational databases [9] and entity retrieval [10]. Examples of relational benchmarks are Coffman's benchmark [3], which uses 50 queries for Mondial, IMDb, and Wikipedia samples (not real user queries extracted from a search engine log), and Oliveira's benchmark [11], which uses Mondial, IMDb, DBLP, and Northwind.

Unfortunately, benchmarks to assess RDF-KwS systems are scarce [4]. To remedy this situation, some authors [7] adapted relational benchmarks to RDF. However, this approach depends on the triplification of relational databases and does not easily induce complete sets of possible answers [12].

State-of-the-art RDF keyword search systems use different benchmarks, which are not always available. For example, Dosso and Silvello [4] described openly available benchmarks over three real datasets, LinkedMDB, IMDb, and a subset of DBpedia [10], and two synthetic databases, the Lehigh University Benchmark (LUBM) [13] and the Berlin SPARQL Benchmark (BSBM) [14]. For IMDb, they defined 50 keyword queries and their correct translations to SPARQL queries. For DBpedia, the authors considered 50 topics from the classes `QALD2_te` and `QALD2_tr` of the *Question Answering over Linked Data* (QALD) campaigns[1]. For the synthetic databases, they used 14 SPARQL queries for LUBM and 13 SPARQL queries for BSBM. For all original `SELECT` queries from these datasets, Dosso and Silvello mapped these queries to SPARQL `CONSTRUCT` queries and produced their equivalent keyword queries.

In particular, the Lehigh University Benchmark (LUBM) [13] is widely used to assess the performance of SPARQL engines and was later extended [15] to cover full-text search performance.

[1] `http://qald.aksw.org`

Some benchmarks are created by trying to optimize the number of queries generated. For example, Poess and Stephens [16] present QGEN, a flexible, high-level query generator optimized for decision support system evaluation. QGEN allows the generation of arbitrary query sets, which conforms to a selected statistical profile.

However, none of these benchmarks was specifically designed for RDF keyword search.

Other benchmarks in the State-of-the-art RDF can be listed. For example Zheng et al. [17] used DBpedia and Yago datasets. The queries used were derived from QALD-4. Han et al. [18] describe the benchmark over the DBpedia with QALD-6 and Freebase with Free917 datasets based on an open question answering benchmark that consists of natural language question and answers pairs over Freebase. Izquierdo et al. [12] used full versions of the Mondial and IMDb datasets and queries from Coffman's benchmark in their experiments.

Lin et al. [19] described the benchmark over datasets: LUBM, Wordnet, BSBM, Barton, and DBpedia Infobox. They used twelve keyword queries to assess their system. Wen et al. [20] used nine queries for YAGO, three queries for DBLP, and six queries for LUBM. Rihany et al. [21] described the benchmark over two datasets: AIFB and DBpedia. They used ten queries for each dataset and the sizes of the queries were between 2 and 8 keywords. Menendez et al. [2] used in your experiments to assess the tool QUIRA full versions of IMDb and MusicBrainz. They used 50 queries of Coffman's Benchmark for IMDb, and 25 queries from QALD-2 for MusicBrainz.

There are some benchmarks that have no public link or are not available for download. For example, Zhou et al. [22] described a benchmark and keyword queries over Mooney Natural Language Learning Data. Zenz et al. [23] used a benchmark containing an initial set of queries extracted from a query log of the AOL search engine. Then, the queries were pruned based on the visited URLs, obtaining 3,000 sample keyword queries for IMDb and Lyrics Web pages. This process yielded 100 queries for IMDb, and 75 queries for Lyrics, consisting of 2–5 keywords. Tran et al. [24] describe a benchmark with 30 queries for DBLP, and 9 for TAP[2]. Elbassuoni and Blanco [25] used datasets derived from the LibraryThing community and IMDb, and 15 queries for each dataset. Lastly, Le et al. [26] used the benchmark LUBM, Wordnet, BSBM, Barton, and DBpedia Infobox. And use 12 queries: 4 for LUBM, 2 for Wordnet, 2 for BSBM, 2 for Barton, 2 for DBpedia Infobox.

The Question Answering over Linked Data challenges[3] provide a series

[2]http://tap.stanford.edu
[3]http://qald.aksw.org

of queries to test Q&A approaches. Pound, Mika, and Zaragoza [27] addressed the task of ad-hoc object retrieval, as opposed to traditional ad-hoc document retrieval. They analyzed a real world Web query log from a semantic search point of view and proposed a semantic search query classification. Lastly, Balog and Neumayer [10] defined a test collection for entity search in DBpedia and summarized related work on entity search. Albeit attractive, in all these references, the proposed query workloads suffer from the same limitation - they retrieve individuals or lists of individuals - and do not fully explore the potentialities of RDF keyword search algorithms.

As well as keyword search systems, the critical aspect of NLIBD systems is their evaluation. However, these systems have not used benchmarks with many queries to evaluate the system. For example, Affolter et al. [28] created a cured list with ten natural language questions to assess 24 NLIB systems and show their strengths and weaknesses.

Some NLIBD systems use workloads with relational data and their queries over different domains in the cloud service. Saha et al. [29] uses geographical, academic, and financial data workloads to evaluate their system. For geographical domain, they used GEOWorkload contains 250 natural language queries over geographical data about the United States. For academic domain, they used MAS Workload for the academic domain, which consists of 196 natural language queries. Lately, for the financial domain, called FIN workload, contains 108 natural language queries.

State-of-the-art NLIBD systems inspire the creation of new benchmarks based on real datasets for evaluating the systems. For example, Arenas et al. [30] evaluated their system over an integrated database of the three sources: Cortellis drug data, a Thomson Reuterspatent dataset, and DrugBank. They generated 5000 random natural language questions with the help of users to complete questions.

There are some benchmarks with natural language queries for NLIB systems. For example, Franco et al. [31] present ExQuestions, a dataset providing multiple paraphrased questions using common-sense knowledge over graphs. This dataset contains 128,000 question-answer pairs with questions in natural language and provides templates for each question.

2.2 Keyword Search Systems

As for the keyword search itself, systems follow, basically, two distinct approaches. They can either take advantage of the declared schema of the dataset or disregard it and use the graph structure of the data to answer

queries. QUIOW [12] (an improved implementation of previous work [7]) and QUICK [23] are both schema-based systems, but they differ in that QUICK relies on user feedback while QUIOW is fully automated. Both systems rewrite the original keyword query into SPARQL queries, with the help of the schema, to return the final answers.

SPARK [22] is a graph-based system that uses techniques, such as synonyms from WordNet and string metrics, to match keywords with RDF graph nodes. The matched nodes are then connected by minimum spanning trees from which SPARQL queries are generated.

Elbassuoni and Blanco [25] described a technique to retrieve a set of subgraphs that match the keywords and to rank them based on statistical language models. Tran et al. [24] introduced the idea of generating summary graphs for the RDF graph to generate and rank candidate SPARQL queries. Lin et al. [19] summarized all the inter-entity relationships from the RDF data to translate keywords to SPARQL queries. Wen et al. [20] introduced a graph summarization technique that amounts to recovering an RDF schema from the RDF graph. Le et al. [26] also proposed to process keyword queries using an RDF graph summarization algorithm. Han et al. [18] described an algorithm that uses the keywords to first obtain elementary query graph building blocks and then applies a bipartite graph matching-based best-first search method to assemble the final query.

2.3 Natural Language Interface to Database - NLIBD

NALIR [32] is a system that uses an interactive natural language interface for relational databases. This system translates a complex natural language question into a SQL query, which may include aggregation and joins. The system is composed of two steps: First, they used an on-the-shelf natural language parser and created a represented by a parse tree. After, they correlated the elements of the parse tree with the concepts database.

Athena [29] is an ontology-driven system for natural language questions to relational databases. This system uses an ontology that describes semantic entities and their relationships in a domain. They proposed a two-stage approach: 1) Given a natural language question, they created an intermediate engine query, called Ontology Query Language (OQL), that uses database data and synonyms to map the tokens of natural language query to ontology elements. 2) is translated OQL into a SQL query, where is created relationships between the ontology elements and the database.

TR Discovery [30] translates the input question in the form of an English

sentence into SQL or SPARQL query. They provided auto-suggestions like autocomplete and prediction based on user input. Their suggestions are based on the relationships and entities in the dataset. The system parses the input question into a First Order Logic (FOL) representation and then translates the generated FOL into a parse tree.

Ferré [33] described an NLIDB for searching and updating an RDF store with context-free generative grammar. The system recognizes the keywords, performs a syntactic analysis based on a descending parser with the grammar rules, generates the logical language based on the definition in the grammar, and finally translates into the chosen formal language.

Ferre [34] developed a guided query builder for SPARQL using natural language for better understanding. The translation process for SPARKLIS is reversed: it translates possible SPARQL queries into a natural language such that the users can understand their choices.

Marginean [35] showed an NLIDB system for biomedical Linked Data that covers basic elements of SPARQL to support term constraints, aggregates, and negations.

3
Background and Definitions

This chapter contains basic definitions and concepts used in the next chapters. First, it describes the Resource Description Framework (RDF) as a data model to represent data. Then, it summarizes SPARQL, the language used to query RDF datasets. Next, it presents a natural language representation model, called BERT. Finally, it introduces additional definitions required in next chapters.

3.1
Resource Description Framework - RDF

The Resource Description Framework (RDF) is a flexible data model to represent data about resources [36]. Resources can be anything, including documents, people, physical objects, and abstract concepts. RDF represents data as triples (s, p, o), where s is called the *subject*, p is called the *predicate*, and o is called the *object* of the triple. A set $\mathcal{T}$ of triples is called an *RDF dataset*, or simply a *dataset*; $\mathcal{T}$ induces a graph $\mathcal{G}_{\mathcal{T}} = (N, E)$, where E is the set of subjects and objects of the triples in $\mathcal{T}$, and there is a directed edge from s to o, labeled with p, in S iff there is a triple (s, p, o) in $\mathcal{T}$.

The subjects of the triples can be IRIs or blank nodes, the predicates are always IRIs, and objects can be IRIs, literals, or blank nodes. An *International Resource Identifier*, or *IRI*, is a character string that complies with the RFC 3987 specification[1]. An IRI denotes something, either physical or abstract, in the world. For example, <`http://dbpedia.org/resource/Meryl_Streep`> represents the actress Meryl Streep in the DBpedia dataset. Note that IRIs are informed between < and > in RDF. A literal is either a typed or untyped value, for example, numbers, dates, strings, etc. and are informed between double-quotes. Unlike IRIs and literals, blank nodes do not identify things in the world, but they represent nodes in the RDF graph [37]. Some properties link resources with literals, and they are called *datatype properties* by the Web Ontology Language (OWL) [38], an RDF extension. Some other properties link resources, and they are called *object properties* [38]. While the former represent attributes of a resource, the latter represent relationships between resources.

[1] `https://www.rfc-editor.org/rfc/rfc3987`

Listing 3.1 shows a fragment of an RDF dataset using Turtle [39] notation. It induces the graphs of Figures 3.1–3.3. Turtle notation allows for aggregating triples with the same subject by separating pairs of predicate/object with semicolons. Also, the IRIs appear shortened with the indication of a namespace before a colon and without the diamond (<>) notation. Namespaces are IRI prefixes which are defined in the beginning of the dataset. Lines 6 and 9 represent the fact that the movie "Mamma Mia! Here We Go Again" (subject – <http://dbpedia.org/resource/Mamma_Mia!_Here_We_Go_Again>) starred (predicate – <http://dbpedia.org/ontology/starring>) Merryl Streep (object – <http://dbpedia.org/resource/Meryl_Streep>). Lines 8, 12, and 18 set friendly names to the IRIs of this triple. The graph in Figure 3.1 uses the friendly names as labels for the nodes and the edge induced by this triple.

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3  PREFIX dbr: <http://dbpedia.org/resource/>
4  PREFIX dbo: <http://dbpedia.org/ontology/>
5
6  dbr:Mamma_Mia!_Here_We_Go_Again
7      rdf:type dbo:Film;
8      rdfs:label "Mamma Mia! Here We Go Again";
9      dbo:starring dbr:Meryl_Streep.
10 dbr:Meryl_Streep
11     rdf:type dbo:Actress;
12     rdfs:label "Meryl Streep".
13 dbo:Actress rdfs:label "Actress".
14 dbo:Film rdfs:label "Film".
15 dbo:starring
16     rdfs:domain dbo:Work;
17     rdfs:range dbo:Actress;
18     rdfs:label "starring".
19 dbo:Film rdfs:subclassOf dbo:Work.
```

Listing 3.1: Fragment of DBpedia dataset using Turtle notation

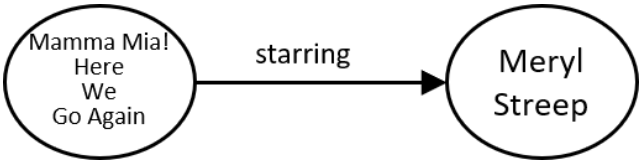


Figure 3.1: Representation of triple.

RDF also allows schema definitions. Lines 6–7 declare that "Mamma Mia! Here We Go Again" is an instance of the class Film, and lines 10–11 declare

that Meryl Streep is an instance of the class Actress. Figure 3.2 shows the graph induced by these definitions.

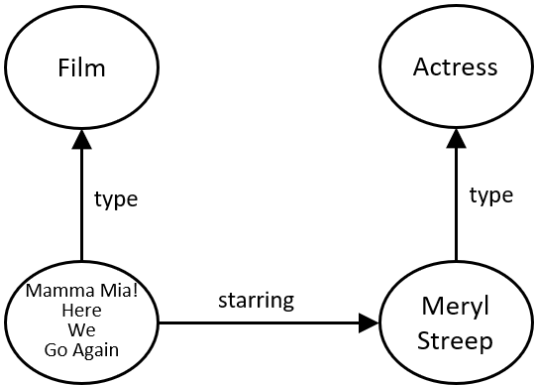


Figure 3.2: Example of a declaration of the class.

RDF can also specify class hierarchies, relationships and attributes. Line 19 declares that the class Film is a subclass of Work. Also, it is possible to define type restrictions about subjects and objects. Line 15–17 represent that the domain of <`http://dbpedia.org/ontology/starring`> is Work, and its range is Actress, i.e. the edges with that predicate must start in a resource of type Actress and end in a resource of type Work. Figure 3.3 shows the graph induced by these triples.

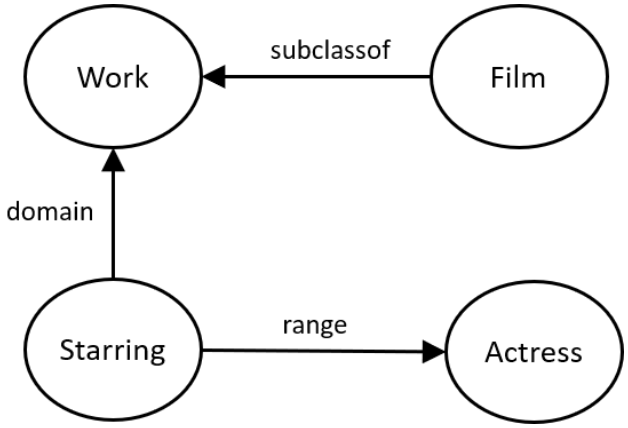


Figure 3.3: Domain and range example of class Work.

3.2 SPARQL: a query language for RDF

SPARQL is a query language for RDF datasets. The result of SPARQL queries can be either a relation or an RDF graph, but, for the sake of conciseness, we will present only the first one.

RDF datasets group triples in subsets, called dataset graphs. Dataset triples can then be viewed as quads: (g, s, p, o) where g is an identifier of the dataset graph to which the triple (s, p, o) belongs in the dataset.

Datasets can be conceptually modeled as a database of only one relational table with four columns, i.e., $dataset(graph, subject, predicate, object)$. One says that dataset tuples with the attribute graph unknown refer to triples that belong to a default graph, i.e., the default graph is the unidentified graph.

For the sake of simplicity, in this work, SPARQL queries will process the only table of a dataset, although there are means, such as using FROM and FROM NAMED clauses and federated queries, to join data from other datasets in the same query.

The basic syntax of a SPARQL query is a simple structure with two clauses: *SELECT* and *WHERE*. Listing 3.2 shows the syntax of a SPARQL query. The *SELECT* clause contains a list of variable names which specifies a relational projection over the resulting table of the *WHERE* clause. The WHERE clause contains a graph pattern expression that operates on the only dataset relation and returns another relation.

```
1 SELECT list of variables
2 WHERE { graph pattern }
```

Listing 3.2: Basic syntax of a SPARQL query

The simplest graph pattern expression is called a triple pattern which is of the form $\{vS\ vP\ vO.\}$, where each element vS, vP or vO can be a variable or a constant. A variable is a character string starting with the question mark character ?, such as "?s". A constant can be either an IRI, a blank node identifier or a literal. Recall that literals are valid only as objects. A triple pattern is equivalent to the relational expression shown in Equation 3-1:

$$R \leftarrow \pi_{(list\ of\ variables)}(\sigma_{\substack{subject=vS \\ And\ predicate=vP \\ And\ object=vO}}(dataset)) \tag{3-1}$$

where variables represent ANY VALUE in the relational selection condition, i.e., subject = ?s returns true for every tuple of the dataset, while subject = `<http://dbpedia.org/resource/Meryl_Streep>` returns true only for triples with this IRI as subject, and *list of variables* is the list of all variables used in the triple pattern. A variable maps the subject column, the predicate column, or the object column of the dataset table, when respectively used as the subject, the predicate, or the object of the triple pattern.

Triple patterns can be nested $\{vS_1\ vP_1\ vO_1.\ vS_2\ vP_2\ vO_2.\ ...\ vS_n\ vP_n\ vO_n.\}$ to express more complex expressions. A graph pattern is equivalent to the

following relational expression:

$$R_1 * R_2 * ... * R_n$$

where $R_i, i = 1, 2, ..., n$ are the resulting relations of each triple pattern individually. The nested triple patterns is then equivalent to a natural join between their resulting relations $R_1, R_2, ..., R_n$. Note that using the same variable names in different triple patterns will cause natural joins, while using distinct variable names will cause a cross product. There is no limit to the number of nested triple patterns.

SPARQL also allows to query dataset graphs. The triple pattern $\{GRAPH\ vG\ \{vS\ vP\ vO.\}\}$ is equivalent to the following relational expression shown in Equation 3-2:

$$R \leftarrow \pi_{(list\ of\ variables)}(\sigma_{\substack{graph=vG \\ Andsubject=vS \\ And\ predicate=vP \\ And\ object=vO}}(dataset)) \tag{3-2}$$

It is possible to apply a relational selection operation over a whole graph pattern. That is, let G be a graph pattern and R_G be its resulting relation, one wants to compute $\sigma_C(R_G)$. The SPARQL constructor to do this is FILTER. The Listing 3.3 shows a query that lists the labels of actors who starred a movie with its label containing the string "Mamma Mia!".

```
1 SELECT ?ls
2 WHERE {
3     ?s dbo:starring ?m.
4     ?m rdfs:label ?lm.
5     ?s rdfs:label ?ls.
6     FILTER regex(?lm, "Mamma Mia!")
7 }
```

Listing 3.3: Example of query lists the labels

3.3 BERT – Bidirectional Encoder Representations from Transformers

Unsupervised learning has been very successful in generating natural language sentence models. Indeed, such models produce better results in complex tasks, such as classification, translation, and query answering. Typically, one trains neural networks to produce contextual word representations, and then uses such representations to train other neural networks for specific tasks. The goal is to create similar representations for words with the same semantic value. Specific neural networks will not process words and sentences but,

instead, they will use their "meanings". Intuitively, different words and sentences with similar meanings are expected to have similar representations and processed by complex task-specific neural networks.

Autoregressive (AR) language modeling and autoencoding (AE) are two of the most successful methods to generate word representations. Autoregressive models seek to estimate, for each word, the probability that it occurs after the words preceding it in the sentence. The term "preceding" here is understood as a relative concept determined by the characteristics of a language, whether it is written from left to right or the other way around. Autoencoding methods, on the other hand, are not based on conditional probabilities but on trying to reconstruct the original sentence when some words are dropped. BERT [40] is a notable example of this second type of system.

Devlin et al. [40] and Yang et al. [41] argued that AE systems produce more significant benefits in training neural networks for specific tasks because they can produce word representations based on all the context of a sentence, that is, based on what occurs before and after a word. In fact, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" [40] became one of the most successful tools for generating semantic representations of words in natural language processing.

Contrasting with Devlin et al. [40], which introduced BERT and applied it to more complex tasks, this thesis uses only BERT representations of words and sentences.

The pre-trained BERT model generates embedding tokens from a sentence. The main advantage of using the embeddings the pre-trained BERT model generates is that they are context-dependent, unlike word2vec [42], which produces the same vector representation for words in the sentence. Listing 3.4 shows a code snippet that creates a token and sentence vector representation using the pre-trained BERT model. The code was inspired in Dhami Dharti [43].

```
1  tokenizer=BertTokenizer.from_pretrained('bert-base-uncased')
2
3  tokens = tokenizer.tokenize(sentence)
4  tokens_ids = tokenizer.convert_tokens_to_ids(tokens)
5  segments_ids = [1] * len(tokens)
6  tensor = torch.tensor([tokens_ids])
7  tensors = torch.tensor([segments_ids])
8
9  model = BertModel.from_pretrained('bert-base-uncased',
       output_hidden_states = True)
10
11 with torch.no_grad():
```

```
12      outputs = model(tensor, tensors)
13      hidden_states = outputs[2]
14
15  embeddings = torch.stack(hidden_states, dim=0)
16  embeddings = torch.squeeze(embeddings, dim=1)
17  embeddings = embeddings.permute(1,0,2)
18
19  token_vecs_sum = []
20  for token in embeddings:
21      sum_vec = torch.sum(token[-4:], dim=0)
22      token_vecs_sum.append(sum_vec)
23
24  sente_embedding = torch.mean(torch.stack(token_vecs_sum),
        dim=0)
```

Listing 3.4: Code of generating embeddings from bert

Line 1 loads the pre-trained BERT model. We have chosen the option uncased, which disregards cased characters.

Line 3 splits the sentence into tokens. These tokens are created according to BERT's vocabulary. Line 4 converts tokens into their respective ids. Line 5 creates a vector of the same size as the tokens vector full of "1', configuring BERT to process just one sentence. Line 6 and 7 create a tensor representation for the sentence.

Line 11–13 applies the BERT model to the sentence representation and take the hidden states. The hidden states contain the outputs of the neural network layers and have four dimensions, in the following order:

1. First dimension: neural network layer number (the first layer is the input layer, and the remaining 12 are the outputs of each subsequent layers).

2. The batch number: in this example is 1 because there is just one sentence.

3. The word/token number: corresponding numbers of tokens of the sentence. We represent this number with n.

4. The hidden unit/feature numbers (a vector with 768 positions).

Line 15 combines the output into a large tensor, which has four components that can be represented as vector with dimension $[13 \times 1 \times n \times 768]$. If we consider tokens over many sentences, it is hard for machines to process a vector of this dimension. Then, we need to reduce dimensionality, producing a single vector representing the embedding of each token from the sentence. To remedy this, there are some steps. Line 16 discards the batch dimension, resulting in a

vector of dimension $[13 \times n \times 768]$. Line 17 permutes the tokens dimension with the layer dimension, producing the hidden state with dimension $[n \times 13 \times 768]$.

Lines 19-22 shows how to compute token embeddings. Each word can have multiple tokens. Tokens of the same word begin with the symbol ## except for the first token. For example, the tokens for the word "embedding" are [em, ##bed, ##ding]. The embedding of a word is the average embedding of the word tokens. Line 24 show how to compute the sentence embedding, which is the average embedding of the sentence words.

Dhami Dharti [43] and Alammar Jay [44] proposed six ways of dimensionality reduction and evaluated each of them in a named-entity recognition task. They found that the two most successful ways of generating embedding are summing up the last four layers of the neural network. This thesis uses the sum of the last four layers because the vectors are smaller and easier to handle, and its performance is one of the best.

3.4 Graph-related Definitions, Queries, and Answers

3.4.1 Graph-related Definitions

As defined in Section 3.1 an *RDF dataset* is a set $\mathcal{T}$ of RDF triples (s, p, o), which is equivalent to an edge-labeled directed graph $\mathcal{G}_\mathcal{T}$ (Figure 3.6(a)) such that the set of nodes of $\mathcal{G}_\mathcal{T}$ is the set of RDF terms that occur as subject or object of the triples in $\mathcal{T}$ and there is an edge (s, o) in $\mathcal{G}_\mathcal{T}$, labeled p, iff the triple (s, p, o) occurs in $\mathcal{T}$.

An RDF dataset $\mathcal{T}$ is also equivalent to a *knowledge base* $\mathcal{KB} = TBox \cup ABox$ in which a subset of triples, $TBox_\mathcal{T} \subseteq \mathcal{T}$, defines the RDF schema (Lines 13–19 of Listing 3.1) and the $ABox_\mathcal{T}$ defines the data (Lines 6–12 of Listing 3.1).

The *resource graph* induced by a subset $\mathcal{T}' \subseteq \mathcal{T}$ is the subgraph $\mathcal{RG}_{\mathcal{T}'}$ of $\mathcal{G}_{\mathcal{T}'}$ obtained by dropping all literal nodes from $\mathcal{G}_{\mathcal{T}'}$. The nodes in $\mathcal{RG}_{\mathcal{T}'}$ are the *resources* of $\mathcal{T}'$

The *entity graph* induced by a subset $\mathcal{T}' \subseteq \mathcal{T}$ is the subgraph $\mathcal{EG}_{\mathcal{T}'}$ of $\mathcal{G}_{\mathcal{T}'}$ obtained by dropping all literal nodes from the graph $\mathcal{G}_{\mathcal{T}'-TBox}$. The nodes in $\mathcal{EG}_{\mathcal{T}'}$ are the *entities* of $\mathcal{T}'$.

The *neighborhood* of distance d of an entity e in $\mathcal{T}$, denoted ${}^d\mathcal{N}^e_\mathcal{T}$, is the set of all nodes visited in a breadth-first walk of distance d starting from e in $\mathcal{EG}_\mathcal{T}$.

The *schema graph* of $\mathcal{T}$, denoted $\mathcal{TG}_{\mathcal{T}}$, is the graph in which the set of nodes are the named classes in *TBox* [45], and there is an edge (A, B), labeled p in $\mathcal{TG}_{\mathcal{T}}$, iff there is an edge labeled p from an entity of type A to an entity of type B in *ABox*, [45], the domain of p contains a subset of A, and the range of p contains a subset of B, such that A and B are either asserted or inferred classes. Such definitions aim at dealing with multiple class assertions.

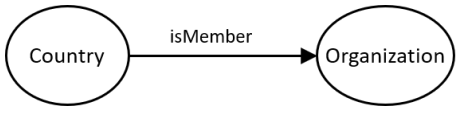
For example, let $ABox_{\mathcal{T}'''} = \{(s, type, A), (o, type, B), (s, type, C), (s, p, o)\}$ be a dataset. Without the restrictions, one could add the edges (A, B) and (C, B) to $\mathcal{TG}_{\mathcal{T}'''}$. However, if the $TBox_{\mathcal{T}'''}$ defined restrictions on the domain and range of p, such as $TBox_{\mathcal{T}'''} = \{(p, domain, A), (p, range, B)\}$, the only valid edge for $\mathcal{TG}_{\mathcal{T}'''}$ would be (A, B). Note that $\mathcal{TG}_{\mathcal{T}}$ is not a replacement for the $\mathcal{G}_{TBox}$, it is just a convenient way of generating graph patterns for extracting keywords as shown later in Chapter 4.1.

A *schema cutout* of $\mathcal{TG}_{\mathcal{T}}$, or simply a *cutout*, induced by ${}^{d}\mathcal{N}^{e}_{\mathcal{T}}$, denoted ${}^{d}\mathcal{CG}^{e}_{\mathcal{T}}$, is the graph resulting from dropping nodes from $\mathcal{TG}_{\mathcal{T}}$ that are not classes, either asserted or inferred, of any entity in ${}^{d}\mathcal{N}^{e}_{\mathcal{T}}$.

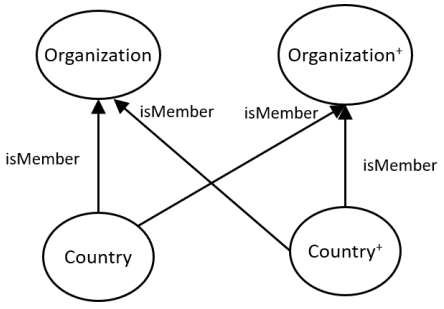
The *one-degree mirroring graph* of ${}^{d}\mathcal{CG}^{e}_{\mathcal{T}}$, denoted ${}^{1,d}\mathcal{CG}_{\mathcal{T}}$, is a graph that has a mirror node A^{+} for each node A in ${}^{d}\mathcal{CG}^{e}_{\mathcal{T}}$ and that, if there is an edge (A, B) in ${}^{d}\mathcal{CG}^{e}_{\mathcal{T}}$, there will be the edges (A, B), (A^{+}, B^{+}), (A^{+}, B), and (A, B^{+}) in ${}^{1,d}\mathcal{CG}_{\mathcal{T}}$. Figure 3.4(b) shows an example of a one-degree mirroring graph for the cutout schema in Figure 3.4(a).

Likewise, a *two-degree mirroring graph* ${}^{2,d}\mathcal{CG}_{\mathcal{T}}$, will have nodes A, A^{+}, and A^{++} and the edges (A, B), (A^{+}, B^{+}), (A^{++}, B^{++}), and so on.

An *n-degree mirroring graph* ${}^{n,d}\mathcal{CG}_{\mathcal{T}}$ can thus be inductively defined, with ${}^{0,d}\mathcal{CG}_{\mathcal{T}} = {}^{d}\mathcal{CG}_{\mathcal{T}}$. The sets of mirrored classes, such as $\{A, A^{+}, A^{++}, ...\}$, are called *mirrored sets*. The symbols A^{+}, A^{++}, ... are distinct aliases for the same class A.



3.4(a): A schema cutout graph $\mathcal{G}$.



3.4(b): The one-degree mirroring graph of $\mathcal{G}$.

Figure 3.4: Mirroring graph example.

A set of triples $\mathcal{T}' \subseteq \mathcal{T}$ *induces a path* $\pi = (s_0, p_0, s_1, p_1, s_2, ..., s_n, p_n, s_{n+1})$

in $\mathcal{G}_\mathcal{T}$ iff, for each $i \in [0, n]$, either $(s_i, p_i, s_{i+1}) \in \mathcal{T}'$ or $(s_{i+1}, p_i, s_i) \in \mathcal{T}'$. We also say that (s_i, p_i, s_{i+1}) (or (s_{i+1}, p_i, s_i)) *occurs* in π. Note that we assume that paths in $\mathcal{G}_\mathcal{T}$ may traverse arcs in both directions. A path $\pi = (s_0, p_0, s_1, p_1, s_2, ..., s_n, p_n, s_{n+1})$ *begins* (resp. *ends*) on a resource r iff $r = s_0$ or $r = p_0$ (resp. $r = s_{n+1}$ or $r = p_n$).

Recall that n-ary relationships, with $n > 2$, and relationships that have attributes or that participate in other relationships have to be reified in RDF. The standard treatment of reified relationships is described in [46] and involves the definition of a *reification class*, a set of *reification object properties*, and a set of *reification axioms* that encode, in Description Logic, the essence of reification.

For example, Figure 3.5 shows a fragment of the relationship between Country and Language in the Mondial[2] dataset, which is reified with reification class SpokenBy. Note that each entity s_i represents a reified relationship and is an instance of SpokenBy (indicated via type). Also note that s_i is associated with an instance of Country, via the languageInfo object property, and an instance of Language, via onLanguage object property.

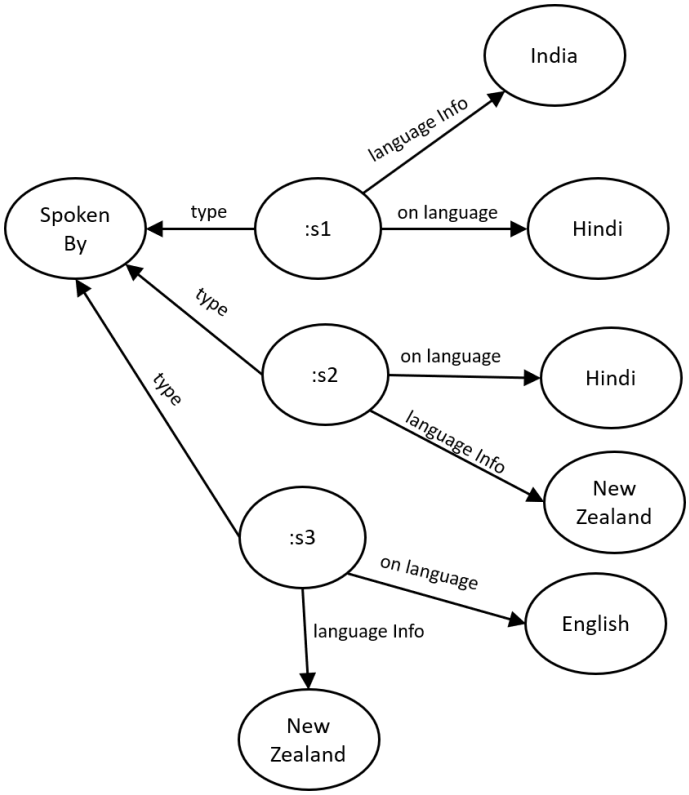


Figure 3.5: Example Reified Class in Mondial Database.

[2]https://www.dbis.informatik.uni-goettingen.de/Mondial/

3.4.2 Keyword Queries and Answers

A *keyword query*, which represents an information need, is a set $\mathcal{K}$ of literals.

A *matching function* in the context of keyword queries is a function σ_L that maps pairs of literals into Boolean values. If $\sigma_L(L_1, L_2) = True$ then we say that L_1 and L_2 *match*. The exact matching function adopted is left unspecified at this point, and varies from system to system. Typically, it is based on string similarity, but it can use more sophisticated strategies.

A triple $(s, p, o) \in \mathcal{T}$ is a *matching triple* for $\mathcal{K}$ iff o is a string literal that matches at least one keyword in $\mathcal{K}$.

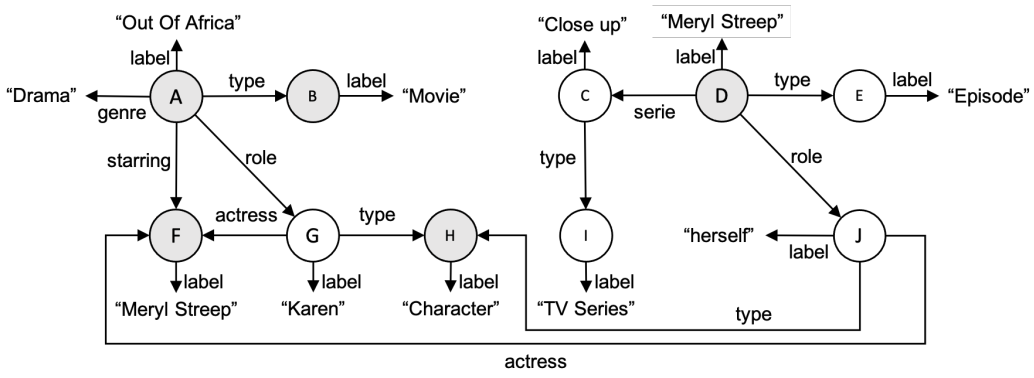
We also say that s is a *matching resource* for $\mathcal{K}$, the set $\mathcal{M}_s$ of all matching triples in $\mathcal{T}$ for $\mathcal{K}$ whose subject is s is the set of *matching triples* of s in $\mathcal{T}$, and the graph induced by $\mathcal{M}_s$ is the *matching subgraph* of s in $\mathcal{G}_\mathcal{T}$. Note that s may occur as the subject, property, or object of a triple in $\mathcal{T}$, but s is always the subject of a matching triple.

A set of triples $\mathcal{A} \subseteq \mathcal{T}$ is an *answer for* $\mathcal{K}$ *over* $\mathcal{T}$ iff $\mathcal{A}$ can be partitioned into two subsets $\mathcal{A}'$ and $\mathcal{A}''$ such that: (i) $\mathcal{A}'$ is the set of all matching triples for $\mathcal{K}$ in $\mathcal{A}$; let $\mathcal{R}_\mathcal{A}$ be the set of subjects of such triples; (ii) $\mathcal{RG}_{\mathcal{A}''}$, the resource graph induced by $\mathcal{A}''$, is connected and contains all resources in $\mathcal{R}_\mathcal{A}$. Condition (i) captures keyword matches and Condition (ii) indicates that an answer must connect the matching resources by paths in $\mathcal{RG}_{\mathcal{A}''}$. We also say that $\mathcal{R}_\mathcal{A}$ is the set of *matching resources* of $\mathcal{A}$ and that $\mathcal{RG}_{\mathcal{A}''}$ is the *connectivity graph* of $\mathcal{A}$. Figure 3.6 shows an RDF graph, Figure 3.6(a), and three answers, Figures 3.6(b)–3.6(d), for the keyword query $\mathcal{MS}$ = {“character”, “meryl”, “streep”, “movie”, “out”, “africa”}.

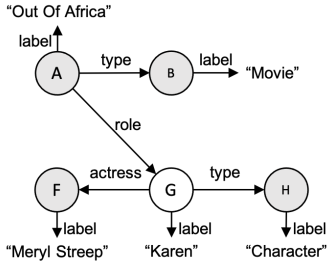
Figure 3.7 shows another example of a dataset and answers for the keywords $\mathcal{MS}$ = {“mauritius”, “india”}. Figure 3.7(a) is the dataset fragment and Figures 3.7(b) and 3.7(c) show two possible answers.

An *annotated answer* is an answer such that each matching triple is annotated with the keywords it matches. The specific way to represent the annotations is left unspecified at this point. It suffices to remark that such annotations help relate triples in an answer with keywords in the query.

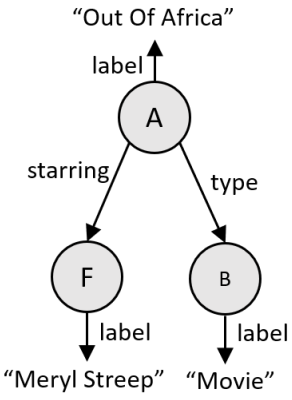
This definition of answer is less stringent than those introduced by Bhalotia et al. [47], Hristidis and Papakonstantinou [48], Kimelfeld and Sagiv [49] since it neither requires all keywords to be matched nor includes a minimality criterion. For later reference, we say that an answer $\mathcal{A}$ for $\mathcal{K}$ over $\mathcal{T}$ is *minimal* iff there is no proper subset $\mathcal{B}$ of $\mathcal{A}$ such that $\mathcal{B}$ is an answer for $\mathcal{K}$ and $\mathcal{B}$ and $\mathcal{A}$ have the same set of matching resources.



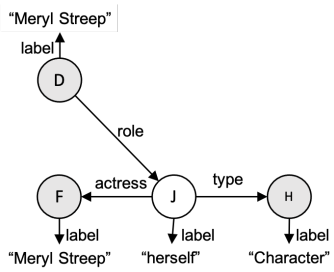
3.6(a): The induced graph of an RDF dataset $\mathcal{T}$.



3.6(b): Answer $\mathcal{A}_1$.



3.6(c): Answer $\mathcal{A}_2$.



3.6(d): Answer $\mathcal{A}_3$.

Figure 3.6: Example of an RDF graph and three answers for the information need "*character of Meryl Streep in the movie Out of Africa*".

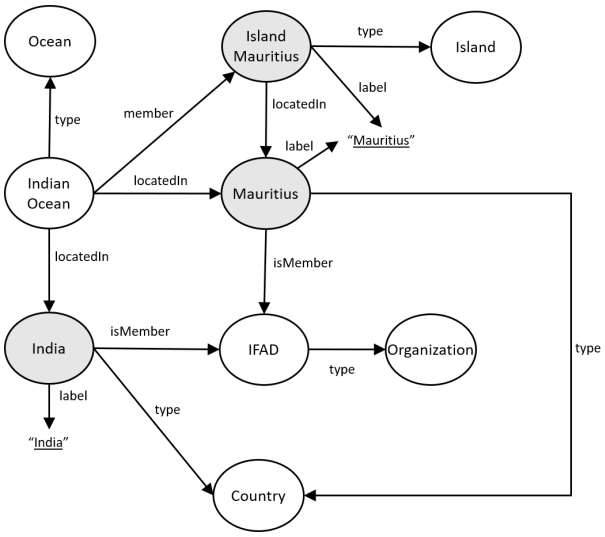
Finally, we can informally state the *RDF KwS-Problem* as: "Given an RDF dataset $\mathcal{T}$ and a keyword query $\mathcal{K}$, find an answer $\mathcal{A}$ for $\mathcal{K}$ over $\mathcal{T}$, preferably, with as many keyword matches as possible and with the smallest set of triples as possible".

3.4.3 Natural Language Queries and Answers

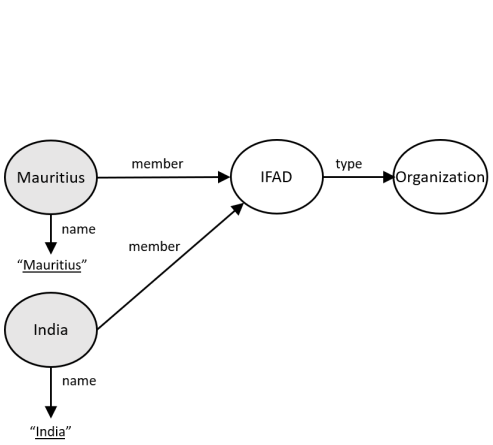
A *natural language query* is simply a sentence S in a natural language, such as *"What is the character of Meryl Streep in the movie Out of Africa?"*.

The definition of a *matching function* σ_N for natural language queries is quite different from the definition of matching functions for keyword queries. However, to maintain the next definitions close to those in Section 3.4.2, we consider σ_N as a function that maps pairs (f, t), where f is a natural language sentence fragment and t is a triple, into Boolean values. If $\sigma_N(f, t) = True$ then we say that f and t *match*.

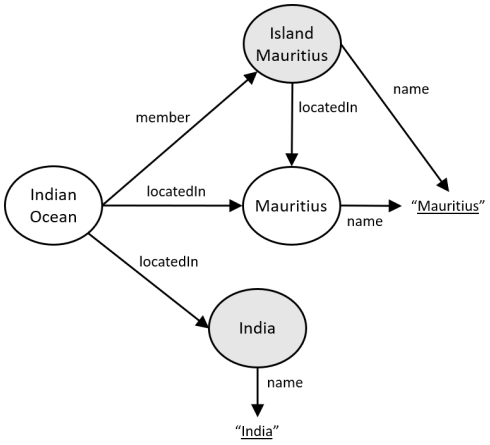
Given a natural language query S, we say that t is a *matching triple* for



3.7(a): The fragment of Mondial dataset graph.



3.7(b): Ground Truth answer.



3.7(c): Additional geographical answer.

Figure 3.7: Example of Mondial fragment and two answers for the information need "*mauritius india*".

S iff $\sigma_N(f, t) = True$, for some fragment f of S.

Again, the exact matching function σ_N adopted is left unspecified at this point, but we observe that the application of σ_N is typically preceded by a natural language processing stage that decomposes the query sentence into fragments. The benchmark creation method described in this thesis is not dependent on the matching function, since it is concerned with synthesizing natural language queries, as discussed in Section 4.2.

The notions of *matching resource*, *answer*, and *annotated answer* then follow as in Section 3.4.2. However, we observe that the notion of annotated answer in this case helps identify which triples in an answer match which fragments of the natural language query.

Figure 3.6 shows an RDF graph and three answers for the natural language query S = "*What is the character of Meryl Streep in the movie Out*

of Africa?".

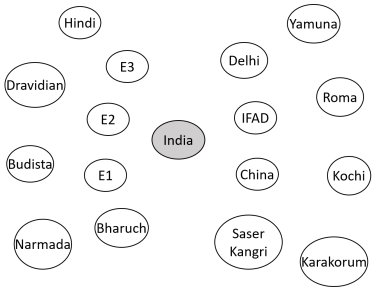
Finally, we can informally state the *RDF NLI-Problem* as: "Given an RDF dataset $\mathcal{T}$ and a natural language query S, find an answer $\mathcal{A}$ for S over $\mathcal{T}$, preferably, with as many matches as possible and with the smallest set of triples as possible".

4
A Method for Generating Benchmarks

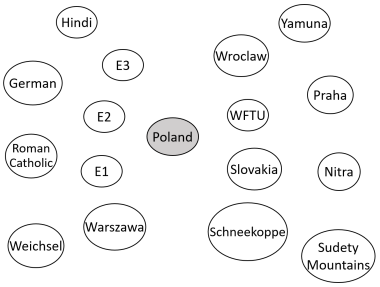
4.1
Computing Keyword Queries

This section describes, with the help of examples, how to automatically generate sets of keyword queries for a given RDF dataset $\mathcal{T}$.

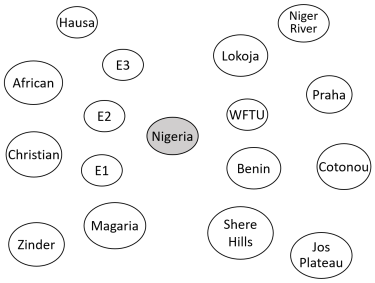
The method consists of: (i) choosing a set $\mathcal{I}$ of relevant entities, called inducers, as starting points of the query generation; (ii) computing the inducers' neighborhood N, which are subgraphs centered in the node $i \in I$; (iii) compute distinct graph patterns connecting entities to i; and (iv) extracting keywords from different instantiations of graph patterns.



4.1(a): Fragment of ${}^2\mathcal{N}_{\mathcal{T}}^{India}$.



4.1(b): Fragment of ${}^2\mathcal{N}_{\mathcal{T}}^{Poland}$.



4.1(c): Fragment of ${}^2\mathcal{N}_{\mathcal{T}}^{Nigeria}$.

Figure 4.1: Neighborhood fragments.

An *inducer function* for $\mathcal{T}$ is a function that maps $\mathcal{T}$ into a set $\mathcal{I}$ of entities of $\mathcal{T}$, called *inducers*. Such function should follow requirements consistent with the benchmark's purpose, i.e., it should select entities from the information domain in question with appropriate relevance scores. High scores induce keywords from relevant entities, while low scores can challenge RDF-

KwS systems by inducing keywords from less relevant entities. For example, let $\mathcal{T}$ be the Mondial RDF dataset[1]. An inducer function for $\mathcal{T}$ would select the *top-k and bottom-k* countries according to their infoRank score [2]. InfoRank estimates entities' relevance for users based on three intuitions: 1) important things have lots of datatype properties, 2) important things are surrounded by other important things, and 3) few good friends are better than many acquaintances. The *first step* of the process for generating queries is to select a set of inducers.

The *second step* is to compute the inducers' neighborhoods ${}^{d}\mathcal{N}^{i}_{\mathcal{T}}$, $i \in I$. Figure 4.1(a) shows an example fragment of the neighborhood ${}^{2}\mathcal{N}^{India}_{\mathcal{T}}$ of the country *India*, which is the 8th country with the most significant infoRank score in the Mondial dataset. We have chosen India as an example because it can lead the query generation process to recreate the 43^{rd} query in Coffman's benchmark. The following examples are also chosen with the same idea. The rationale is to demonstrate that the proposed approach could generate the same queries as a known benchmark. Note, though, that an automatic running of the approach would not necessarily generate the same queries as another benchmark. It will depend on arbitrary choices, such as selecting relevant inducers to be used in the process.

The *third step* is to compute the n-degree mirroring graphs ${}^{n,d}\mathcal{CG}^{i}_{\mathcal{T}}$, and the set of all possible minimal Steiner trees in each one. Figure 4.2 shows a fragment of the cutout schema ${}^{2}\mathcal{CG}^{India}_{\mathcal{T}}$ which is coincidentally similar to ${}^{2}\mathcal{CG}^{Poland}_{\mathcal{T}}$ and ${}^{2}\mathcal{CG}^{Nigeria}_{\mathcal{T}}$. The Steiner trees induce different SPARQL graph patterns P_i that correlate entities from ${}^{d}\mathcal{N}^{i}_{\mathcal{T}}$. Note that n is a user-defined parameter that allows one to control the cardinality of elements of the same type in the graph patterns.

These graph patterns, called *answer graph patterns*, represent typical answer templates for keyword sets. For example, Figure 4.3(a) shows a Steiner tree from ${}^{1,2}\mathcal{CG}^{India}_{\mathcal{T}}$ that induces the graph pattern in Listing 4.1. Note that the expression *typical answer template* refers to the process that guides the query generation. It does not mean that it is the most common or desired answer pattern among all valid answers. There can be many more valid answer graph patterns that search systems can find for the same keyword set, as explained in Section 4.3.

Each node of a Steiner tree maps to distinct SPARQL variables, the edges of the trees straightforwardly derive the joining triple patterns (Lines 6–7 in Listing 4.1), and the types of the nodes filter out entities of their respective types. We call the joining triple patterns the *core graph pattern*

[1] `http://www.dbis.informatik.uni-goettingen.de/Mondial/`

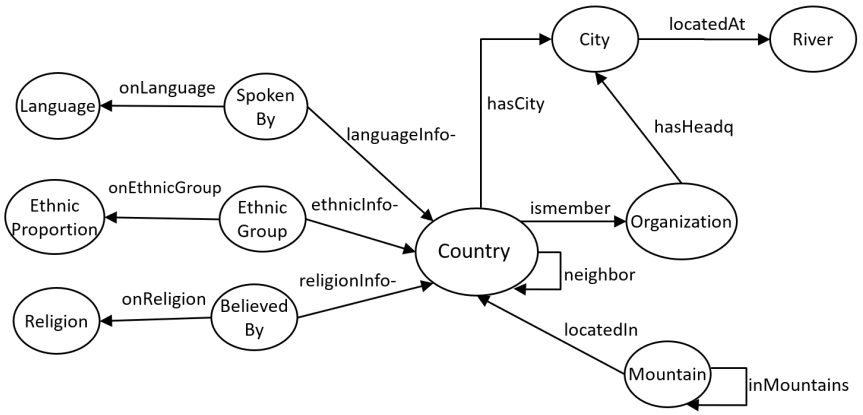
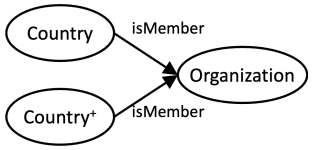


Figure 4.2: A fragment of the cutout schema ${}^{2}\mathcal{CG}_{\mathcal{T}}^{India}$ induced by the neighborhood graph of India, which is coincidentally similar to ${}^{2}\mathcal{CG}_{\mathcal{T}}^{Poland}$ and ${}^{2}\mathcal{CG}_{\mathcal{T}}^{Nigeria}$.



4.3(a): A Steiner tree for ${}^{1,2}\mathcal{CG}_{\mathcal{T}}^{India}$.

4.3(b): A Steiner tree for ${}^{0,2}\mathcal{CG}_{\mathcal{T}}^{Poland}$.

4.3(c): A Steiner tree for ${}^{0,2}\mathcal{CG}_{\mathcal{T}}^{Nigeria}$.

Figure 4.3: Example of Steiner trees.

of the mirroring graph. The bind expression determines that solutions must contain the inducer and the filter dismiss solutions with repetitive entities. This last restriction aims to dismiss solutions such as `?s1=:IND` and `?s2=:IND`, where `:IND` is the India's URI. It seems not to make sense in this pattern. We opted to automatically generate the full mutual inequality filter rather than a more specific one such as `FILTER (?s1!=?s2)` to avoid more sophisticated inferences, although this strategy can dismiss good repetitions. For the sake of query generation, it is not required to generate all possible ones.

Keywords are then extracted for each entity, property, and class in the patterns by instantiating P_i from $\mathcal{T}$ and retrieving values of their datatype properties. The query in Listing 4.2 shows the value extraction for the pattern in Listing 4.1, and Table 4.1 shows fragments of two binding sets from the Mondial dataset. We assume that all terminological elements have a single property `rdfs:label`. More complex queries could avoid such assumptions, which are disregarded here for conciseness. Lines 12–14 extract label values for schema elements in the pattern. Lines 16–27 extract label values from all properties for each entity in the pattern. Lines 29–38 extract literal values from all properties for each entity in the pattern.

```
{
    BIND (:IND as ?s1) # sets India's URI
    ?s1 a :Country.
    ?s2 a :Country.
    ?s3 a :Organization.
    ?s1 :isMember ?s3.
    ?s2 :isMember ?s3.
    FILTER (?s1!=?s2 && ?s1!=?s3 && ?s2!=?s3)
}
```

Listing 4.1: Answer Graph Pattern for keywords {"*mauritius*", "*india*"}

```
SELECT ?l1 ?l2 ?l3 ?l4 ?l5 ?l6 ?v1 ?v2 ?v3
WHERE {
    {
        BIND (:IND AS ?s1) # sets India's URI
        ?s1 a :Country.
        ?s2 a :Country.
        ?s3 a :Organization.
        ?s1 :isMember ?s3.
        ?s2 :isMember ?s3.
        FILTER (?s1!=?s2 && ?s1!=?s3 && ?s2!=?s3)
    }
    :Country rdfs:label ?l1.
    :Organization rdfs:label ?l2.
    :isMember rdfs:label ?l3.

    OPTIONAL
    {SELECT ?s1 (group_concat(DISTINCT ?l) AS ?l4)
    WHERE {?s1 ?p ?o. ?p rdfs:label ?l.}
    GROUP BY ?s1}
    OPTIONAL
    {SELECT ?s2 (group_concat(DISTINCT ?l) AS ?l5)
    WHERE {?s2 ?p ?o. ?p rdfs:label ?l.}
    GROUP BY ?s2}
    OPTIONAL
    {SELECT ?s3 (group_concat(DISTINCT ?l) AS ?l6)
    WHERE {?s3 ?p ?o. ?p rdfs:label ?l.}
    GROUP BY ?s3}

    {SELECT ?s1 (group_concat(?o) AS ?v1)
    WHERE {?s1 ?p ?o FILTER isLiteral(?o)}
    GROUP BY ?s1}
    OPTIONAL
    {SELECT ?s2 (group_concat(?o) AS ?v2)
    WHERE {?s2 ?p ?o FILTER isLiteral(?o)}
    GROUP BY ?s2}
    {SELECT ?s3 (group_concat(?o) AS ?v3)
```

```
37      WHERE {?s3 ?p ?o FILTER isLiteral(?o)}
38      GROUP BY ?s3}
39  }
```

Listing 4.2: Query for retrieving property values for keywords {"*mauritius*", "*india*"}

?l1	?l2	?l3	...	?v1	?v2	?v3
"Country"	"Organization"	"is member"	...	"India..."	"Mauritius..."	"IFAD"
"Country"	"Organization"	"is member"	...	"India..."	"Mauritius..."	"G-77"

Table 4.1: A fragment of the binding sets for the query in Listing 4.2

.

Each line of Table 4.1 corresponds to a different entity instantiation set and, therefore, it may be considered a different answer for the pattern. Still, each line may produce distinct keyword sets.

Table 4.1 enables benchmark developers to create queries, such as queries number 43, 30, and 27 of Coffman's benchmark for Mondial, $\mathcal{K}_{Coff\text{-}Q43}$ = {"*mauritius*", "*india*"}, $\mathcal{K}_{Coff\text{-}Q30}$ = {"*poland*", "*language*"}, and $\mathcal{K}_{Coff\text{-}Q27}$ = {"*nigeria*", "*gdp*"}, respectively. The first query contains keywords for the names of two countries, the second query contains keywords for the name of a country and the name of an object property, and the third contains keywords for the name of a country and one of its datatype properties (Gross Domestic Product - GDP).

The *fourth and final step* of the keyword-query generation is to extract keywords from the bindings. For example, the 43rd Coffman's query, $\mathcal{K}_{Coff\text{-}Q43}$ = {"*mauritius*", "*india*"}, can be created by extracting the literals from the bindings for `?v1` and `?v2`.

The SPARQL query in Listing 4.3 can generate Coffman's ground truth for this query. Each created named graph is considered a different solution. This ground truth is adapted from Coffman's benchmark and our previous answer definition for RDF-KwS since Coffman's benchmark is built for relational databases and does not define answers as graphs.

```
1  INSERT {
2      GRAPH ?g {
3          ?s1 :isMember ?s3; ?p1 ?v1.
4          ?s2 :isMember ?s3; ?p2 ?v2.
5      }
6  }
7  WHERE {
8      BIND (UUID() AS ?g)
9      {
```

```
10          ?s1 a :Country.
11          ?s2 a :Country.
12          ?s3 a :Organization.
13          ?s1 :isMember ?s3.
14          ?s2 :isMember ?s3.
15          FILTER (?s1!=?s2 && ?s1!=?s3 && ?s2!=?s3)
16      }
17      ?s1 ?p1 ?v1.
18      ?s2 ?p2 ?v2.
19
20      FILTER (REGEX(?v1,"India","i") &&
21              REGEX(?v2,"Mauritius","i"))
22  }
```

Listing 4.3: Query to generate the Coffman's ground truth for keywords {"*mauritius*","*india*"}

The query in Listing 4.3 is not necessarily the query an RDF-KwS system would generate to answer $\mathcal{K}_{Coff\text{-}Q43}$. It simply shows an intuition that the method can generate queries similar to Coffman's associated with the same ground truth. Section 4.3 defines the actual ground truths.

Each tuple of bindings in Table 4.1 can derive many more keyword queries, each literal combination would be a different query. However, some combinations are not allowed: those with no literals coming from property values or classes of any leaf entity. *Leaf entities* are instances of leaf classes. *Leaf classes* are those connected by just one edge in Steiner trees. The leaf entities for the Steiner tree in Figure 4.3(a) are bound to the variables `?s1` and `?s2` in Listing 4.1. They are instances of the class $Country$. Recall that $Country^+$ is an alias to the class $Country$. We call this restriction the *must-include-all-leaves rule*. Such restriction is necessary because keyword search systems usually try to interconnect nodes linked to the given keywords. For example, Coffman's 43rd query intends to connect the nodes corresponding to India and Mauritius. Search systems would try to find paths to connect these nodes. Eventually, they would find IFAD and G-77, representing international organizations in which both countries participate. If, on the other hand, search systems used the keywords $\{India, IFAD\}$, the algorithms could never find Mauritius since a path interconnecting India and IFAD traversing Mauritius could not exist. Those keywords then could not be appropriate for the pattern in Figure 4.3(a).

Because of the combinatorial nature of this problem, we propose not to materialize every keyword combination. Instead, we extract keywords from every binding for each binding set (each line of Table 4.1) and leave the generation of appropriate keyword combinations to benchmark developers. The

method outputs for each inducer a set of pairs (K, G), called *keyword query generators*, such that K is a set of bindings, in the form of Table 4.1, and G is an answer graph pattern. The pairs (K, G) allow developers to follow the *must-include-all-leaves rule* to generate their own keyword sets.

Query generators are also convenient to compare different benchmarks in terms of the variety of expected answers. Sets of expected answers that show considerable variety pose more challenges to search engines. For example, from the 50 queries provided by Coffman for the Mondial database, one can count just 14 answer patterns. Indeed, for instance, Queries 36–45 aim at discovering relationships between two countries and international organizations and, therefore, have the same answer pattern. Differently from the existing benchmarks, the proposed method can generate many distinct answer patterns.

A second example illustrates a limitation of the method and how it can generate a query similar to Query 30 of Coffman's benchmark for Mondial: $\mathcal{K}_{Coff\text{-}Q30} = \{$"*poland*", "*language*"$\}$. This query aims at finding the language spoken in Poland and could be generated with the Steiner tree in Figure 4.3(b). Poland is the name of an instance of type `:Country` and language is the label of the property `:onLanguage` with domain `:SpokenBy` which points to the Language entity `:Polish`. However, the must-include-all-leaves rule requires one to extract keywords from property values of `:SpokenBy` instances or from the class itself. If one replaced the original query with $\{$"*poland*", "*spoken*", "*language*"$\}$ the answer graph pattern would be that in Listing 4.4, the extraction query that in Listing 4.5, the binding table that in Table 4.2, and the ground truth that in Listing 4.6. The keywords would be extracted from `?l2`, `?l5`, and `?v1`.

```
1 {
2     BIND (:PL as ?s1) # sets Poland's URI
3     ?s1 a :Country.
4     ?s2 a :SpokenBy.
5     ?s2 :languageInfo- ?s1.
6     FILTER (?s1!=?s2)
7 }
```

Listing 4.4: Answer graph pattern for keywords $\{$"*poland*", "*spoken*", "*language*"$\}$

```
1 SELECT ?l1 ?l2 ?l3 ?l4 ?l5 ?v1 ?v2
2 WHERE {
3     {
4         BIND (:PL AS ?s1) #sets Poland's URI
5         ?s1 a :Country.
6         ?s2 a :SpokenBy.
7         ?s2 :languageInfo- ?s1.
8         FILTER (?s1!=?s2)
```

```
9       }
10      :Country rdfs:label ?l1.
11      :SpokenBy rdfs:label ?l2.
12      :languageInfo- rdfs:label ?l3.
13
14      OPTIONAL
15      {SELECT ?s1 (group_concat(DISTINCT ?l) AS ?l4)
16      WHERE {?s1 ?p ?o. ?p rdfs:label ?l.}
17      GROUP BY ?s1}
18      OPTIONAL
19      {SELECT ?s2 (group_concat(DISTINCT ?l) AS ?l5)
20      WHERE {?s2 ?p ?o. ?p rdfs:label ?l.}
21      GROUP BY ?s2}
22
23      {SELECT ?s1 (group_concat(?o) AS ?v1)
24      WHERE {?s1 ?p ?o FILTER isLiteral(?o)}
25      GROUP BY ?s1}
26      {SELECT ?s2 (group_concat(?o) AS ?v2)
27      WHERE {?s2 ?p ?o FILTER isLiteral(?o)}
28      GROUP BY ?s2}
29  }
```

Listing 4.5: Query for retrieving property values for keywords {"*poland*", "*spoken*", "*language*"}

```
1   INSERT {
2       GRAPH ?g {
3           ?s2 a ?cls.
4           ?s2 :languageInfo- ?s1.
5           ?s2 ?p2 ?s3.
6           ?s1 ?p1 ?v1.
7           ?cls rdfs:label ?l2.
8       }
9   }
10  WHERE {
11      BIND (UUID() AS ?g)
12      {
13          ?s1 a :Country.
14          ?s2 a ?cls.
15          ?s2 :languageInfo- ?s1.
16          FILTER (?s1!=?s2)
17      }
18      ?s1 ?p1 ?v1.
19      ?cls rdfs:label ?l2.
20      ?s2 ?p2 ?s3.
21      ?p2 rdfs:label ?l5.
22
23
```

```
24      FILTER (REGEX(?v1,"Poland","i") &&
25              REGEX(?l2,"Spoken","i") &&
26              REGEX(?l5,"language","i"))
27  }
```

Listing 4.6: Query for generate the Coffman's ground truth for keywords {"*poland*", "*spoken*", "*language*"}

?l1	?l2	...	?l5	?v1	...
"Country"	"Spoken By"	...	"... on language..."	"Poland..."	...

Table 4.2: A fragment of the binding sets for the query in Listing 4.5.

The last example generates Query 27 of Coffman's benchmark for Mondial: $\mathcal{K}_{Coff\text{-}Q27} = \{$"*nigeria*", "*gdp*"$\}$. Nigeria is the 15th country by ranking with the infoRank score, and *gdp* is the Gross Domestic Product datatype property. The Steiner tree is in Figure 4.3(c), the answer patterns are in Listing 4.7, the extraction query is in Listing 4.8, the binding table is Table 4.3, and the ground truth is in Listing 4.9. The keywords would be extracted from `?l2`, and `?v1`.

```
1  {
2      BIND (:WAN AS ?s1) #sets Nigeria's URI
3      ?s1 a :Country.
4  }
```

Listing 4.7: Answer graph patter for keywords {"*nigeria*", "*gpd*"}

```
1  SELECT ?l1 ?l2 ?v1
2  WHERE {
3      {
4          BIND (:WAN AS ?s1) # sets Nigeria's URI
5          ?s1 a :Country.
6      }
7      :Country rdfs:label ?l1.
8
9      {SELECT ?s1 (group_concat(DISTINCT ?l) AS ?l2)
10     WHERE {?s1 ?p ?o. ?p rdfs:label ?l.}
11     GROUP BY ?s1}
12
13     {SELECT ?s1 (group_concat(?o) AS ?v1)
14     WHERE {?s1 ?p ?o FILTER isLiteral(?o)}
15     GROUP BY ?s1}
16 }
```

Listing 4.8: Query for retrieving property values for keywords {"*nigeria*", "*gpd*"}

?l1	?l2	?v1
“Country”	“... gdp...”	“Nigeria”

Table 4.3: A fragment of the binding sets for the query in Listing 4.8.

```
1  INSERT {
2      GRAPH ?g {
3          ?s1 ?p1 ?v1.
4          ?s1 ?p2 ?v2.
5      }
6  }
7  WHERE {
8      BIND (UUID() AS ?g)
9      {
10         ?s1 a :Country.
11     }
12     ?s1 ?p1 ?v1.
13     ?s1 ?p2 ?v2.
14     ?p2 rdfs:label ?l2.
15
16     FILTER (REGEX(?v1,"Nigeria","i") &&
17             REGEX(?l2,"gdp","i"))
18 }
```

Listing 4.9: Query to generate the Coffman’s ground truth for keywords {“*nigeria*”, “*gpd*”}

Computing all Steiner trees for ${}^{n,d}\mathcal{CG}^{i}_{\mathcal{T}}$ can be computationally expensive. Besides limiting n and d, we then defined the following heuristics to filter out unwanted patterns: 1) dismiss Steiner trees with more than ρ_1 nodes; 2) dismiss Steiner trees that do not include the inducer’s classes; 3) dismiss Steiner trees with a max distance between two nodes longer than ρ_2; and 4) dismiss Steiner trees with more than ρ_3 mirrored sets.

The first heuristic expresses the intuition that small solutions are preferable to complex ones. The complexity is estimated by the number of nodes since minimal Steiner trees, by definition, contain the smallest set of edges possible for a given set of terminal nodes. The second heuristic expresses the intuition that inducers guide the query generation process. The third heuristic expresses the intuition that users do not easily understand distant relationships, as argued by Nunes et al. [50]. Finally, the last heuristic allows one to control the answer patterns. For example, solutions with two countries and two organizations in an answer graph pattern may seem less satisfactory than

the graph pattern in Figure 4.3(a). The third parameter (ρ_3) then limits the number of mirrored sets in the patterns.

4.2 Computing Natural Language Queries

Recent research on textual search over RDF datasets has shown important results using natural language queries. In this section, we propose a method to synthesize natural language queries (NLQ) to help create benchmarks for evaluating Natural Language Interfaces to Databases.

The method consists of: (i) choosing a set $\mathcal{I}$ of relevant entities, called inducers, as starting points of the query generation; (ii) computing the inducers' neighborhood N, which are subgraphs centered in the node $i \in I$; (iii) computing distinct graph patterns connecting entities to i; (iv) extracting keywords from different instantiations of graph patterns; (iv) creating a NLQ based on template; and (v) improving the verbalization of predicate labels, including labels of reified relationships.

Steps (iv) and (v) aim at making the benchmarks useful for RDF-TS systems based on natural language sentences. One computes answers for a natural language question by extracting its keywords, as Web Search Engines do, and computing answers in the same way as for keyword queries. Therefore, natural language questions and keyword queries are two distinct ways of creating benchmark queries.

The process is based on the same mirroring graphs described in the previous section. For example, for the reference answer of the 43rd query in Coffman's benchmark, $\mathcal{K}_{Coff\text{-}Q43} = \{\text{“}mauritius''\text{, “}india''\}$, and the mirroring graph in Figure 4.3(a), one wants to generate a natural language query as follows:

> “What are the organizations of which India and Mauritius Islands are both members?”.

Natural language queries will then be processed much in the same way as keyword queries to compute their valid answers, as shown in Section 4.3.

4.2.1 Creating Natural Language Questions Based on Templates

The process of generating natural language queries in this thesis is inspired by Bordes et al. [1], that proposed question patterns to generate natural language questions from RDF triples. For example, given a triple (s, r, e), the question pattern $(?, r, e)$ generate questions such as “Who r e?”

and "What r e?", where r must be replaced by its label and e must be replaced by its label when it is not a literal. The label can be the value of the predicate *rdfs:label* or any other predicate indicating spoken descriptions for r and e. We call the element with the symbol "?" in the question pattern as the *target element of the question pattern* {*? r e.*}. Table 4.4 shows all question patterns proposed by Bordes et al. [1].

KB Triple	Question Pattern	KB Triple	Question Pattern
(?, r, e)	*who r e ?*	(?, r, e)	*what is e's r ?*
(?, r, e)	*what r e ?*	(e, r, ?)	*who is r by e ?*
(e, r, ?)	*who does e r ?*	(e, r-in, ?)	*when did e r ?*
(e, r, ?)	*what does e r ?*	(e, r-on, ?)	*when did e r ?*
(?, r, e)	*what is the r of e ?*	(e, r-in, ?)	*when was e r ?*
(?, r, e)	*who is the r of e ?*	(e, r-on, ?)	*when was e r ?*
(e, r, ?)	*what is r by e ?*	(e, r-in, ?)	*where was e r ?*
(?, r, e)	*who is e's r ?*	(e, r-in, ?)	*where did e r ?*

Figure 4.4: All question patterns created by Bordes et al. [1].

We extended such a method to generate more complex questions, involving graph patterns with more than one triple, such as "What are the organizations of which India and Mauritius Islands are both members and have their headquarters in Italy?". This question can be an intuitive derivation of the mirroring graph ${}^{2,2}\mathcal{CG}_{\mathcal{T}}^{India}$ in Figure 4.5(a) when the target elements are the Organization (?1) and the City (?2) in Figure 4.5(b). Since we admit multiple target nodes, we number each node to distinguish them.

The first step is querying the dataset for instantiating the graph pattern in Figure 4.5(a) with resource labels, using the query in Listing 4.11, as explained in Section 4.1. Table 4.4 shows a fragment of the result set.

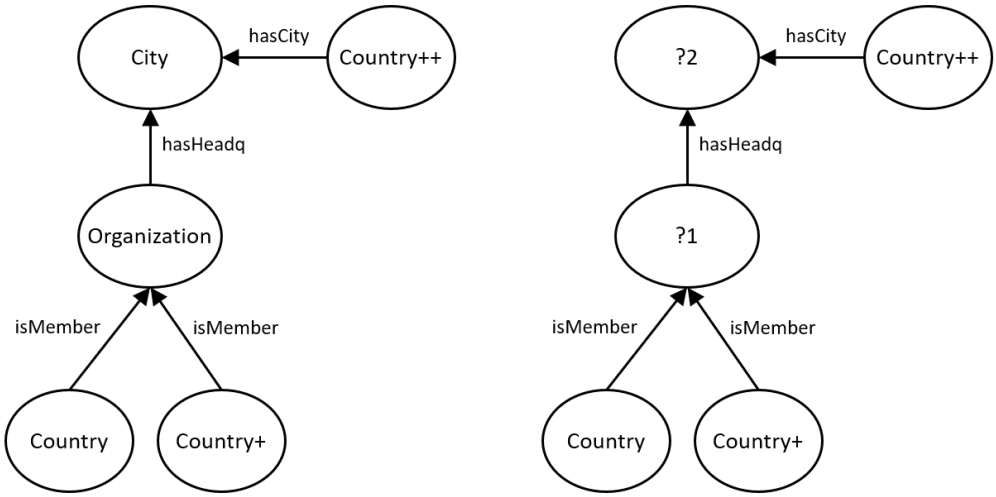
The second step is to build a generic natural language question based on the template in Listing 4.10. We express templates according to the Transact-SQL Reference syntax conventions[2] for convenience.

```
{What is target_node_id | What are target_node_id [, ...] and
   target_node_id} such that {triple_pattern_translation |
   triple_pattern_translation [, ...] and
   triple_pattern_translation}

where
    target_node_id is one of the {?1, ?2, ...} and
    triple_pattern_translation is a sentence decribing a triple
        pattern.
```

Listing 4.10: Generic template.

[2] https://documentation.help/tsqlref/ts_syntaxc_9kvn.htm



4.5(a): The minimum Steiner tree. 4.5(b): The target elements indication.

Figure 4.5: Example of a question pattern from a minimum Steiner tree derived from the mirroring graph ${}^{2,2}\mathcal{CG}_{\mathcal{T}}^{India}$.

For example, given the mirroring graph in Figure 4.5(a), the correspondent generic question would be

> "*What are ?1 and ?2* such that *India is member ?1*, *Mauritius Islands is member ?1*, *?1 has headquarter ?2* and *Italy has city ?2?*".

It has two targets, *?1* and *?2*, and four `triple_pattern_translations`. Note that the `triple_pattern_translations` are built with the resource and predicate labels in Table 4.4 and the core graph pattern in Lines 10–13 of Listing 4.11.

```
1  SELECT ?l1 ?l2 ?l3 ?l4 ?l5 ?l6 ?v1 ?v2 ?v3 ?v4 ?v5
2  WHERE {
3      {
4          BIND (:IND as ?s1) # sets India's URI
5          ?s1 a :Country.
6          ?s2 a :Country.
7          ?s3 a :Country.
8          ?s4 a :Organization.
9          ?s5 a :City.
10         ?s1 :isMember ?s4.
11         ?s2 :isMember ?s4.
12         ?s4 :hasHeadq ?s5.
13         ?s3 :hasCity  ?s5.
14         FILTER (?s1!=?s2 && ?s1!=?s3 && ?s1!=?s4 && ?s1!=?s5
               && ?s2!=?s3 && ?s2!=?s4 && ?s2!=?s5 && ?s3!=?s4
               && ?s3!=?s5 && ?s4!=?s5)
15     }
```

```
16      :Country rdfs:label ?l1.
17      :Organization rdfs:label ?l2.
18      :City rdfs:label ?l3.
19      :isMember rdfs:label ?l4.
20      :hasHeadq rdfs:label ?l5.
21      :hasCity rdfs:label ?l6.
22      ?s1 rdfs:label ?v1.
23      ?s2 rdfs:label ?v2.
24      ?s3 rdfs:label ?v3.
25      ?s4 rdfs:label ?v4.
26      ?s5 rdfs:label ?v5.
27  }
```

Listing 4.11: Query for retrieving labels class, predicates and entities

?l1	?l2	?l3	?l4	?l5	?l6	?v1	?v2	?v3	?v4	?v5
Country	Organization	City	is member	headquarters	located in	Mauritius Island	India	Italy	IFAD	Roma

Table 4.4: Fragment of the binding sets derived from Figure 4.5

As one can see, target node identifications and the existing dataset predicate labels may not produce good verbalization. The same sentence could be improved as follows.

> "*What are the entities E1 and E2* such that *India is a member of E1*, *Mauritius Islands is a member of E1*, *E1 has headquarters in E2* and *Italy has the city E2?"*

or

> "*What are the Organization O and the City C* such that *India is a member of O*, *Mauritius Islands is a member of O*, *O has headquarters in C* and *Italy has the city C?"*

Improvements can optionally use the classes of the target nodes and add prepositions, articles, and other complements for the predicate labels. In this case, the predicate improvements are subtle, but more complex replacements may occur, such as "is in the mountain of" instead of "in mountains", "river flows through" instead of "flows through", and replacements of reified $N \times N$ relationships. Sections 4.2.2 and 4.2.3 address these issues.

The third and final step is a grammatical optimization of the question using the following rules.

– $(?s_1, r, e), ..., (?s_n, r, e)$ replace with: $?s_1$, ..., $?s_n$ r e

– $(e, r, ?s_1), ..., (e, r, ?s_n)$ replace with: e r $?s_1$, ..., and $?s_n$

For example,

> "*What are the Organization O and the City C* such that *India and Mauritius Islands are members of O*, *O has headquarters in C* and *Italy has the city C?"*

Generic questions assume that all predicate labels induce sentences with good readability. However, this is not always the case. For example, the predicate label "is member" should be "is a member of" , and "has headq" should be "has headquarters in". The next section shows how to improve the predicate labels to generate better natural language questions.

4.2.2 Improving the Verbalization of Predicate Labels

This section describes, with the help of examples, how to improve the verbalization of the predicate label for a given RDF triple t.

Predicate labels do not always induce good readability. One can verbalize the triple (*</countries/MS>*, *:isMember*, *</organizations/IFAD/>*) as "Mauritius Islands is member International Fund for Agricultural Development" using the URI's labels. We call it the *triple's original verbalization*. However, a sentence like "Mauritius Islands is a member of the International Fund for Agricultural Development", called the *triple's improved verbalization*, would be preferable. Towards this end, we propose the technique presented in what follows.

The first step consists in creating vector representations of the words in the predicate labels for a triple t. We use the Bidirectional Encoder Representations from Transformers (BERT) [40], as in Section 3.3, to encode the triple's original verbalization and to get the word embeddings.

The next step is to select a sentence from a corpus that has a similar expression for the predicate. In this case, we have found the sentence "Algeria is a member of the African Union, the Arab League, OPEC, the United Nations and is the founding member of the Maghreb Union".

In detail, let C be a sentence corpus. Then, BERT encodes each sentence $s \in C$, and computes their word embeddings. We used the sentences extracted from the DBpedia abstracts.

Let $\vec{w}$ be the word embedding of the predicate label's word w of triple t, $\vec{w'}$ be the word embedding of the word w' of a sentence $s \in C$, and $cos(\vec{v_1}, \vec{v_2})$ be the cosine similarity function between two vectors $\vec{v_1}$ and $\vec{v_2}$. We rank the

corpus sentences with $max_{\overrightarrow{w'_1},...,\overrightarrow{w'_n}}(H(cos(\overrightarrow{w_1}, \overrightarrow{w'_1}), ..., cos(\overrightarrow{w_n}, \overrightarrow{w'_n})))$, where H is the harmonic mean of the cosine similarities and n is the number of words in the predicate label of t. The top-ranked sentences should contain a convenient verbalization of the predicate of t, if the words are not distant from each adjacent word for more than three words.

The third and final step is to extract words from the sentence to replace the original predicate label. One starts by computing part-of-speech and the syntactic dependency tree of words in the top ranked sentence s. We used spaCy[3] for this task. Our running example selected the sentence "Algeria is a member of the African Union, the Arab League, OPEC, the United Nations and is the founding member of the Maghreb Union", whose fragment of the part-of-speech and dependency tree is shown in Figure 4.6.

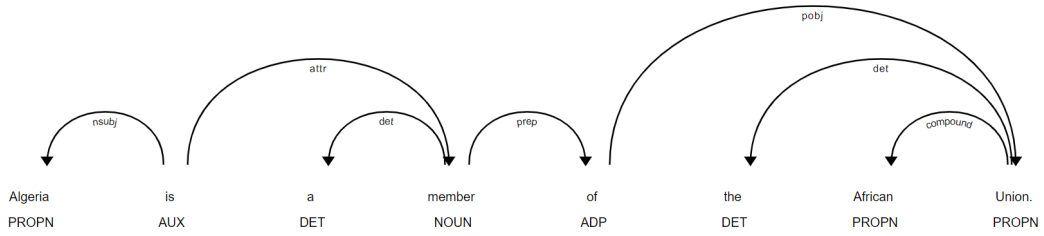


Figure 4.6: spaCy Dependency Parse Tree and Part-of-Speech example.

The grammatical information is used to define some rules to extract the new predicate label, as follows.

1. Select the words w'_i that match words w_j in the predicate label using cosine similarity of embeddings (for example "is" and "member")

2. Select the words w''_k linked to w'_i by the syntactic tree at a distance 1 (for example "Algeria", "a", and "of")

3. Filter in the words linked with the following relationships: "det" (determiner) and "prep" (preposition) and (such as "a" and "of")

The replacement label is then "is a member of".

Another example is the verbalization of triple (*</mountains/Maipo>*, *:locatedIn*, *</countries/RCH/>*) as "Maipo located in Chile" using the URI's labels. Running all the steps of our method, the select sentence is "The city is located in the province of North Holland.", whose part-of-speech and dependency tree fragment is in Figure 4.7.

This grammatical information is used to define some rules to extract the new predicate label, as follows.

[3] `https://spacy.io/`

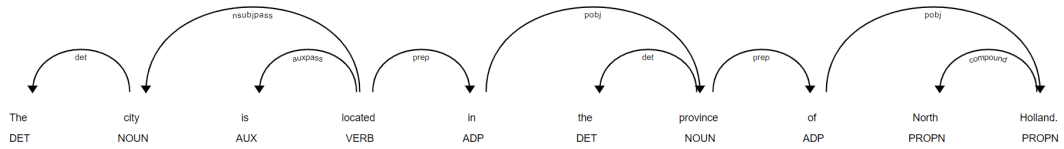


Figure 4.7: Second example of spaCy Dependency Parse Tree and Part-of-Speech example.

1. Select the words w'_i that match words w_j in the predicate label using cosine similarity of embeddings (for example, “is” and “located”);

2. Select the words w''_k linked to w'_i by the syntactic tree at a distance 1 (for example, “city” and “in”);

3. Filter in the words linked with the following relationships: "prep" (preposition) (such as “in”).

In our running example, the replacement label is then “is located in”.

We applied this method for all predicate labels in the Mondial dataset, achieving 100% precision for all of them. Appendix B.1 presents the results.

4.2.3 Verbalization of reified relationships

Reified relationships, describe in 3.4.1, should not be verbalized as defined in Section 4.2.2. Instead, the triples whose subjects are instances of the reification class should be part of the same sentence such as, for example, “Hindi is spoken by India” or “India speaks Hindi”. This section will address the issues of identifying such cases and properly verbalizing these relationships.

Intuitively, all instances of a reification class would share the same object properties. The first step is then to identify the RDF classes that follow this intuition as candidate classes that encode reified relationships. The second step is determining which object properties play the role of reification object properties.

Given a property pair p, such as “language info” and “on language”, and a candidate class C, such as “Spoken by”, one computes the average cosine similarity between the vectorial representations of the class label and each property label in p. Vectorial representations are computed with BERT using sentences derived from triples but using the label of the candidate class instead of the label of the relationship instances. For example, given the relationship instance $\_{:}s_1$ in Figure 3.5, the triple ($\_{:}s_1$, *:languageInfo-*, *</countries/IND/>*) would be written as “Spoken by language info India” and the triple ($\_{:}s_1$, *:onLanguage*, *</languages/Hindi/>*)

would be written as "Spoken by on language Hindi". The process repeats for a sample of relationship instances, and one computes the average similarity in the sample. The property pair with the highest average similarity above a given threshold is the property pair embodying the binary relationship.

Given the detected pair of object properties embodying the reified binary relationship, the final step is determining the relationship verbalization. One can read it both ways, i.e., "Hindi spoken by India" or "India spoken by Hindi". To choose the best verbalization, one computes embeddings for each sentence and proceeds with the same process as the in the previous section. The corpus sentence with the highest score will indicate the correct verbalization. In the running example, the best sentence is "English is spoken by American people", which matches the sentence "Hindi spoken by India" and therefore would indicate that the correct reading direction, from Language to Country, and the replacement label "is spoken by".

We applied this method to detect reified relationships over the Mondial dataset and found all reified relationships. That is, the recall was 100% and there were no false positives, i.e., precision was also 100%. Appendix C.1 presents in more detail such results over verbalization of reified relationships.

4.3 Computing Solution Generators

This section addresses the computation of solution generators for keyword queries through four heuristics to circumvent the complexity of the problem.

Let $\mathcal{T}$ be an RDF dataset. Recall from Section 3.4.1 that the *entity graph* induced by a subset $\mathcal{T}' \subseteq \mathcal{T}$ is the subgraph $\mathcal{EG}_{\mathcal{T}'}$ of $\mathcal{G}_{\mathcal{T}'}$ obtained by dropping all literal nodes from the graph $\mathcal{G}_{\mathcal{T}'-TBox}$. Also recall that the *neighborhood* of distance d of an entity e in $\mathcal{T}$, denoted ${}^{d}\mathcal{N}^{e}_{\mathcal{T}}$, is the set of all nodes visited in a breadth-first walk of distance d starting from e in $\mathcal{EG}_{\mathcal{T}}$.

The next step is to compute possible answers for $\mathcal{K}$. The method computes the complete set of matching resources for $\mathcal{K}$, called the *set of seeds* for $\mathcal{K}$, denoted $\mathcal{S}_{\mathcal{K}}$. By the previous definitions, the set of matching resources of an answer $\mathcal{A}$ for $\mathcal{K}$ must be a subset of $\mathcal{S}_{\mathcal{K}}$. Preferably, answers should contain subsets of $\mathcal{S}_{\mathcal{K}}$ that match all keywords in $\mathcal{K}$.

Consider the following example. Let *"character of Meryl Streep in the movie Out of Africa"* be an information need over the dataset $\mathcal{T}$ of Figure 3.6(a). This information need can be translated to a keyword query $\mathcal{MS}$ = {"character", "meryl", "streep", "movie", "out", "africa"}. The set of seeds for $\mathcal{MS}$ is $\mathcal{S}_{\mathcal{MS}} = \{A, B, D, F, H\}$. Figures 3.6(b)–3.6(d) show the graphs induced

by three possible answers for $\mathcal{MS}$ containing subsets of $\mathcal{S}_{\mathcal{MS}}$. Intuitively, answer $\mathcal{A}_1$ (Figure 3.6(b)) is better than $\mathcal{A}_2$ (Figure 3.6(c)) and $\mathcal{A}_3$ (Figure 3.6(d)), because $\mathcal{A}_1$ addresses what seems to be the query intention, which is to find the character played by the actress in the movie. Also, $\mathcal{A}_1$ matches 6 of the 6 keywords in $\mathcal{MS}$, while $\mathcal{A}_2$ and $\mathcal{A}_3$ match only 5 and 3 keywords, respectively. Note that, in $\mathcal{A}_1$ and $\mathcal{A}_2$, but not in $\mathcal{A}_3$, keywords are not matched by more than one literal. Also, note that the keywords *"movie"* and *"character"* are associated with elements of the schema, nodes B and H.

This example illustrates two important characteristics of keyword queries and correct answers: *keyword queries name existing resources, and answers correlate these resources*. In this example, "Meryl Streep", "Out of Africa", "character", and "movie" are informal resource identifiers that represent an actress, a movie, the `Character` class, and the `Movie` class, respectively.

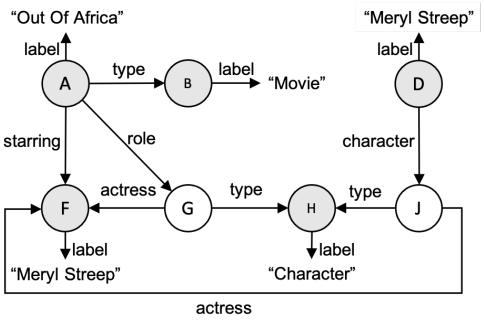
Let $\mathcal{R} \subseteq \mathcal{S}_{\mathcal{K}}$ be a subset of seeds. Assume that all seeds in $\mathcal{R}$ belong to the same connected component of $\mathcal{RG}_{\mathcal{T}}$, the resources graph induced by $\mathcal{T}$. One can compute all possible answers whose matching resources are subsets of $\mathcal{R}$ by computing all acyclic paths in $\mathcal{RG}_{\mathcal{T}}$ between pairs of distinct nodes in $\mathcal{R}$, combining them in all distinct ways to construct connected graphs containing all nodes in $\mathcal{R}$, and adding the matching subgraphs for the seeds $r \in \mathcal{R}$.

Consider the set of seeds $\mathcal{R}'_{\mathcal{MS}} = \{A, B, F, H\}$, for example. If one selects the paths $(B \leftarrow A \rightarrow G \rightarrow F)$ and $(G \rightarrow H)$ and adds the matching subgraphs $(A \rightarrow \text{“}Out\ of\ Africa\text{”})$, $(B \rightarrow \text{“}Movie\text{”})$, $(F \rightarrow \text{“}Meryl\ Streep\text{”})$, and $(H \rightarrow \text{“}Character\text{”})$, one will obtain the answer in Figure 3.6(b). If one selects the path $(B \leftarrow A \rightarrow F \leftarrow G \rightarrow H)$ and adds the same matching subgraphs, one will obtain another valid answer (not shown in the figure).

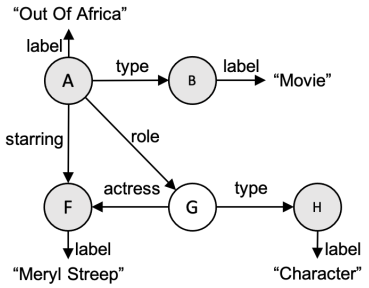
Consider now the set of seeds $\mathcal{R}''_{\mathcal{MS}} = \{A, B, F\}$. If one selects the path $(B \leftarrow A \rightarrow F)$ and adds the matching subgraphs $(A \rightarrow \text{“}Out\ of\ Africa\text{”})$, $(B \rightarrow \text{“}Movie\text{”})$, and $(F \rightarrow \text{“}Meryl\ Streep\text{”})$, one will obtain the answer in Figure 3.6(c). If one selects a different path $(B \leftarrow A \rightarrow G \rightarrow F)$, one will obtain yet another answer. In general, distinct path combinations may lead to distinct valid answers for the same set of seeds.

More precisely, let $\mathcal{K}$ again be a keyword query and $\mathcal{R} \subseteq \mathcal{S}_{\mathcal{K}}$. Assume that all seeds in $\mathcal{R}$ belong to the same connected component of $\mathcal{RG}_{\mathcal{T}}$. A set of triples $\mathcal{SG} \subseteq \mathcal{T}$ is a *solution generator* for $\mathcal{R}$ iff $\mathcal{SG}$ can be partitioned into two sets, $\mathcal{SG}'$ and $\mathcal{SG}''$, such that: (i) $\mathcal{SG}'$ is the set of all matching triples of the resources in $\mathcal{R}$; (ii) $\mathcal{SG}''$ is the set of all triples that occur in paths in $\mathcal{RG}_{\mathcal{T}}$ that begin and end on the seeds in $\mathcal{R}$. We say that $\mathcal{SG}$ *expresses* an answer $\mathcal{A}$ iff $\mathcal{A} \subseteq \mathcal{SG}$ and the set of matching resources of $\mathcal{A}$ is $\mathcal{R}$.

Figure 4.8 shows the solution generators for $\mathcal{R}_{\mathcal{MS}} = \{A, B, D, F, H\}$ and



4.8(a): The solution generator for $\mathcal{S}_\mathcal{K} = \{A, B, D, F, H\}$ from the dataset in Fig. 3.6(a).



4.8(b): The solution generator for $\mathcal{S}_\mathcal{K} = \{A, B, F, H\}$ from the dataset in Fig. 3.6(a).

Figure 4.8: Examples of solution generators.

$\mathcal{R}'_{\mathcal{MS}} = \{A, B, F, H\}$. Note that, even though both $\mathcal{R}_{\mathcal{MS}}$ and $\mathcal{R}'_{\mathcal{MS}}$ cover all keywords, they are distinct and express distinct sets of answers.

A *synthetic benchmark* is then a triple $s = (\mathcal{T}, Q_s, A_s)$ such that $\mathcal{T}$ is an RDF dataset, Q_s is a list of keyword queries, and A_s contains, for each query in Q_s, a list of solution generators. Chapter 5 illustrates this concept.

However, computing all solution generators can be expensive, depending on the cardinality of $\mathcal{S}_\mathcal{K}$ and the number of paths between the seeds. We then propose to compute solution generators that capture only the most relevant answers. Fortunately, unlike traditional Information Retrieval (IR) systems, one can exploit peculiarities of the RDF KwS-Problem to define optimization heuristics that reduce the computational cost and help rank solution generators.

The differences between traditional IR systems and RDF-KwS systems stem from the fact that keywords that co-occur in a document may not convey the idea of keyword correlation. By contrast, an RDF subgraph connecting resources linked to these keywords is much more likely to be related to the intended meaning of the keyword query because resources are not connected by chance in an RDF graph, as it may be the case in a text document.

By assuming such differences, one can define some characteristics of good answers: the keywords must be connected as closely as possible, and answers must contain as many keywords as possible. One can define other features based on the number of resources in answers and the co-occurrence of keywords among the literals. These features may guide the automatic computation of answers by helping prune the search space.

We use four heuristics to work around the complexity of the problem of computing solution generators, guided by five questions: (1) Are all seeds relevant?; (2) Are all paths between seeds relevant?; (3) Should we prefer

answers that match more literals or answers that match fewer literals?; (4) Should we prefer answers in which literals in the keyword query occur in many seeds, or answers in which literals occur in only one seed?; (5) Should we prefer answers with many seeds or answers with fewer seeds?

The first heuristic refers to the selection of the most relevant seeds. Assume that Lucene for Apache Jena Fuseki is the text search engine over RDF adopted. The *Lucene score* is a TF-IDF-based score that captures the relevance of property values for the keywords. The complete set of seeds of a keyword query can be obtained from the Lucene inverted index

However, the set of seeds can be large. One could then limit this set to the top-k resources by the Lucene score, but the resource labeled "Meryl Streep" would be the 7th entry in the ranked list of seeds, and the resource labeled "Out of Africa" would be the 30th entry. If one took the top 30 resources in the ranking, the two seeds would be selected, but many less relevant seeds would also be selected. Therefore, we define a more elaborated entity score as follows.

Let $\mathcal{K}$ be keyword query and $\mathcal{S}_\mathcal{K}$ be the set of seeds of $\mathcal{K}$. We define the *entity score* of a seed $s \in \mathcal{S}_\mathcal{K}$ for $\mathcal{K}$ as

$$es(s, \mathcal{K}) = \frac{1}{2}(max_{v_j}(lucene(s, v_j, \mathcal{K})) + infoRank(s)) \tag{4-1}$$

where v_j is a string value such that there is a triple $(s, p, v_j) \in \mathcal{T}$. The infoRank score [2] reflects the relevance of s to users. The entity score ranges in $[0, 1]$, since we used normalized versions of $lucene(s, v_j, \mathcal{K})$ and $infoRank(s)$. If a text search engine other than Lucene is adopted or a resource relevance measure other than infoRank is used, Eq. 4-1 should be adjusted accordingly.

By ranking the set of seeds of $\mathcal{K}$ according to the entity score, the resource labeled "Meryl Streep" would appear in the 1st position, and that labeled with "Out of Africa" would appear in the 18th position.

The *first heuristic* is then to refine the set of seeds $\mathcal{S}_\mathcal{K}$ to be the set $\Sigma_\mathcal{K}$ of the top-σ_2 elements of $\{s \in \mathcal{S}_\mathcal{K} | es(s, \mathcal{K}) \geq \sigma_1\}$ ordered in decreasing order of entity score, where σ_1 and σ_2 are thresholds empirically defined to optimize computing resources and match user's preferences.

The threshold σ_1 defines a minimum seed entity score to speed up the Lucene engine for practical reasons, and σ_2 effectively limits the number of selected seeds. By defining $\sigma_1 = 0$ and $\sigma_2 = \infty$, one would select the full set of seeds. But, by defining $\sigma_1 > 0$ and $\sigma_2 < \infty$ one can restrict the cardinality of $2^{\Sigma_\mathcal{K}}$ and, consequently, the total number of paths to compute.

However, $\Sigma_\mathcal{K}$ may not cover all keywords in $\mathcal{K}$. We then compute another set $\mathcal{S}_{\mathcal{K}_i}$ of matching resources, where $\mathcal{K}_i$ is the subset of $\mathcal{K}$ not matched by resources in $\Sigma_\mathcal{K}$, and repeat this procedure to extend $\Sigma_\mathcal{K}$ until no other

keyword can be matched or $\mathcal{K}_i = \{\}$. More precisely, for any $\mathcal{K}_i \subseteq \mathcal{K}$, let $\mathcal{S}_{\mathcal{K}_i}$ be the set of seeds in $\mathcal{S}_{\mathcal{K}}$ that match keywords in $\mathcal{K}_i$. Let $\mu[\sigma_1](\mathcal{K}_i, \mathcal{S}_{\mathcal{K}_i}) = \{s \in \mathcal{S}_{\mathcal{K}_i} | es(s, \mathcal{K}_i) \geq \sigma_1\}$, and $\tau[\sigma_1, \sigma_2](\mathcal{K}_i, \mathcal{S}_{\mathcal{K}_i})$ be the top-σ_2 elements of $\mu[\sigma_1](\mathcal{K}_i, \mathcal{S}_{\mathcal{K}_i})$, ordered by the entity score of the seeds.

Let $(\mathcal{K}_0, ..., \mathcal{K}_N)$ be the longest sequence such that $\mathcal{K}_0 = \mathcal{K}$ and, for each $i \in [1, N]$, $\mathcal{K}_i$ is the set of keywords in $\mathcal{K}$ not matched by seeds in $\tau[\sigma_1, \sigma_2](\mathcal{K}_{i-1}, \mathcal{S}_{\mathcal{K}_{i-1}})$ and $\mathcal{K}_i \neq \emptyset$ and $\mathcal{S}_{\mathcal{K}_i} \neq \emptyset$. Then, $\Sigma_{\mathcal{K}}$ is defined as

$$\Sigma_{\mathcal{K}} = \bigcup_{i=0}^{N} \tau[\sigma_1, \sigma_2](\mathcal{K}_i, \mathcal{S}_{\mathcal{K}_i}) \tag{4-2}$$

The *second heuristic* refers to the selection of sets in $2^{\Sigma_{\mathcal{K}}}$ for which one would compute the solution generators. It is done by scoring each $\mathcal{R} \in 2^{\mathcal{S}_{\mathcal{K}}}$ and selecting the top-ranked ones based on four principles. First, the set of matching keywords of $\mathcal{R}$ is the union of the set of keywords matched by each resource in $\mathcal{R}$; the sets $\mathcal{R}$ with the most significant number of keyword matchings are preferable and potentially would generate better answers. Second, the keywords should preferably match just a few seeds of an answer because keywords identify entities. We assume that answers where keywords do not match more than one resource, such as those in Figures 3.6.3.6(b) and 3.6.3.6(c), are more relevant.

Nevertheless, as detailed later in this chapter, this constraint can be relaxed to allow answers such as that in Figure 3.6(d). Third, smaller answers, in terms of the number of seeds, are preferable over larger ones since they are easier to understand. Lastly, small sets are preferable, but it is also necessary that their resources are the most relevant to users. Based on these principles, we define the following scores.

Let $\mathcal{E}$ be a set of resources. The *coverage score* of $\mathcal{E}$, denoted $cs(\mathcal{E}, \mathcal{K})$, measures the number of keywords in $\mathcal{K}$ matched by resources in $\mathcal{E}$; $cs(\mathcal{E}, \mathcal{K}) = 1$, if all keywords are matched, and $cs(\mathcal{E}, \mathcal{K}) = 0$, if no keyword is matched:

$$cs(\mathcal{E}, \mathcal{K}) = \frac{\sum_{k_i \in \mathcal{K}} occur(\mathcal{E}, k_i)}{|\mathcal{K}|} \tag{4-3}$$

where $occur(\mathcal{E}, k_i) = 1$, if some resource in $\mathcal{E}$ matches k_i, and $occur(\mathcal{E}, k_i) = 0$, otherwise.

The *co-occurrence score* of $\mathcal{E}$, denoted $os(\mathcal{E}, \mathcal{K})$, measures the repetition degree of keywords in $\mathcal{K}$ among resources in $\mathcal{E}$; $os(\mathcal{E}, \mathcal{K}) = 1$, if each keyword is matched by only one resource, and $os(\mathcal{E}) = 0$, if all keywords are matched by all resources:

$$os(\mathcal{E}, \mathcal{K}) = \frac{1 - \frac{f(\mathcal{E}, \mathcal{K})}{|\mathcal{E}|}}{1 - \frac{1}{|\mathcal{E}|}} \tag{4-4}$$

where $f(\mathcal{E}, \mathcal{K})$ is the average number of resources in $\mathcal{E}$ that match keywords in $\mathcal{K}$. Keywords not covered in $\mathcal{E}$ are not taken into account; $os(\mathcal{E}, \mathcal{K})$ is assumed to be 1, if the denominator is 0.

Let $\mathcal{C}$ be the collection of sets of resources considered. The *size score* of $\mathcal{E}$ w.r.t. $\mathcal{C}$, denoted $ss(\mathcal{E})$, measures the relative size of $\mathcal{E}$; $ss(\mathcal{E}) = 1$, if $\mathcal{E}$ is one of the smallest sets, and $ss(\mathcal{E}) = 0$, if $\mathcal{E}$ is one of the largest sets:

$$ss(\mathcal{E}) = \frac{\mathcal{N} - |\mathcal{E}|}{\mathcal{N} - 1} \tag{4-5}$$

where $\mathcal{N}$ is the cardinality of the largest set in $\mathcal{C}$; $ss(\mathcal{E}, \mathcal{K})$ is assumed to be 1, if the denominator is 0.

The *infoRank score* of $\mathcal{E}$, denoted $is(\mathcal{E})$, is the average infoRank value [2] of the resources in $\mathcal{E}$:

$$is(\mathcal{E}) = average(\{infoRank(s_i) | s_i \in \mathcal{E}\}) \tag{4-6}$$

The second heuristic is then the refinement of the set $2^{\Sigma_\mathcal{K}}$ by choosing only those $\mathcal{R} \in 2^{\Sigma_\mathcal{K}}$ with better coverage, lower co-occurrence, fewer number of resources, and with more relevant nodes. Recall that a lower co-occurrence would favor answers such as those in Figures 3.6.3.6(b) and 3.6.3.6(c), while allowing answers such as that in Figure 3.6.3.6(d). On the other hand, no co-occurrence ($os(\mathcal{R}, \mathcal{K}) = 1$) would allow answers such as those in Figures 3.6.3.6(b) and 3.6.3.6(c) only. The refinement is expressed by defining set Π_K as follows:

$$\begin{aligned} \Pi_K = \{\mathcal{R} \in 2^{\Sigma_\mathcal{K}} | cs(\mathcal{R}, \mathcal{K}) \geq \sigma_3 \wedge os(\mathcal{R}, \mathcal{K}) \geq \\ \sigma_4 \wedge ss(\mathcal{R}) \geq \sigma_5 \wedge is(\mathcal{R}) \geq \sigma_6\} \end{aligned} \tag{4-7}$$

where σ_3, σ_4, σ_5, and σ_6 are empirically defined according to the available computing resources and user preferences. If $\sigma_3 = \sigma_4 = \sigma_5 = \sigma_6 = 0$ then $\Pi_K = 2^{\Sigma_\mathcal{K}}$ and all possible solution generators with $\Sigma_\mathcal{K}$ would be computed. The above scores can be redefined for subsets of triples $\mathcal{U} \subseteq \mathcal{T}$ by taking $\mathcal{E} = \mathcal{E}_\mathcal{U}$, where $\mathcal{E}_\mathcal{U}$ is the set of all resources in $\mathcal{U}$.

The *third heuristic* is to consider only paths between seeds with length less than or equal to a given limit L, say $L = 4$, to compute solution generators. As argued by Nunes et al. [50], paths longer than 4 would express unusual relationships, which users might misinterpret.

The *fourth heuristic* is to rank the solution generators in Π_K according to their scores, following user needs. For example, if one ranks based only on the coverage and size, one may define a Boolean function "order" between solution

generators as follows:

$$order(\mathcal{SG}_1, \mathcal{SG}_2) = \begin{cases} cs(\mathcal{SG}_1, \mathcal{K}) \geq cs(\mathcal{SG}_2, \mathcal{K}), \text{ if } C \text{ holds} \\ ss(\mathcal{SG}_1, \mathcal{K}) \geq ss(\mathcal{SG}_2), \text{ otherwise} \end{cases} \tag{4-8}$$

where C is the condition $cs(\mathcal{SG}_1, \mathcal{K}) \neq cs(\mathcal{SG}_2, \mathcal{K})$. Alternatively, one could rank solution generators by their average scores as in Eq. 4-9 to balance the losses and gains of each individual score:

$$\begin{aligned} order(\mathcal{SG}_1, \mathcal{SG}_2) = \frac{1}{4}(cs(\mathcal{SG}_2, \mathcal{K}) + \\ os(\mathcal{SG}_2, \mathcal{K}) + ss(\mathcal{SG}_2) + is(\mathcal{SG}_2)) \end{aligned} \tag{4-9}$$

Eqs. 4-2, 4-7, 4-8, and 4-9 are, in fact, flexibilization points of the method. For example, instead of using the Lucene score in Eq. 4-1, one could use a keyword count, and instead of a selection in Eq. 4-7, one could use the top of the ranking of the sets $\mathcal{R} \in 2^{\Sigma_{\mathcal{MS}}}$ with the order function defined in Eq. 4-9. The method chosen will determine the set of solution generators that will be computed. The central contribution of this work lies, then, in using the peculiarities of RDF keyword search to define a process for optimizing the computation of solution generators.

Algorithm 1 Algorithm to compute a ranked list of solution generators for a keyword query $\mathcal{K}$ over an RDF dataset $\mathcal{T}$.

Require: *a keyword query* $\mathcal{K}$ *and an RDF dataset* $\mathcal{T}$
1: $\Sigma_{\mathcal{K}} = \bigcup_{i=0}^{N} \tau[\sigma_1, \sigma_2](\mathcal{K}_i, \mathcal{S}_{\mathcal{K}_i})$
2: $\Pi_K = \{\mathcal{R} \in 2^{\mathcal{S}_{\mathcal{K}}} | cs(\mathcal{R}, \mathcal{K}) \geq \sigma_3 \wedge os(\mathcal{R}, \mathcal{K}) \geq \sigma_4 \wedge ss(\mathcal{R}) \geq \sigma_5 \wedge is(\mathcal{R}, \mathcal{K}) \geq \sigma_6\}$
3: $\mathcal{U} = \{\}$
4: **for all** $\mathcal{R} \in \Pi_K$ **do**
5: $\quad \mathcal{SG} = \{\}$
6: $\quad$ **for all** *distinct unordered pairs of nodes* $\{n_1, n_2\}$ *such that* $n_1, n_2 \in \Sigma_{\mathcal{K}}$ **do**
7: $\quad\quad \mathcal{SG} = \mathcal{SG} \cup \{(s,p,o) \in \mathcal{T} | (s,p,o)$ *is in a path with length* ≤ 4 *between* n_1 *and* $n_2\}$
8: $\quad$ **end for**
9: $\quad$ **if** $\mathcal{G}'_{\mathcal{SG}}$ *is connected* **then**
10: $\quad\quad \mathcal{U} = \mathcal{U} \cup \{\mathcal{SG}\}$
11: $\quad$ **end if**
12: **end for**
13: *Create* $\mathcal{U}'$ *by ordering* $\mathcal{U}$ *using the score function in Eq. 4-8*
14: **return** $\mathcal{U}'$

Algorithm 1 embeds the proposed heuristics to reduce the cost of com-

puting solution generators by disregarding the less relevant ones. It takes as input a keyword query $\mathcal{K}$ and an RDF dataset $\mathcal{T}$, and outputs a ranked list of solution generators according to the score function in Eq. 4-8. Lines 1 and 2 prepare the sets of nodes that will guide the computation of the solution generators according to Eqs. 4-2 and 4-7. Lines 4–12 compute solution generators for each set of seeds in Π_K. In lines 9–11, if the set of triples $\mathcal{SG}$ computed for a set of seeds $\mathcal{R} \in \Pi_K$ does not induce a connected graph $\mathcal{G}'_{\mathcal{SG}}$ (connectivity graph of $\mathcal{SG}$), then $\mathcal{SG}$ is discarded because, for each connected component $\mathcal{C}_i$ of $\mathcal{G}'_{\mathcal{SG}}$, there is a strict subset $\mathcal{R}_j$ of $\mathcal{R}$ such that the set of triples computed for $\mathcal{R}_j$ induces $\mathcal{C}_i$.

5
Evaluation of the Benchmark Generation Method

This chapter addresses the critical question of evaluating the proposed benchmark generation method. The evaluation concentrates on assessing the quality of the solution generators.

5.1
Evaluation Strategy

The evaluation strategy goes as follows. Let $b = (\mathcal{D}, Q_b, A_b)$ be a *baseline benchmark*, where $\mathcal{D}$ is an RDF dataset, Q_b is a set of keyword queries, and A_b defines the correct answers for the queries in Q_b. The strategy is to construct a synthetic benchmark $s = (\mathcal{D}, Q_s, A_s)$ for the same dataset $\mathcal{D}$, using the proposed method, where $Q_b \cap Q_s \neq \emptyset$ and A_s contains, for each keyword query in Q_s, a list of solution generators over $\mathcal{D}$, synthesized using an implementation of Algorithm 1. Then, for each keyword query in $Q_b \cap Q_s$, we compare the answers in A_b with the solution generators in A_s, as explained in what follows; this is the critical point of the evaluation. Note that we have to guarantee that $Q_b \cap Q_s \neq \emptyset$ as otherwise, the benchmark comparison would have a vacuous effect. Rather than using the keyword query generation method discussion in Section 4.1, we manually selected keyword queries from the baseline benchmarks to include in the synthetic benchmarks, as also discussed in what follows.

As baselines, we adopted a benchmark for RDF-KwS based on Coffman's [3] and Dosso's [4] benchmarks. Coffman's benchmark was created to evaluate keyword search systems over relational databases and is based on data and schemas of relational samples of IMDb, Mondial, and Wikipedia. Each database has 50 keyword queries and their correct answers. Dosso and Silvello [4] used three real RDF datasets, LinkedMDB, IMDb, and DBpedia, and two synthetic RDF datasets, The Lehigh University Benchmark – LUBM, and The Berlin SPARQL Benchmark – BSBM.

As for the datasets, we chose triplifications of relational versions of the full Mondial and IMDb datasets, and not just samples as in Coffman's benchmark, and the RDF datasets BSBM, LUBM, and DBpedia from Dosso's benchmark. For each such RDF dataset, we computed the infoRank scores [2].

We selected the keyword queries of the synthetic benchmarks as follows. Quite a few keyword queries in Coffman's benchmark are *simple queries* in that their expected answers are single entities. Since these queries do not explore the complexity of the graph structure of the RDF datasets, we disregarded them for IMDb and Mondial. We used the keyword queries for BSBM, LUBM, and DBpedia as defined in Dosso's benchmark. In total, we used 35 keyword queries for IMDb and 12 for Mondial from Coffman's benchmark, and 50 keyword queries for DBpedia, 14 for LUBM, and 13 for BSBM, from Dosso's benchmark.

Finally, we ran an implementation of Algorithm 1 to obtain a list of solution generators, for each of the selected keyword queries. The parameters described in Section 4.3 were set as $\sigma_1 = 0, \sigma_2 = 5, \sigma_3 = 1.0, \sigma_4 = 0.5, \sigma_5 = 0.2, \sigma_6 = 0.2$, for all datasets.

The above process resulted in 5 synthetic benchmarks using IMDb, Mondial, BSBM, LUBM, and DBpedia. The RDF datasets are available at `https://doi.org/10.6084/m9.figshare.11347676.v3`, and the implementation of the Algorithm 1, the keyword queries, the solution generators, and statistics are available at `https://doi.org/10.6084/m9.figshare.16598813.v1`.

5.2 Results

We compared the baseline benchmark with the corresponding synthetic benchmark for each of the five RDF datasets. Consider the following questions (where $\mathcal{K}$ is a keyword query of both the baseline benchmark and the corresponding synthetic benchmark, as explained above):

Q1. What is the total number $s_{\mathcal{K}}$ of answers expressed by the solution generators for $\mathcal{K}$ in the synthetic benchmark?

Q2. What is the total number $bs_{\mathcal{K}}$ of answers of $\mathcal{K}$, defined in the baseline benchmark, that are expressed by the solution generators for $\mathcal{K}$ in the synthetic benchmark?

Q3. What is the total number $sn_{\mathcal{K}}$ of answers expressed by the solution generators for $\mathcal{K}$ in the synthetic benchmark that are not defined in the baseline benchmark?

Let $B_{\mathcal{K}}$ be the set of answers for $\mathcal{K}$ defined in the baseline benchmark and $b_{\mathcal{K}} = |B_{\mathcal{K}}|$.

To address these questions, one has to compute the set $S_{\mathcal{K}}$ of answers for $\mathcal{K}$ that the solution generators for $\mathcal{K}$ express. It depends on the exact

notion of answer one is adopting. For example, the set of minimal answers can be estimated by counting the minimal Steiner Trees [51, 52] of a solution generator $\mathcal{SG}$ whose terminal nodes are the set of seeds of $\mathcal{SG}$. To compute $|B_{\mathcal{K}} \cap S_{\mathcal{K}}|$, one has to test, for each answer $\mathcal{A} \in B_{\mathcal{K}}$, if there is some solution generator for $\mathcal{K}$ that expresses $\mathcal{A}$. Hence, we have that

- $s_{\mathcal{K}} = |S_{\mathcal{K}}|$
- $bs_{\mathcal{K}} = |B_{\mathcal{K}} \cap S_{\mathcal{K}}|$
- $sn_{\mathcal{K}} = |S_{\mathcal{K}} - B_{\mathcal{K}}| = |S_{\mathcal{K}}| - |B_{\mathcal{K}} \cap S_{\mathcal{K}}|$

Column $\#\mathcal{M}$**s** of Table 5.1 shows the total number of minimal answers expressed by the solution generators for $\mathcal{K}$ that are not defined in the original benchmarks.

Q1 and Q2 lead to an interesting discussion. Consider that the baseline benchmarks are keyword search systems to be evaluated against the synthetic benchmarks. Then, one can compute the precision of the baseline benchmark for $\mathcal{K}$ against the equivalent synthetic benchmark as $p_{\mathcal{K}} = bs_{\mathcal{K}}/b_{\mathcal{K}}$. The larger $p_{\mathcal{K}}$ is, the larger will be the number of correct answers for $\mathcal{K}$, in the baseline benchmark, that the solution generators express. Likewise, one can compute the recall of the baseline benchmark for $\mathcal{K}$ against the equivalent synthetic benchmark as $r_{\mathcal{K}} = bs_{\mathcal{K}}/s_{\mathcal{K}}$.

Table 5.1: Benchmarks statistics obtained for Mondial, IMDb, and DBpedia.

Datasets	Sample Keyword Queries	#Seeds	Precision	#Sol. Generators	#$\mathcal{M}$s
Mondial	niger country	4	1.00	4	23
	haiti religion	2	1.00	1	2
	mongolia china	4	1.00	2	3
	lebanon syria	5	1.00	3	10
	poland cape verde organization	5	0.82	4	132
	rhein germany province	5	0.50	2	82
	OVERALL AVERAGE		**0.91**	**5.00**	**184.83**
IMDb	Johnny Depp Actor	5	1.00	12	46
	Will Smith Male	5	1.00	6	21
	Atticus Finch Movie	5	1.00	10	35
	russell crowe gladiator character	5	0.50	11	21
	sean connery ian fleming work	5	0.11	10	27
	OVERALL AVERAGE		**0.52**	**5.51**	**38.60**
DBpedia	Captain America creator notable works	5	1.00	5	47
	Canada Capital	5	1.00	3	57
	governor of Texas	5	1.00	5	58
	Francis Ford Coppola film director	5	1.00	11	61
	mayor of new york city	5	0.00	2	9
	NASA launchpad	5	0.00	3	5
	OVERALL AVERAGE		**0.58**	**8.22**	**91.86**

Table 5.1 summarizes statistics for Mondial, IMDb, and DBpedia (Appendix D.1 shows the results for all queries). For sample keyword queries, it shows the number of retrieved seeds, the precision values that the baseline benchmarks achieved, the number of solution generators obtained from the

seeds, and the number of minimal answers expressed by the solution generators that are not defined in the baseline benchmarks. For example, for the keyword query $\mathcal{K} = \{\text{“}niger\text{”}, \text{“}country\text{”}\}$ from Mondial, the algorithm selected four seeds: the `country Niger`, the `province Niger`, the `river Niger`, and the class `Country`. Then, it computed four solution generators: 1) with all seeds; 2) with all seeds, except the class `Country`; 3) with two seeds, the class `Country` and the node `Niger`, which is an instance of class `Country`; and 4) with only the class `Country`.

The Overall Averages can be interpreted as the percentage of the correct answers, defined in the baseline benchmarks, that the solution generators express - 91% for Mondial, 52% for IMDb, and 58% for DBpedia - which is quite reasonable for synthetic benchmarks.

The results for IMDb and DBPedia can be explained as follows. Observe that these datasets contain ambiguities, i.e., entities with similar names. The automatic selecting seeds could not always choose the same entities as in the original benchmarks. Furthermore, the ambiguity entities have many data properties, which implies that the inforank score selects the ambiguous entity.

Finally, we remark that, for the synthetic datasets (BSBM and LUBM), Algorithm 1 precisely found the correct answers listed in Dosso's benchmark.

6
Contributions and Future Work

6.1
Contributions

One of the main issues in developing RDF keyword search algorithms is the lack of appropriate benchmarks. The main contribution of the thesis, described in Chapter 4, is a offline method to build benchmarks automatically, allowing larger sets of queries and more complete answers.

The method has two steps: query generation and answer generation. The query generation step, described in Sections 4.1 and 4.2, selects a set of inducers and uses heuristics to extract related queries for each inducer. In particular, Sections 4.2.2 and 4.2.3 discussed a process to improve the readability of natural language queries by improving the verbalization of predicate labels and treating exceptional cases, such as reified classes. The answer generation step takes the queries and computes solution generators (SG), which are subgraphs of the original dataset containing different answers to the queries. We also proposed some heuristics that guide the construction of SGs and avoid irrelevant answers.

The second contribution of the thesis is an implementation of the method, also presented in Chapter 4. We explained the method step by step, with the help of real examples taken from the Mondial database.

The third contribution of the thesis, described in Chapter 5, is an evaluation of the benchmark generation process. We proceeded to describe synthetic benchmarks for IMDb, Mondial, BSBM, LUBM, and DBpedia. The experiments compared the synthetic benchmarks with baseline benchmarks. The results showed that the obtained solution generators express most of the correct answers defined in the baseline benchmarks, but express many more answers for IMDb, Mondial, and DBpedia and precisely the defined answers for the synthetic datasets, BSBM and LUBM.

Partial results related to this thesis, as well other relevant results, were reported in the following articles:

– Neves, A., Leme, L.A.P.P., Izquierdo, Y.T., Casanova, M.A., Automatic Construction of Benchmarks for RDF Keyword Search Systems Evalua-

tion. In Proceedings of the 23rd International Conference on Enterprise Information Systems - Volume 1: ICEIS, ISBN 978-989-758-509-8; ISSN 2184-4992, pages 126-137. (Best student paper award)

– Izquierdo, Y.T., García, G.M., Menendez, E.S., Leme, L.A.P.P., Neves, A., Lemos, M., Finamore, A.C., Oliveira, C., Casanova, M.A. Keyword search over schema-less RDF datasets by SPARQL query compilation. Information Systems, v.102, article 101814 (21 pages), 2021.

– Neves, A.B., Leme, L.A.P.P., Izquierdo, Y.T., Jiménez, J.H., Lopes, G.R., Casanova, M.A. Automatically Creating Benchmarks for RDF Keyword Search Evaluation. Science Nature Computer Science, 3:4, pp 1-17 (July 2022).

6.2 Future Work

As future work, we may suggest the following.

The first suggestion is to improve the process of computing SGs, which we recall is a combinatorial problem.

The second suggestion is to generate answers from keywords that allow Boolean expressions and keyword phrases using quotes.

The third suggestion is to improve the answer generation method for natural language queries. Indeed, this thesis described an answer generation method for keyword queries and then adapted the method to natural language queries. However, natural language queries, and their answers, can express more complex relationships than keyword queries.

For example, the proposed method does not generate natural language questions with aggregations. The fourth suggestion therefore is to adapt the method to generate natural language queries with aggregations (or other language constructions) and their respective answers.

The fifth suggestion stems from the fact that natural language database interfaces, based on Deep Learning methods, are highly dependent on the availability of training datasets, that is, datasets with sets of NL queries and their answers, to obtain good results. Therefore, such systems would benefit from an (easy) adaption of the benchmark construction process described in this thesis to create such training datasets.

The sixth suggestion is to conduct a new experiment to evaluate the keyword queries generated. The experiment would aim at discovering which keyword queries in baseline benchmarks, such as Coffman's, a benchmark generated by the proposed method managed to produce.

The last suggestion for future work is to extend the proposed benchmark generation method to create multi-language natural language queries and their answers.

Bibliography

[1] A. Bordes, J. Weston, and N. Usunier, "Open question answering with weakly supervised embedding models," in Joint European conference on machine learning and knowledge discovery in databases. Springer, 2014, pp. 165–180.

[2] E. S. Menendez, M. A. Casanova, L. A. P. Paes Leme, and M. Boughanem, "Novel Node Importance Measures to Improve Keyword Search over RDF Graphs," in Proceedings of the 31st International Conference on Database and Expert Systems Applications (DEXA'19), vol. 11707, 2019, pp. 143–158.

[3] J. Coffman and A. C. Weaver, "A framework for evaluating database keyword search strategies," in Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM'10). New York, New York, USA: ACM Press, 2010, p. 729.

[4] D. Dosso and G. Silvello, "Search Text to Retrieve Graphs: A Scalable RDF Keyword-Based Search System," IEEE Access, vol. 8, pp. 14 089–14 111, 2020.

[5] M. Dubey, D. Banerjee, A. Abdelkawi, and J. Lehmann, "LC-QuAD 2.0: A Large Dataset for Complex Question Answering over Wikidata and DBpedia," in Proceedings of the 18th International Semantic Web Conference (ISWC'19), 2019, pp. 69–78.

[6] P. Trivedi, G. Maheshwari, M. Dubey, and J. Lehmann, "LC-QuAD: A Corpus for Complex Question Answering over Knowledge Graphs," in Proceedings of the 16th International Semantic Web Conference (ISWC'17), 2017, pp. 210–218.

[7] G. M. García, Y. T. Izquierdo, E. S. Menendez, F. Dartayre, and M. A. Casanova, "RDF Keyword-based Query Technology Meets a Real-World Dataset," EDBT/ICDT 2017, pp. 656–667, 2017.

[8] J. Coffman and A. C. Weaver, "Benchmark for Relational Keyword Search." [Online]. Available: https://doi.org/10.18130/V3/KEVCF8

[9] H. Bast, B. Buchhold, and E. Haussmann, "Semantic Search on Text and Knowledge Bases," Foundations and Trends in Information Retrieval, vol. 10, no. 1, pp. 119–271, 2016.

[10] K. Balog and R. Neumayer, "A test collection for entity search in DBpedia," in Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval. New York, NY, USA: ACM, 7 2013, pp. 737–740.

[11] A. d. C. Oliveira Filho, "Benchmark para métodos de consultas por palavras-chave a bancos de dados relacionais," Universidade Federal de Goiás, Goiás, Tech. Rep., 2018.

[12] Y. T. Izquierdo, G. M. García, E. S. Menendez, M. A. Casanova, F. Dartayre, and C. H. Levy, "QUIOW: A Keyword-Based Query Processing Tool for RDF Datasets and Relational Databases," in Proceedings of the 30th International Conference on Database and Expert Systems Applications (DEXA'18), vol. 11030 LNCS, 2018, pp. 259–269.

[13] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A Benchmark for OWL Knowledge Base Systems," Journal of Web Semantics, vol. 3, no. 2-3, pp. 158–182, 10 2005.

[14] C. Bizer and A. Schultz, "The Berlin SPARQL Benchmark," International Journal on Semantic Web and Information Systems, vol. 5, no. 2, pp. 1–24, 4 2009.

[15] E. Minack, W. Siberski, and W. Nejdl, "Benchmarking fulltext search performance of RDF stores," in Proceedings of the 6th European Semantic Web Conference (ESWC'09), vol. 5554 LNCS. Heraklion, Greece: Springer-Verlag, 2009, pp. 81–95.

[16] M. Poess and J. M. Stephens, "Generating Thousand Benchmark Queries in Seconds," in Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30, ser. VLDB '04. VLDB Endowment, 2004, p. 1045–1053.

[17] W. Zheng, L. Zou, W. Peng, X. Yan, S. Song, and D. Zhao, "Semantic SPARQL similarity search over RDF knowledge graphs," Proceedings of the VLDB (VLDB'16), vol. 9, no. 11, pp. 840–851, 2016.

[18] S. Han, L. Zou, J. X. Yu, and D. Zhao, "Keyword Search on RDF Graphs - A Query Graph Assembly Approach," in Proceedings of the 2017 ACM on

Conference on Information and Knowledge (CIKM'17). New York, NY, USA: ACM, 11 2017, pp. 227–236.

[19] X. Q. Lin, Z. M. Ma, and L. Yan, "RDF keyword search using a type-based summary," Journal of Information Science and Engineering, vol. 34, no. 2, pp. 489–504, 2018.

[20] Y. Wen, Y. Jin, and X. Yuan, "KAT: Keywords-to-SPARQL translation over RDF graphs," in Proceedings of the 23rd International Conference on Database Systems for Advanced Applications (DASFAA'18), vol. 10827 LNCS. Gold Coast, Australia: Springer, 2018, pp. 802–810.

[21] M. Rihany, Z. Kedad, and S. Lopes, "Keyword search over RDF graphs using wordnet," in Proceedings of the 1st International Conference on Big Data and Cyber-Security Intelligence (BDCSIntell'18), vol. 2343, 2018, pp. 75–82.

[22] Q. Zhou, C. Wang, M. Xiong, H. Wang, and Y. Yu, "SPARK: Adapting keyword query to semantic search," in Proceedings of the 6th International Semantic Web Conference (ISWC'07), vol. 4825 LNCS. Busan, Korea: Springer, Berlin, Heidelberg, 2007, pp. 694–707.

[23] G. Zenz, X. Zhou, E. Minack, W. Siberski, and W. Nejdl, "From keywords to semantic queries—Incremental query construction on the semantic web," Journal of Web Semantics, vol. 7, no. 3, pp. 166–176, 9 2009.

[24] T. Tran, H. Wang, S. Rudolph, and P. Cimiano, "Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data," 25th International Conference on Data Engineering (ICDE), pp. 405–416, 2009.

[25] S. Elbassuoni and R. Blanco, "Keyword search over RDF graphs," in Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM'11). Glasgow, UK: ACM Press, 2011, pp. 237–242.

[26] W. Le, F. Li, A. Kementsietsidis, and S. Duan, "Scalable Keyword Search on Large RDF Data," IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 11, pp. 2774–2788, 11 2014.

[27] J. Pound, P. Mika, and H. Zaragoza, "Ad-hoc Object Retrieval in the Web of Data," in Proceedings of the 19th International Conference on World Wide Web (WWW '10). New York, NY, USA: ACM, 2010, pp. 771–780.

[28] K. Affolter, K. Stockinger, and A. Bernstein, "A comparative survey of recent natural language interfaces for databases," VLDB

Journal, vol. 28, no. 5, pp. 793–819, 2019. [Online]. Available: https://doi.org/10.1007/s00778-019-00567-8

[29] D. Saha, A. Floratou, K. Sankaranarayanan, U. F. Minhas, A. R. Mittal, and F. Özcan, "ATHENA: An Ontology-Driven System for Natural Language Querying over Relational Data Stores," Proceedings of the VLDB Endow., vol. 9, no. 12, p. 1209–1220, 2016. [Online]. Available: https://doi.org/10.14778/2994509.2994536

[30] M. Arenas, O. Corcho, E. Simperl, M. Strohmaier, M. D'Aquin, K. Srinivas, P. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, and S. Staab, "TR Discover: A Natural Language Interface for Querying and Analyzing Interlinked Datasets," Lecture Notes in Computer Science, vol. 9367, no. December 2016, 2015.

[31] W. Franco, A. O. R. Franco, C. V. Avila, L. Cabral, G. Maia, V. Pinheiro, V. Vidal, and J. Machado, "ExQuestions: An Expanded Factual Corpus for Question Answering over Knowledge Graphs," in 2022 IEEE 16th International Conference on Semantic Computing (ICSC), 2022, pp. 235–242.

[32] F. Li and H. V. Jagadish, "NaLIR: an interactive natural language interface for querying relational databases," Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOID'14), 2014.

[33] S. Ferré, "SQUALL: a High-Level Language for Querying and Updating the Semantic Web," pp. 1–18, 2011. [Online]. Available: https://hal.inria.fr/inria-00628427/document

[34] S. Ferre, "SPARKLIS: A SPARQL endpoint explorer for expressive question answering," CEUR Workshop Proceedings, vol. 1272, no. October 2014, pp. 45–48, 2014.

[35] A. Marginean, "Question answering over biomedical linked data with Grammatical Framework," Semantic Web, vol. 8, no. 4, pp. 565–580, 2017.

[36] Y. SCHREIBER, GUUS; RAIMOND, "RDF 1.1 Primer," 2014. [Online]. Available: https://www.w3.org/TR/rdf11-primer/#

[37] M. L. R. Cyganiak, D. Wood, "RDF 1.1 Concepts and Abstract Syntax," 2014. [Online]. Available: https://www.w3.org/TR/rdf11-concepts/

[38] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea

Stein, "OWL Web Ontology Language," 2009. [Online]. Available: https://www.w3.org/TR/owl-ref/

[39] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers, "RDF 1.1 Turtle," 2014. [Online]. Available: http://www.w3.org/TR/2014/REC-turtle-20140225/

[40] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," Proceedings of the Conference North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT' 2019), vol. 1, no. Mlm, pp. 4171–4186, 2019.

[41] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized autoregressive pretraining for language understanding," Advances in Neural Information Processing Systems, vol. 32, pp. 1–18, 2019.

[42] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," Proceedings of Workshop at ICLR, 2013.

[43] Dhami Dharti, "Understanding BERT — Word Embeddings," 2020. [Online]. Available: https://medium.com/@dhartidhami/understanding-bert-word-embeddings-7dc4d2ea54ca

[44] Alammar Jay, "The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)," 2021. [Online]. Available: http://jalammar.github.io/illustrated-bert/

[45] F. Baader, D. Calvanese, D. L. McGuinness, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2007.

[46] D. Berardi, D. Calvanese, and G. D. Giacomo, "Reasoning on UML class diagrams," Artif. Intell., vol. 168, pp. 70–118, 2005.

[47] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword searching and browsing in databases using BANKS," in Proceedings of the 18th International Conference on Data Engineering (ICDE'02). IEEE Comput. Soc, 2002, pp. 431–440.

[48] V. Hristidis and Y. Papakonstantinou, "Discover: Keyword Search in Relational Databases," in Proceedings of the 28th International Conference on Very Large Databases (VLDB'02). Elsevier, 2002, pp. 670–681.

[49] B. Kimelfeld and Y. Sagiv, "Efficiently enumerating results of keyword search over data graphs," Information Systems, vol. 33, no. 4-5, pp. 335–359, 6 2008.

[50] B. P. Nunes, J. Herrera, D. Taibi, G. R. Lopes, M. A. Casanova, and S. Dietze, "SCS Connector - Quantifying and Visualising Semantic Paths Between Entity Pairs," in Proceedings of the Satellite Events of the 11th European Semantic Web Conference (ESWC'14), 2014, pp. 461–466.

[51] P. S. de Oliveira, A. Da Silva, E. Moura, and R. De Freitas, "Efficient Match-Based Candidate Network Generation for Keyword Queries over Relational Databases," IEEE Transactions on Knowledge and Data Engineering, p. 1, 2020.

[52] M. C. Dourado, R. A. de Oliveira, and F. Protti, "Generating all the Steiner trees and computing Steiner intervals for a fixed number of terminals," Electronic Notes in Discrete Mathematics, vol. 35, pp. 323–328, 12 2009.

A
Fragment of Benchmark for Dataset: Mondial

Question: What are the Organization O such that India and Mauritius are members of O ?

Keywords: Mauritus, India and Organization

Solution Generator:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX mondial: <http://www.semwebtech.org/mondial/10/>

mondial:IND
    rdf:type Country;
    rdfs:label "India";
    mondial:isMember mondial:IFAD ;
    mondial:isMember mondial:G-77 .

mondial:MS
    rdf:type mondial:Country;
    rdfs:label "Mauritius";
    mondial:isMember mondial:IFAD;
    mondial:isMember mondial:G-77.

mondial:islandMauritius
    rdf:type mondial:Island;
    rdfs:label "Mauritius";
    mondial:locatedInWater; mondial:IndianOcean;
    mondial:locatedIn mondial:MS.

mondial:IndianOcean
   rdf:type mondial:Ocean;
   rdfs:label "Indian Ocean".

mondial:IFAD
    rdf:type mondial:Organization;
    rdfs:label "Int. Fund for Agricultural Development".

```

```
mondial:G-77
   rdf:type mondial:Organization;
   rdfs:label "Group of G-77".

mondial:Country
    rdfs:label "Country".

mondial:Organization
    rdfs:label "Organization".

mondial:Island
   rdfs:label "Island".
```

Answer 1:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX mondial: <http://www.semwebtech.org/mondial/10/>

mondial:IND
    rdf:type Country;
    rdfs:label "India";
    mondial:isMember mondial:IFAD;
    mondial:isMember mondial:G-77.

mondial:MS
    rdf:type mondial:Country;
    rdfs:label "Mauritius";
    mondial:isMember mondial:IFAD;
    mondial:isMember mondial:G-77.


mondial:IFAD
    rdf:type mondial:Organization;
    rdfs:label "Int. Fund for Agricultural Development".

mondial:G-77
   rdf:type mondial:Organization;
   rdfs:label "Group of G-77".

mondial:Country
    rdfs:label "Country".

mondial:Organization
    rdfs:label "Organization".
```

Answer 2:

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

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX mondial: <http://www.semwebtech.org/mondial/10/>

mondial:IND
    rdf:type Country;
    rdfs:label "India";
    mondial:isMember mondial:IFAD;
    mondial:isMember mondial:G-77.

mondial:MS
    rdf:type mondial:Country;
    rdfs:label "Mauritius";
    mondial:isMember mondial:IFAD;
    mondial:isMember mondial:G-77.

mondial:islandMauritius
    rdf:type mondial:Island;
    rdfs:label "Mauritius";
    mondial:locatedIn mondial:MS.

mondial:IFAD
    rdf:type mondial:Organization;
    rdfs:label "Int. Fund for Agricultural Development".

mondial:G-77
   rdf:type mondial:Organization;
   rdfs:label "Group of G-77".

mondial:Country
    rdfs:label "Country".

mondial:Organization
    rdfs:label "Organization".

mondial:Island
   rdfs:label "Island".
```

B
Natural Language Sentences generated in the Mondial dataset

Table B.1: Natural Language Sentence Results.

Results by ismember predicate		
Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
India is member International Fund for Agricultural Development Italy is member the Black Sea Economic Cooperation Zone United Arab Emirates is member Group of 77.	Algeria is a member of the African Union, the Arab League, OPEC, the United Nations and is the founding member of the Maghreb Union	is a member of
Results by hascity predicate		
Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
The United States has city Arlington India has city Ahmadabad France has city Paris	The archipelago, with a total population of nearly 254,000 inhabitants, has the city of Funchal as its most important centre.	has the city of
Results by hasprovince predicate		
Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
China has province Heilongjiang Russia has province Primorskiy. India has province Chhattisgarh.	It has the province of Catanzaro to the north, Reggio di Calabria.	has the province of
Results by locatedin predicate		
Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
Acre located in Peru Maipo located in Chile. Donau located in Slovakia.	The city is located in the province of North Holland in the west of the country.	is located in

Results by locatedat predicate

Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
Formosa located at Paraguay	India Gate, which is located at the eastern end of the Rajpath.	is located at
Posadas located at Parana.		
Minneapolis located at Mississippi.		

Results by mergeswith predicate

Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
Skagerrak merges with Kattegat.	By sea Ukraine borders with Romania and Russia.	borders with
Kattegat merges with Skagerrak.		

Results by dependentof predicate

Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
Hong Kong dependent of China.	This index describes how a country is dependent of importation and how diverse are its importation.	is dependent of
Puerto Rico dependent of United States.		
Macao dependent of China.		

Results by inmountains predicate

Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
Mt. Everest in mountains Himalaya.	Our future home is in the mountain of the house of the Lord in the world to come.	is in the mountain of
Iremel in mountains Ural.		
Montalto in mountains Apennin.		

Results by hasheadquarters predicate

Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
Bank for International Settlements has headquarters Basel	The World Council of Churches, including Orthodox Churches, has its headquarters in Geneva, Switzerland.	has its headquarters in
International Energy Agency has headquarters Paris.		
International Civil Aviation Organization has headquarters Montréal.		

Results by capital predicate

Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
United States capital Washington D C	For example, do you know what country has the capital in Bogota?	has the capital in
United Kingdom capital London		
Brazil capital Brasilia		

Results by encompassed predicate

Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
China encompassed Asia	The majority of the United States is encompassed by either the North American ECA or the U S.	is encompassed by
Russia encompassed Europe.		
Mali encompassed Africa		

Results by dependentof predicate

Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
Mongolia dependent of China	After China was dependent of bigger countries, they were afraid it would happen to them.	was dependent of
India dependent of United Kingdom.		
Qatar dependent of United Kingdom		

Results by locatedin predicate

Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
Chongming located in water Yangtze	Chongming is located in the Yangtze River, dividing the river into northern and southern channels.	is located in the
Fünen located in water Kattegat.		
Seeland located in water Kattegat.		

Results by flowsthrough predicate

Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
Ammer flows through Ammersee	What river flows through Kiev Ukraine?.	river flows through
Aare flows through Brienzersee.		
Vuoksi flows through Saimaa.		

Results by locatedinwater predicate

Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
Chongming located in water Yangtze	Chongming is located in the Yangtze River, dividing the river into northern and southern channels.	is located in the
Fünen located in water Kattegat.		
Seeland located in water Kattegat.		
Results by locatedonisland predicate		
Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
Psiloritis located on island Crete	Faedra Beach is located on the island of Crete to the east of Heraklion.	is located on the island of
Galway located on island Ireland.		
Results by neighbor predicate		
Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
Hungary neighbor Slovakia	Ukraine borders Poland.	borders
Brazil neighbor Guyana.		
Bolivia neighbor Peru.		

C
Natural Language Sentences generated for reified Relationships in the Mondial dataset

Table C.1: Natural Language Sentence Results - Reified Relationships.

Reified relationship by SpokenBy class		
Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
French Spoken By Luxembourg	English is spoken by American people.	is spoken by
Spanish Spoken By Argentina.		
Hindi Spoken By India.		
Reified relationship by BelievedBy class		
Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
Christian believed by India	God is believed by the mystic to be real outside of the occasional mystical experiences, and to reveal himself in the experiences.	is believed by
Muslim believed by Bulgaria.		
Anglican believed by Canada.		
Reified relationship by EthnicProportion class		
Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
Polish Ethnic Proportion Ukraine	the Polish has Ethnic group in Ukraine.	has ethnic group in
German Ethnic Proportion Germany.		
Roma Ethnic Proportion Slovakia.		
Reified relationship by Membership class		

Labeled RDF Triples	Selected Sentence from Corpus	Extracted Predicate Label
Feerum membership Lower Silesian Chamber of Commerce	Germany is a membership of the Lower Silesian Chamber of Commerce.	is a membership of the
Austria Membership Food and Agriculture Organization.		
Ukraine Membership United Nations Conference on Trade and Development.		

D
Benchmarks Statistics

Table D.1: Benchmarks statistics obtained for Mondial, IMDb, and DBpedia.

Datasets	Sample Keyword Queries	#Seeds	Precision	#SG	#$\mathcal{M}$s
Mondial	niger country	4	1.00	4	23
	spain galician	5	1.00	5	20
	poland language	5	1.00	5	22
	haiti religion	2	1.00	1	2
	mongolia china	4	1.00	2	3
	lebanon syria	5	1.00	3	10
	poland cape verde organization	5	0.82	4	132
	iceland mali organization	5	0.84	4	150
	mauritius india organization	5	0.89	8	532
	vanuatu afghanistan organization	5	0.87	16	850
	hutu country africa	5	1.0	6	380
	rhein germany province	5	0.50	2	82
	OVERALL AVERAGE		**0.91**	**5.00**	**184.83**
IMDb	Denzel Washington Person	5	1.00	1	2
	Johnny Depp Actor	5	1.00	12	46
	Forrest Gump Work	5	1.00	4	29
	Star Wars Movie	5	0.16	1	3
	Angelina Jolie gender	5	1.00	3	5
	the sound of music length	5	1.00	2	3
	lord of the rings novel	5	1.00	7	12
	Will Smith Male	5	1.00	6	21
	tom hanks 9 July 1956	5	1.00	3	60
	gone with the wind August 1991	5	0.50	10	92
	casablanca They had a date with fate in Casablanca	5	0.50	1	3
	johnny depp Work	5	1.00	6	15
	morgan freeman Work	5	1.00	4	8
	atticus finch movie	5	0.50	10	35
	indiana jones movie	5	0.20	10	150
	james bond movie	5	0.43	3	26
	will kane movie	5	0.00	3	8
	dr hannibal lecter movie	5	0.50	2	7
	darth vader movie	5	0.20	2	5
	the wicked witch of the west movie	5	0.50	3	6

	nurse ratched movie	5	0.50	9	21
	jack ryan actor	5	0.25	11	40
	terminator actor	5	0.25	4	177
	clint eastwood Frank Horrigan	5	1.00	4	4
	tom hanks 2004	5	0.20	6	145
	audrey hepburn 1951	5	0.14	5	8
	julia roberts richard gere work	5	0.00	4	45
	harrison ford george lucas work	5	0.09	2	119
	sean connery ian fleming work	5	0.11	10	27
	indiana Jones and the last crusade raiders of the lost ark person	5	0.40	24	180
	nathan algren tom cruise Work	5	0.50	2	7
	rocky balboa sylvester stallone Work	5	0.50	3	15
	Henry Jaynes Fonda Yours Mine and Ours character	5	0.50	2	4
	russell crowe gladiator character	5	0.40	11	21
	brent spiner work star trek the next generation character	5	0.20	3	3
	OVERALL AVERAGE		**0.52**	**5.51**	**38.60**
DBpedia	Clint Eastwood starring director	5	1.00	7	52
	John F Kennedy predecessor	5	1.00	1	9
	Governing Mayor of Berlin	5	0.00	2	21
	current tenants prime minister of spain	5	1.00	3	25
	professional skateboarders Sweden	5	0.00	3	36
	postalabbreviation MN	5	1.00	4	97
	occupation bandleader instrument trumpet	5	0.82	20	146
	Kerouac author Viking Press publisher	5	1.00	5	96
	world heritage sites	5	1.00	7	150
	Captain America creator notable works	5	1.00	5	47
	prodigy associated acts	5	1.00	7	11
	female Russian astronauts	5	0.66	10	10
	automobile assembly Germany	5	0.00	8	11
	Vienna place of Birth Death Berlin	5	0.00	3	63
	Canada Capital	5	1.00	3	57
	governor of Texas	5	1.00	5	58
	issue Elizabeth II	5	1.00	39	150
	SPD ruling party	5	1.00	2	60
	Sean Parnell governor	5	0.00	14	190
	Sean Parnell governor	5	0.00	14	190
	Francis Ford Coppola film director	5	1.00	11	61
	starring director William Shatner	5	0.00	6	10

US state gold mineral	5	0.00	3	95
Australian company type nonprofit organization	5	0.36	5	20
T E Lawrence battles	5	1.00	2	124
products Skype	5	1.00	1	3
company location Munich	5	0.00	2	30
game GMT publisher	5	0.00	3	44
Intel founder	5	1.00	28	180
Amanda Palmer spouse	5	1.00	20	220
German Shepherd Dog breed	5	0.63	25	185
Weser city	5	1.00	5	91
Rhine city	5	1.00	8	130
born Philippines occupation surfing	5	0.00	5	197
goofy creator	5	0.20	6	45
Uzi designer	5	0.50	20	173
Frisian island subdivisionName netherland	5	0.00	4	97
Lisbon leader party	5	1.00	13	230
abode Mount Olympus	5	1.00	20	135
Apollo 14 mission astronaut	5	1.00	8	135
Salt Lake City timezone	5	0.00	6	170
mayor of new york city	5	0.00	2	9
lake Denmark basin country	5	0.00	5	162
africa city capital	5	0.00	20	217
NASA launchpad	5	0.00	3	5
John Lennon instrument	5	1.00	6	115
The Scream Edvard Munch museum	5	0.00	3	35
television show Walt Disney creator	5	1.00	9	115
Area 51 location	5	1.00	8	180
Margaret Thatcher children	5	1.00	2	93
Scarface nickname	5	0.00	4	25
OVERALL AVERAGE		**0.58**	**8.22**	**91.86**