Pontifícia Universidade Católica
do Rio de Janeiro

José Eduardo Talavera Herrera

On the Connectivity of Entity Pairs in Knowledge Bases

Tese de Doutorado

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

Advisor: Prof. Marco Antonio Casanova

Rio de Janeiro

May 2017

Pontifícia Universidade Católica
do Rio de Janeiro

José Eduardo Talavera Herrera

On the Connectivity of Entity Pairs in Knowledge Bases

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 undersigned Examination Committee.

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

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

Prof. Bernardo Pereira Nunes
Departamento de Informática – PUC-Rio

Prof. Giseli Rabello Lopes
UFRJ

Prof. Luiz André Portes Paes Leme
UFF

Prof. Marcio da Silveira Carvalho
Vice Dean of Graduate Studies
Centro Técnico Científico – PUC-Rio

Rio de Janeiro, May 19th, 2017

José Eduardo Talavera Herrera

José Eduardo Talavera Herrera holds a Master in Informatics degree from the Pontifical Catholic University of Rio de Janeiro (PUC-Rio), and a System Engineering degree from the University of San Agustín (UNSA). José worked as a system analyst at the Bank of Credit of Peru (BCP). In addition, he worked as a system analyst at the Central Coordination for Planning and Evaluation (CCPA) of PUC-Rio. His main research topic areas include Semantic Web, Information Retrieval, Linked Data and Cloud Computing.

Bibliographic data

Talavera Herrera, José Eduardo

On the Connectivity of Entity Pairs in Knowledge Bases / José Eduardo Talavera Herrera ; advisor: Marco Antonio Casanova. – 2017.

102 f. ; 30 cm

Tese (doutorado) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2017.

Inclui bibliografia

1. Informática – Teses. 2. Busca de Caminhos. 3. Medidas de Similaridade. 4. Ranqueamento de Caminhos. 5. Grafos RDF. 6. Consultas SPARQL I. Casanova, Marco Antonio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CCD: 004

I dedicate this thesis to my mother.

Acknowledgments

First of all, I would like to express my deep gratitude to my advisor, Prof. Dr. Marco Antonio Casanova, who are always available to share new ideas.

I am also thankful with to Prof. Bernardo Pereira Nunes, Prof. Giseli Rabello Lopes, Prof. Luiz André Portes Paes Leme and Prof. Antonio Luz Furtado, who helped me finish this work.

I also would like to thank Michele dos Santos Soares, who with her patience and love helped me persevere with optimism this challenge.

Thanks to CAPES, FAPERJ and PUC-Rio for the grants that made this research possible.

Abstract

Talavera Herrera, José Eduardo; Casanova, Marco Antonio (Advisor). **On the Connectivity of Entity Pairs in Knowledge Bases**. Rio de Janeiro, 2017. 102p. Tese de Doutorado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Knowledge bases are a powerful tool for supporting a large spectrum of applications such as exploratory search, ranking, and recommendation. Knowledge bases can be viewed as graphs whose nodes represent entities and whose edges represent relationships. Currently, search engines take advantage of knowledge bases to improve their recommendations. However, search engines are single entity-centric and face difficulties when trying to explain why and how two entities are related, a problem known as entity relatedness. This thesis explores the use of knowledge bases in RDF format to address the entity relatedness problem, in two directions. In one direction, it defines the concept of connectivity profiles for entity pairs, which are concise explanations about how the entities are related. The thesis introduces a strategy to generate a connectivity profile for an entity pair that combines semantic annotations and similarity metrics to summarize a set of relationship paths between the given entity pair. The thesis then describes the DBpedia profiler tool, which implements the strategy for DBpedia, and whose effectiveness was evaluated through user experiments. In another direction, motivated by the challenges of exploring large online knowledge bases, the thesis introduces a generic search strategy, based on the backward search heuristic, to prioritize certain paths over others. The strategy combines similarity and ranking measures to create different alternatives. Finally, the thesis evaluates and compares the different alternatives in two domains, music and movies, based on specialized path rankings taken as ground truth.

Keywords

Pathfinding; Similarity Measure; Path Ranking; RDF Graph; SPARQL Query.

Resumo

Talavera Herrera, José Eduardo; Casanova, Marco Antonio. **Sobre a Conectividade de Pares de Entidades em Bases de Conhecimento**. Rio de Janeiro, 2017. 102p. Tese de Doutorado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Bases de conhecimento são ferramentas poderosas que fornecem suporte a um amplo espectro de aplicações como, por exemplo, busca exploratória, ranqueamento e recomendação. Bases de conhecimento podem ser vistas como grafos, onde os nós representam entidades e as arestas seus relacionamentos. Atualmente, motores de busca usam bases de conhecimento para melhorar suas recomendações. No entanto, motores de busca são orientados a uma única entidade e enfrentam dificuldades ao tentar explicar porque e como duas entidades estão relacionadas, um problema conhecido como relacionamento entre entidades. Esta tese explora o uso de bases de conhecimento em formato RDF para endereçar o problema de relacionamento entre entidades, em duas direções. Em uma direção, a tese define o conceito de perfis de conectividade para pares de entidades, que são explicações concisas sobre como as entidades se relacionam. A tese introduz uma estratégia para gerar um perfil de conectividade entre um par de entidades, que combina anotações semânticas e métricas de similaridade para resumir um conjunto de caminhos entre as duas entidades. Em seguida, introduz a ferramenta *DBpedia profiler*, que implementa a estratégia proposta, e cuja efetividade foi medida através de experimentos com usuários. Em outra direção, considerando os desafios para explorar grandes bases de conhecimento online, a tese apresenta uma estratégia genérica de busca baseada na heurística *backward*, a qual prioriza alguns caminhos sobre outros. A estratégia combina medidas de similaridade e de ranqueamento, criando diferentes alternativas. Por último, a tese avalia e compara as diferentes alternativas em dois domínios, música e filmes, adotando como *ground truth* rankings especializados de caminhos especialmente desenvolvidos para os experimentos.

Palavras-chave

Busca de caminhos; Medidas de Similaridade; Ranqueamento de Caminhos; grafos RDF; Consultas SPARQL.

Table of Contents

List of Figures

List of Tables

List of Acronyms

RDF	Resource Description Framework
URI	Uniform Resource Identifier
KB	Knowledge Base
PMI	Pointwise Mutual Information
ITF-IDF	Term Frequency–Inverse Document Frequency
J&I	Jaccard Distance & Predicate Frequency Inverse Triple Frequency
J&E	Jaccard Distance & Exclusivity-based Relatedness
J&P	Jaccard Distance & Pointwise Mutual Information
W&I	Wikipedia Link-based Measure & Predicate Frequency Inverse Triple Frequency
W&E	Wikipedia Link-based Measure & Exclusivity-based Relatedness
W&P	Wikipedia Link-based Measure & Pointwise Mutual Information
S&I	SimRank & Predicate Frequency Inverse Triple Frequency
S&E	SimRank & Exclusivity-based Relatedness
S&P	SimRank & Pointwise Mutual Information

1
Introduction

1.1.
Motivation

Knowledge bases are an effective mechanism to provide information about entities, and provide an implicit method to relate entities (BERNERS-LEE, 2006; BIZER *et al.*, 2009). Knowledge bases can be viewed as graphs with nodes representing entities and edges representing their relationships. Therefore, knowledge bases are a powerful tool for supporting a large spectrum of applications, such as exploratory search, ranking, recommendation, and Web search (Dong *et al*., 2014).

For example, search engines take advantage of knowledge bases to improve their recommendations. When processing a search query, a search engine may identify the real-world entity that is being referenced to in the search query and link it to a knowledge base. Then, the search engine can recommend related entities based on the relationships explicitly encoded in the knowledge base. In general, a search engine can exploit the content of a knowledge base to not only display additional facts and direct information about an entity, but also suggest related entities and explain the relationships that are being presented to the user (Pound *et al.*, 2010). This last feature is called *entity relatedness*.

Figure 1 gives an example of the information provided by search engines, where the search result page retrieved by Bing presents information about the entity "Michael Jackson". The content shows information from Bing's semi-structured knowledge base and others databases, such as Wikipedia and IMDb. It also displays several multimedia and social media resources associated with the central entity. Additionally, search engines provide a rank of related entities, accompanied with explanations of their relationships. This feature can also be seen on search engines, like Google and Yahoo!.

Figure 1: Search result page for the query “Michael Jackson” on Bing, in top of the Figure, the entities related with central entity.

In spite of the simplified recommendation process of related entities that search engines provide, they often fail to generate explanations for entities that are not directly connected to the central entity or entities that are less popular. For example, in Figure 1, the direct relationship between entities “Michael Jackson” and “Paris-Michael Katherine Jackson” is clearly explained by the natural language term “Daughter”. However, it is difficult for the user to understand why “Michael Jackson” is related to “Whitney Houston” or to “Donald Trump”. In general, search engines fail to explain less evident or non-trivial relationships.

The problem of entity relatedness is also important in several domains, such as business markets, academic community, and government agencies. In the business markets domain, product-client knowledge bases help improve the recommendation of new products, and generate effective advertisements to potential clients on the Internet. In a similar way, the academic community takes advantage of co-authorship relationships to detect communities and possible scenarios of collaboration between scientists of the same filed. The government agencies pay attention to their security systems to prevent terrorist attacks, where the extraction and identification of complex relations between entities becomes a

priority to discover semantic relationships that link people with terrorist organizations.

However, in large knowledge bases, most of the entities share many direct relations and relationships paths, thus it becomes a challenge to identify relevant sequences of relationships that explain the connectivity between the entities.

1.2. Challenges

Motivated by the entity relatedness problem, this thesis explores the connectivity of entities in knowledge bases to understand why and how two entities are related.

In what follows, a *relationships path* in a knowledge base is a sequence of entities and properties that connect two target entities. Currently, several approaches have been proposed to explore the connectivity of the two entities in large knowledge bases (HEIM *et al*., 2009; FANG *et al*., 2011; PIRRÒ, 2015; CHENG *et al*., 2014; MOHAN *et al*., 2014). Such approaches apply a simple strategy:

- Search for relationship paths between pairs of entities – the larger the number of paths found, the stronger the connectivity between the entities is likely to be.
- Sort the paths found and select relevant paths;
- Summarize the groups of paths.

However, this simple strategy poses three challenges:

(1) How to find relationship paths between a given pair of entities in a knowledge base?

Indeed, a knowledge base may contain millions of entities and billions of relationships, and it may make the data available as RDF dumps, through simple HTTP interfaces or using the SPARQL protocol. Furthermore, the access interfaces are restricted directly by the request time limit and by the size of query result. Thus, if one wants to find relevant relationship paths between two entities, he must devise an efficient graph-exploration method.

(2) How to sort and select the most relevant relationship paths?

The methods proposed in the literature generate a ranking of relationship paths to explain the connectivity between two entities, and they use metrics, such as frequency, rarity and informativeness, to express the relevance of a relationship path. A ranking is used to exclude information, with the side effect that the user cannot infer the hidden meaning of each relationship path, and therefore he still has to decide which relationship paths to explore. However, the large number of relationship paths and the lack of semantics of the results make it difficult to interpret and explore the connectivity of the pair of entities.

(3) How to group the relationship paths in a meaningful way?

In knowledge bases, the description of the entities contains several type and property values and relationships with other entities. Intuitively, in a group of entities, it is possible to detect topics of interest through the overlapping of common characteristics. However, the description of an entity can be noisy, having excessive data that is not relevant, and there can be a large number of relationship paths which may contribute to an increase in noise, errors, and ambiguities. Therefore, a mechanism for organizing a large set of relationship paths between the entities in a concise and detailed way is needed.

Another important challenge that must also be addressed is:

(4) How to evaluate and compare search strategies that explore the connectivity of entity pairs?

Today, there is no adequate benchmarks that cover all facets of the approaches exploring entity relatedness. In some cases, to compare exploratory approaches, the results of the search tasks are evaluated by expert users, and an apparently reliable method to judge the effectiveness of the approach is introduced. In others, a *ground truth* is created, which is a difficult and time-consuming task. Therefore, the definition of a strategy to evaluate and compare exploratory approaches is an interesting challenge that needs to be addressed and discussed.

1.3. Contributions

This thesis contributes to the entity relatedness problem by defining, implementing and evaluating strategies to explore the connectivity of two entities in knowledge bases in RDF format.

The first contribution is a strategy to generate *connectivity profiles* for entity pairs represented in a knowledge base. The strategy uses the SPARQL language to describe the relationship paths and generate semantic annotations. It applies similarity metrics to summarize the observed groups of relationship paths, creating *explanation paths,* and uses topic categories to enhance the hidden content of the explanation paths. Finally, it presents to the user a *connectivity profile* that explains the connectivity of the entity pair, in a concise, yet detailed way.

The second contribution is the introduction of the DBpedia profiler tool, which implements the strategy to generate connectivity profiles for entity pairs represented in DBpedia. Briefly, the DBpedia profiler receives as input two entities, v_{start} and v_{end}, and a distance k, which sets the maximum size of a relationship path between v_{start} and v_{end}. The tool creates a set of SPARQL queries and sends them to the DBpedia endpoint to extract relationship paths that connect v_{start} and v_{end}. It generates a stream of relationship paths, and annotates entities and paths using the hierarchy class of the entities in DBpedia and their topic categories. It then groups the relationship paths and organizes them using a cluster algorithm, creating explanation paths. Finally, it presents a set of connectivity profiles to the user.

In order to validate these contributions, user evaluation and performance comparisons were executed. In the user evaluation experiment, experts in Semantic Web technology expressed that the experience using the connectivity profiles strategy was more satisfactory in resolving the search tasks than competitive tools. The performance evaluation over DBpedia revealed that the proposed strategy executes the search tasks efficiently.

The third contribution is the development of a family of search strategies that combine entity similarity measures and path-ranking approaches to find relevant paths between entity pairs. All strategies follow a backward search heuristic with the aim of covering any RDF graph of an online knowledge base. In the backward

search, the expansion process uses simple HTTP requests to recover the entity description. The strategies differ with respect to: (1) the entity similarity measure they adopt to guide the expansion process and how they use the entity degree to prioritize certain paths over others; (2) the path-ranking approach they select to determine the relevance of the relationships paths that link the entity pair.

The fourth contribution of this thesis is the definition of a ground truth in the music and movies domains to compare the search strategies based on backward search heuristic. For each domain, the ground truth consists of a set of entity pairs and, for each pair, a set of relationship paths, ranked using domain-specific knowledge.

In the course of this research, contributions were published in conferences in the areas of Semantic Web and Web Science. Motivated by our Best Demo Paper Award (NUNES *et al.*, 2014), we developed the connectivity profile strategy. The DBpedia profiler tool was published in (HERRERA *et al.*, 2016). The work on the search strategies, including the construction of a ground truth, was submitted to a large database conference and is under review at present.

1.4. Organization

The remainder of this thesis is structured as follows. Chapter 2 gives an introduction to RDF concepts and SPARQL queries, and reviews entity similarity measures and relationship-path ranking measures. Chapter 3 discusses related work. Chapter 4 presents a strategy to generate connectivity profiles for entity pairs represented in a knowledge base. Chapter 5 describes the experimental study to compare search strategies in a knowledge base. Finally, Chapter 6 concludes with a summary of the contributions of the thesis, and directions for future work.

2
Background

This section very briefly reviews some basic concepts of the Resource Description Framework (RDF) data model, the SPARQL query language, Linked Data, similarity measures and path-ranking approaches.

2.1.
RDF

A *Uniform Resource Identifier* (URI) represents a *resource* or an *entity* of the real world. A literal is a string that represents a (datatype) value. An *RDF term* is a URI or a literal. We denote the set of URIs by $\mathcal{U}$ and the set of literals by $\mathcal{L}$. An example of URI is dbr:Albert_Einstein[1].

An RDF triple is a triple $(s, p, o) \in \mathcal{U} \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L})$, which states that its *subject* s has *property* p whose value is *object* o. Note that the object can be a URI or a literal. For example,

(dbr:Ernest_Mach, **dbo:influenced**, dbr:Albert_Einstein)

is an RDF triple that states that Ernest Mach was influenced by Albert Einstein. This triple can also be denoted without parentheses and commas:

dbr:Ernest_Mach **dbo:influenced** dbr:Albert_Einstein .

A list of tuples is denoted with a dot "." separating the tuples, as in:

dbr:David_Hume **dbp:influenced** dbp:Edmund_Husserl .
dbp:Edmund_Husserl **dbp:influenced** dbr:Kurt_Gödel .

We disregard the so-called *blank nodes* in this work, which is always possible by replacing blank nodes by Skolem URIs (CYGANIAK *et al.*, 2014).

[1] http://dbpedia.org/resource/Albert_Einstein

A *knowledge base B* is a set of RDF triples. We say that an *entity* of *B* is a URI that occurs as a subject or object of a triple in *B* and denote the set of all entities of *B* as EN_B. The *RDF graph* of *B* is the edge-labelled graph $G_B = (N_B, E_B, e_B)$ such that N_B is the set of RDF terms that occur as subject or object of a triple in *B* and there is an edge (s, o) in E_B, labelled with $e_B((s, o))=p$, iff there is a RDF triple (s, p, o) in *B*. For example, DBpedia has the triple (see Figure 1) (dbr:Michael_Jackson, dbo:genre, dbr:Pop_music) for the entity dbr:Michael_Jackson. We will use the terms *entity* of *B* and *node* of the RDF graph G_B interchangeably. As example in DBpedia, Figure 2 shows an RDF graph between Albert Einstein and Kurt Gödel.

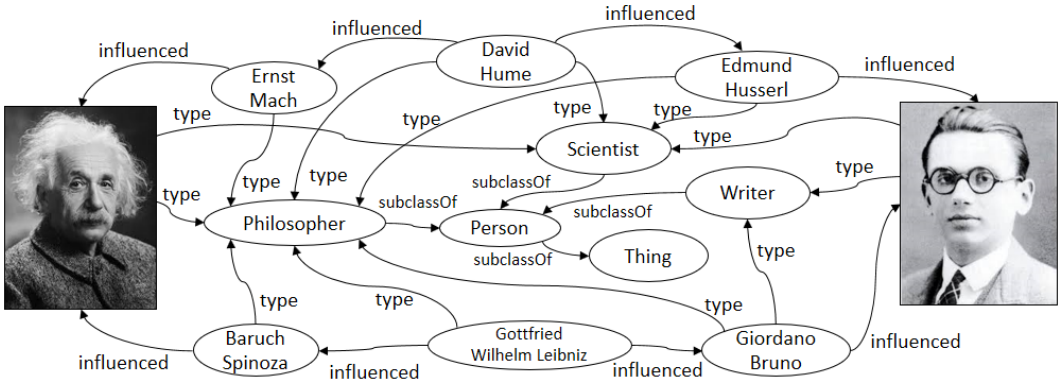

Figure 2: Example of a RDF Graph between Albert Einstein and Kurt Gödel.

A *path* π of *B* is an undirected path in the graph G_B induced by *B*. The notions of *start node* and *end node* of a path are defined as usual. Note that, by considering an undirected path, we allow the edges of the RDF graph to be reversely traversed. A path π then corresponds to a sequence of RDF triples $(s_1, p_1, o_1),\ldots,(s_k, p_k, o_k)$ such that $s_{i+1} = o_i$, for $i=1,\ldots k\text{-}1$, and either $(s_i, p_i, o_i) \in B$ or $(o_i, p_i, s_i) \in B$, for $i=1,\ldots k$ (the second triple accounts for the possible traversal of an edge in the reverse direction). For example, the following list of RDF triples represent a path that connects Albert Einstein and Kurt Gödel, where $k = 4$, "Albert Einstein" is the *start node* and "Kurt Gödel" the *end node*:

dbr:Albert_Einstein **^dbp:influenced** dbr:Ernest_Match .
dbr:Ernest_Match **^dbp:influenced** dbr:David_Hume .
dbr:David_Hume **dbp:influenced** dbp:Edmund_Husserl .
dbp:Edmund_Husserl **dbp:influenced** dbr:Kurt_Gödel .

At this point, we recall that, in SPARQL notation, "$\hat{}p$" denotes the inverse of a property *p*.

Entities are typically assigned to *classes*, which may in turn be organized as a *class hierarchy*. This is captured in RDF with the help of the predefined terms rdf:type, rdfs:Class and rdfs:subclassOf, where the first term belongs to the RDF vocabulary and the last two to the RDF Schema vocabulary. The term owl:Thing of the OWL vocabulary denotes the universe (the set of all things). The class hierarchy of B induces a graph $H_B = (C_B, S_B)$ such that C_B is the set of classes declared in B and $(C_i, C_j) \in S_B$ iff C_i is declared as a subset of C_j in B. As an abuse of language, we refer to H_B also as the class hierarchy of B. Continuing with Figure 2 to explain the last concepts, the RDF triple

dbr:Albert_Einstein **rdf:type** dbo:Scientist .

captures the fact that Albert is of the type Scientist, and the triple

dbo:Scientist **rdfs:subclassOf** dbo:Person .

describes that the class dbo:Scientist is a subclass of dbo:Person in the class hierarchy of DBpedia.

2.2. SPARQL Concepts

Let $\mathcal{V}$ be a set of variables, disjoint from $\mathcal{U}$ and $\mathcal{L}$. Variables are denoted in SPARQL prefixed with "?", such as "?x". A *triple pattern* in a member of $(\mathcal{U} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{V} \cup \mathcal{L})$. That is, a triple pattern is like an RDF triple, except that the subject, predicate or object can be a variable. A *basic graph pattern* (BGP) is a set of triple patterns.

The evaluation of a query binds values to the variables using a *solution mapping* (HARRIS *et al*., 2013). The application of a solution mapping to a basic graph pattern *b*, denoted by $\mu[b]$, uniformly replaces each variable in *b* by the RDF term given by μ (HARTIG *et al*. 2009). We say that $\mu[b]$ is *correct* against a knowledge base *B* iff $\mu[b]$ is a subgraph of *B*.

We use the SPARQL query language (HARRIS *et al*., 2013) to access a knowledge base. Specifically, we adopt SPARQL query patterns to find RDF paths

between a given pair of entities, as well as to represent explanations for the connectivity of entity pairs.

For example, given the RDF graph of the Figure 2, to identify RDF paths of length 4 between dbr:Albert_Einstein and dbr:Kurt_Gödel, one may use the following SPARQL query:

SELECT * FROM { dbr:Albert_Einstein (p_1| ^p_1) ?e_1 .
?e_1 (p_2 | ^p_2) ?e_2 .
?e_2 (p_3 | ^p_3) ?e_3 .
?e_3 (p_4 | ^p_4) dbr:Kurt_Gödel . }

In this query, the BGP is the set of triple patterns that describe the possible entities e_i and properties p_i between dbr:Albert_Einstein and dbr:Kurt_Gödel, and the solution mapping is a set RDF triples that represents the graph that connects the last entities, such as the RDF path defined in Section 2.1. The expression "(p_i | ^p_i)" allows an edge to be traversed in both directions.

2.3. Linked Data

The collection of interrelated RDF datasets on the Web is referred as Linked Data. Four principles guide the publication of Linked Data (BERNERS-LEE, 2006; BIZER *et al.*, 2009):

(1) Use URIs to name (identify) things.
(2) Use HTTP URIs so that these things can be looked up (interpreted, "dereferenced").
(3) Provide useful information about what a name identifies when it's looked up, using open standards such as RDF, SPARQL, etc.
(4) Refer to other things using their HTTP URI-based names when publishing data on the Web.

We are interested in the identification of entities with URI references that can be resolved over the HTTP protocol into RDF data that describes the identified entity. These descriptions can include RDF links pointing to other data sources. RDF links take the form of RDF triples, where the subject of the triple is a URI reference in the namespace of one data source, while the object is a URI reference in the namespace of the other (HARTIG *et al.*, 2009). Thus, Linked Data connects

data from different sources via RDF links and, therefore, Linked Data can be understood as a single, globally distributed dataspace (FRANKLIN *et al.*, 2005). Linked Data is accessed in different manners, such as: through a SPARQL Endpoint, through an URI, or by processing HTML pages.

2.4. Similarity Measures

Similarity involves the assessment of intrinsic common characteristics between two or more concepts. A characteristic is intrinsic to an object when it defines the nature of the object itself and cannot be separated from it (GUESSOUM *et al.*, 2015). Similarity measures quantitatively or qualitatively estimate the robustness of semantic relations between units of language, concepts, or instances through a numerical or symbolic description obtained according to the comparison of information formally or implicitly supporting their meaning or describing their nature (HARISPE *et al.*, 2015).

The work from (EHRIG *et al*, 2005) classifies the "contextual" semantic similarity measures between ontologies and intra-ontologies into three layers. First, in the data layer, similarity measures are only simple measures between the values of the entities (i.e., integers or characters). Second, in the ontology layer, the similarity between two concepts is based on the ontological hierarchy and semantic relations represented by the ontology. Third, in the context layer, the comparison of the concepts considers the external context where they are involved.

In this thesis, the semantic similarity measures between entities are effectively computed through the comparison of their common characteristics. We use similarity measures to generate relevant path by taking advantage of the structure of the RDF graph. In the context of our problem, a similarity measure s: $G_B \times G_B \rightarrow \mathbb{R}$ assigns a similarity score to each entity pair, which indicates the semantic closeness between the entities.

In the following, we briefly describe the entity similarity measures that will be used by the generic search in Chapter 5. The similarity measures compute the entity relevance in a pathfinding process, independently of the domain. We use well-known similarity measures, such as the *Jaccard distance*, the *Wikipedia link-based measure* (MILNE *et al.*, 2008) and *SimRank* (JEH *et al.*, 2002), to estimate

the similarity between entities by computing their common characteristics (Jaccard distance), shared links (Wikipedia link-based measure) and common neighbors (SimRank).

Let *B* be a knowledge base and $G_B = (N_B, E_B, e_B)$ be the *RDF graph* of *B*. The entities in G_B are described by sets of property values and relationships with other entities. For entities *a* and *c*, two abstract walkers are deployed to traverse G_B to a specific depth *b* and collect the sets of characteristics (properties and neighbors), A_b and C_b, of *a* and *c* up to depth *b*, respectively. We use $A_b[i]$ to denote the i^{th} element of A_b (and likewise for C_b). The following definitions will be use such sets.

The *Jaccard distance* is a simple measure that considers the common characteristics between two sets to compute their similarity. The *Jaccard distance* between two entities *a* and *c* is defined as the cardinality of the intersection of the set of characteristics divided by the cardinality of their union:

$$Jaccard(a,c) = \frac{|A_b \cap C_b|}{|A_b \cup C_b|} \quad \text{, if } A_b \cup C_b \neq \emptyset$$

$$Jaccard(a,c) = 1 \quad \text{, if } A_b \cup C_b = \emptyset$$

(where $|S|$ denotes the cardinality of a set S, as usual). The maximum score of the Jaccard similarity measure is "1", if the intersection of the set of characteristics is equal to their union.

The *Wikipedia link-based measure* (WLM), proposed by Milne and Witten (MILNE *et al*., 2008), was initially defined to compute the similarity between articles in Wikipedia. However, it can be adopted to measure the similarity between two entities *a* and *c* in G_B, and is defined as

$$WLM(a,c) = \frac{log(max(|A_b|,|C_b|)) - log(|A_b \cap C_b|)}{log(|N_B|) - log(min(|A_b|,|C_b|))}$$

where, we recall, N_B is the set of nodes of G_B (i.e., entities of B).

SimRank is a general link-based similarity measure, proposed by Jeh and Widom (JEH *et al*., 2002). SimRank is a domain independent similarity measure and is computed on a directed graph. This measure assumes that two objects are similar if they are referenced by similar objects. In other words, the similarity of two objects is based on the similarity of their neighbors. The SimRank score *s*,

collected at depth *b*, aggregates the similarity from neighboring pairs and is defined as

$$s(a,c) = \frac{D}{|A_b||C_b|}\sum_{i=1}^{|A_b|}\sum_{j=1}^{|C_b|} s(A_b[i], C_b[j])$$

where $D \in (0,1)$ is the decay factor. Note that the definition of SimRank is recursive and requires the definition of an entity similarity measure at depth *b*, the basis of the recursion.

We use the notion of *maximum common graph* (*mcs*) (BUNKE *et al*., 1998) for this purpose. Assume that an entity *u* is represented by a URI that, when dereferenced, retrieves a graph G_u that describes the entity (HARTIG *et al*. 2009). Then, given two entities *u* and *v*, we define:

$mcs(u, v) = G$ iff G is the maximum common graph of G_u and G_v

Finally, we define the basis of SimRank as:

$$s(u,v) = \frac{|mcs(G_u, G_v)|}{max(|G_u|, |G_v|)}$$

2.5. Path-Ranking Problem

In large RDF graphs, there can be many paths that connect two entities. We denote an RDF path that connects two entities a and c by $\pi(a \to c)$, and indicate the sequence of RDF triples as:

$$\pi(a \to c) = \left(a \xrightarrow{p_i} w_i\right); \left(w_i \xrightarrow{p_{i+1}} w_{i+1}\right); \ldots; \left(w_{k-1} \xrightarrow{p_{k-1}} w_k\right); \left(w_k \xrightarrow{p_k} c\right)$$

where a, c, w_i, are different nodes of the RDF path ($w_i \neq w_j$), k is the length of the RDF path and $0 \leq i \neq j \leq k$.

In this thesis, we use SPARQL queries and the HTTP protocol to define path-finding processes to find all RDF paths that connect two entities in an RDF graph, as illustrated in Figure 3. After getting the paths between the entties, we need to identify the best relevant path, a problem called *path-ranking*.

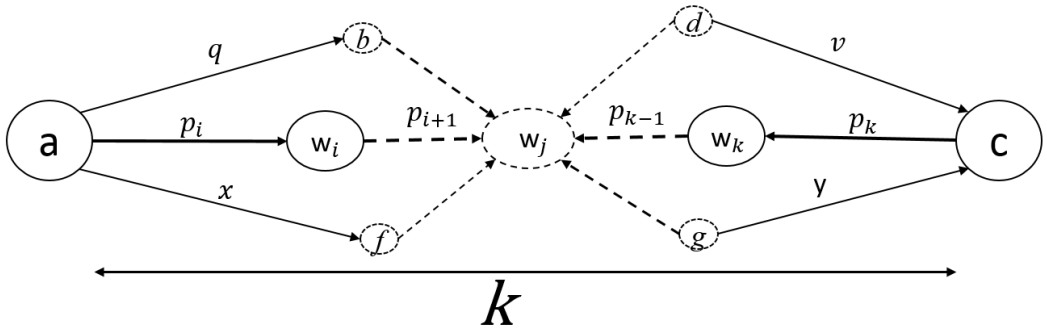

Figure 3: Overview of RDF paths between two entities.

Let *B* be a knowledge base and $G_B = (N_B, E_B, e_B)$ be the *RDF graph* of *B*. Let T_B be the set of all paths in G_B.

A relationship path ranking measure considers a path as sequence of properties and nodes, analyses each component in the path, and generates a score. More precisely, given a path π between two entities *a* and *c* in an RDF graph G_B, a *relationship path ranking measure* function $r: T_B \rightarrow \mathbb{R}$ that assigns a score to each path $\pi \in T_B$ between two nodes of G_B. It generates an ordered list of paths, as shown in Figure 4. In this list, $r(\pi_i) > r(\pi_{i+1})$ and the path in the first position is the most relevant. Given that the size of the list is *N*, the top-*n* paths are those that better describe the connectivity of the two entities, and $n \leq N$.

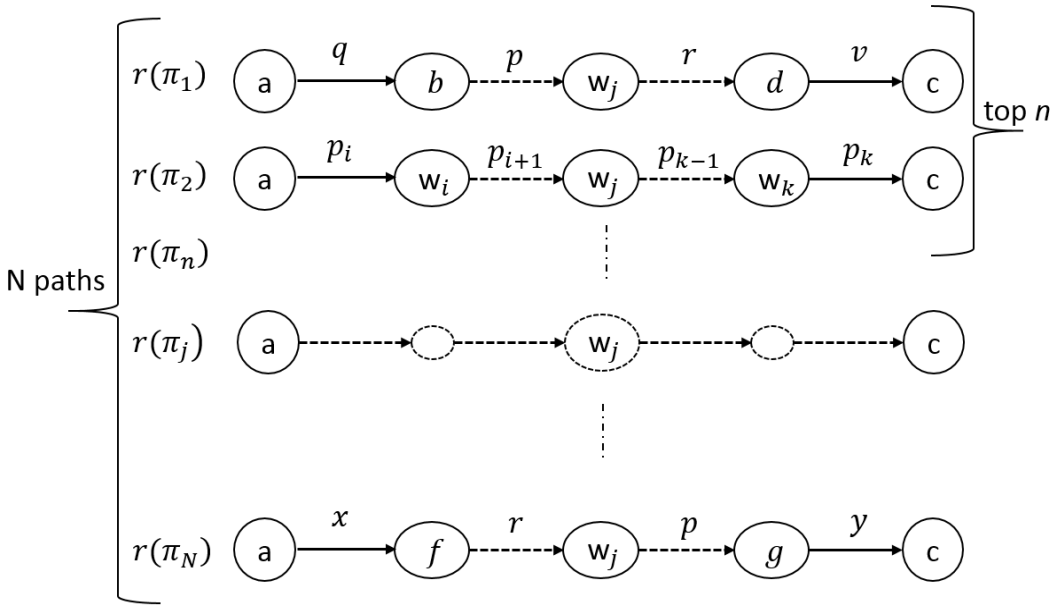

Figure 4: Path ranking of two entities.

In what follows, we briefly describe the path-ranking measures that will be used in Chapter 5. Although, there are many approaches to rank the relationship paths between entity pairs, the following approaches were selected since they consider the semantic of the relationships that link the entities.

The *predicate frequency inverse triple frequency* (PF-ITF), proposed in (PIRRÒ, 2015), is an adaptation of the original TF-IDF used in information retrieval. This measure uses as basic unit the RDF triples. The PF-ITF considers the participation of a property p in all triples in *B*, and uses different factors to sort relationship paths.

Let *p* be a property and *w* be an entity in *B*. We use the following notation (we leave B as an implicit argument for simplicity):

- $(*\xrightarrow{p} w)$ denotes the set of all triples of the form (*x*, *p*, *w*) in *B*; we say that *p* is *incoming to w* in G_B
- $(w \xrightarrow{p} *)$ denotes the set of all triples of the form (*w*, *p*, *z*) in *B*; we say that *p* is *outgoing from w* in G_B
- $(*\xrightarrow{p}*)$ is the set of triples of the form (*x*, *p*, *z*) in *B*
- $(*\rightarrow w)$ denotes the set of all triples of the form (*x*, *y*, *w*) in *B*
- $(w \rightarrow *)$ denotes the set of all triples of the form (*w*, *y*, *z*) in *B*

We then define:

- $pf_i^w(p, G_B)$ is the frequency of *p* incoming to *w* in G_B
- $pf_o^w(p, G_B)$ is the frequency of *p* outgoing from *w* in G_B
- $itf(p, G_B)$ is the *inverse triple frequency* of *p* in G_B
- $pfitf_x^w(p, G)$ is the *predicate frequency inverse triple frequency* of p incoming (for *x*="*i*") to *w* in G_B, or outgoing (for *x*="*o*") from *w* in G_B

These measures are defined as

$$pf_i^w(p, G_B) = \frac{|(*\xrightarrow{p} w)|}{|(*\rightarrow w)|} \quad (i) \qquad pf_o^w(p, G_B) = \frac{|(w \xrightarrow{p} *)|}{|(w \rightarrow *)|} \quad (ii)$$

$$itf(p, G_B) = \log \frac{|B|}{\left|(*\xrightarrow{p}*)\right|} \quad (iii) \qquad pfitf_x^w(p, G_B) = pf_x^w(p, G_B) \times itf(p, G_B) \quad (iv)$$

where $|S|$ denotes the cardinality of a set S, as usual.

The score of a path $\pi = (a, p_1, w_1, p_2, w_2, \ldots, p_{k-1}, w_{k-1}, p_k, c)$ is then defined as

$$score(\pi(a,c), G_B) = \frac{[pfitf_i^a(p_1, G_B) + pfitf_o^c(p_1, G_B)]}{2} \; ; \; k = 1$$

$$score(\pi(a,c), G_B) = \frac{score(\pi a, w_1), G) + \cdots + score(\pi(w_{k-1}, c), G_B)}{k} \; ; \quad k > 1$$

The *exclusivity-based relatedness*, proposed in (HULPUS *et al*., 2015), claims that a relation between two concepts is stronger if each of the concepts is related through the same type of relation to fewer other concepts. This property is called relations exclusivity in (HULPUS *et al*., 2015) and is described as follows:

Given a simple relationship "e", denoted as $a \xrightarrow{p} c$, between two entities a and c, where "p" is the type of the relationship, the exclusivity of the relationship "e" is defined as the probability that if we randomly select an edge "e'" outgoing of the set of all edges of type "p" that exit node "a" and all relationships of type "p" incoming to a node "c", that relationship "e'" is relationship "e".

Using the notation introduced to define PF-ITF, the *exclusivity* of a relationship "$a \xrightarrow{p} c$" is defined as (B is an implicit argument):

$$exclusivity\left(a \xrightarrow{p} c\right) = \frac{1}{\left|a \xrightarrow{p} *\right| + \left|* \xrightarrow{p} c\right| - 1} \; ; k = 1$$

The score of a path $\pi = (a, p_1, w_1, p_2, w_2, \ldots, p_{k-1}, w_{k-1}, p_k, c)$ is then defined as

$$score\left(_{score(\pi(a,\, c),\, G_B)}\right) = \frac{1}{\sum_i 1/exclusivity\left(w_i \xrightarrow{p_i} w_{i+1}\right)} \; ; \quad 1 > i \geq k$$

The *Pointwise Mutual Information* (PMI) (CHURCH *et al.*, 1990) measures the co-occurrence strength between two items. It works by relating the probabilities of the individual occurrence of the items to the probability of both items occurring together. We estimate the PMI score of a relationship path based on the co-occurrence of the properties and entities in the relationship path. We consider three cases in the computation of the relevance of a relationship path:

(i) co-occurrence of two properties.

(ii) co-occurrence of a property and an entity, where the property is outgoing from the entity.

(iii) co-occurrence of a property and an entity, where the property is incoming to the entity.

We formalize these cases as follows (G_B is an implicit argument):

$$PMI(p_r, p_s) = \log\left(\frac{f(p_r, p_s)}{f(p_r) * f(p_s)}\right) \qquad (i)$$

$$PMI(w, p) = \log\left(\frac{f(w, p)}{f(w) * f(p)}\right) \qquad (ii)$$

$$PMI(p, w) = \log\left(\frac{f(p, w)}{f(p) * f(w)}\right) \qquad (iii)$$

where $f(.,.)$ is the frequency that two items co-occur in G and $f(.)$ is the frequency of a property or entity in G.

The relevance of a path $\pi = (a, p_1, w_1, p_2, w_2, \ldots, p_{k-1}, w_{k-1}, p_k, c)$ is then defined as:

$$score(\pi, G_B) = median\{\, PMI(p_i, p_j) \,/\, 1 \le i \ne j \le k \,\} + 1/2k * (PMI(a, p_1) + \ldots + PMI(w_{k-1}, p_k) + PMI(p_1, w_1) + \ldots + PMI(p_k, c))$$

3
Related Work

To facilitate the reading and understanding, this chapter groups related work into three topics: explaining relationships in knowledge bases; evaluating the relatedness between entities and relationship-path ranking; and, finally, entity recommendation in knowledge bases.

3.1.
Explaining Relationships in Knowledge Bases

In this section, we describe research that explores and explains the connectivity between entities using paths or subgraphs, especially measures that lead to interesting explanations, such as node centrality, node frequency or edge informativeness applied to the paths retrieved from knowledge bases.

REX, introduced in (FANG *et al.*, 2011), examines the relationships between objects in knowledge bases to explain the connections between entity pairs. REX computes relationship explanations in the form of instances of informative graph patterns (edge-labeled templates) that connect two entities. REX generates a ranked list of relationship explanations using interestingness measures. REX assumes that all relationship paths are stored in a local database. Usually, a knowledge base, such as DBpedia or Wikidata, stores data about millions of entities and rarely a user will have sufficient computing resources to analyze such large amount of data. Another drawback of REX is that it covers only in one specific domain.

The work proposed in (PIRRÒ, 2015), based on the algorithm of (FANG *et al.*, 2011), introduced RECAP to overcome some limitations of REX. RECAP is a tool specialized in RDF graphs, and it uses SPARQL queries to generate subgraphs/paths to explain how entities are related. RECAP has the advantage of not requiring any data preprocessing, and obtains information by querying (remote) SPARQL endpoints. RECAP ranks paths using informative and diversity measures, and combines the top-*k* paths to create subgraph. The most informative patterns are

used to find entity pairs related to the target entity pair. The main problem of RECAP is the complexity of the algorithm used to generate the explanation patterns, as reported in (FANG *et al.*, 2011).

In the work of (CHENG, *et al.* 2014), the authors propose EXPLASS. This tool uses entity classes and property labels to create descriptors and to facilitate the exploration of the explanations discovered. EXPLASS provides a flat list (top-*k*) clusters and facet values for refocusing and refining a search. The approach detects clusters by running pattern matches on the datasets to compute frequent, informative and small overlapping patterns. As RECAP, EXPLASS also incurs in an expensive sorting of paths and explanation patterns.

The work of (HEIM *et al.*, 2009) proposes RelFinder, which explores the existing relations between two or more DBpedia entities. These relations can be simple or can include more complex paths. In Relfinder, the visualization of relations can be filtered, thanks to a faceted menu, according to their length, the entity types and property names. RelFinder, however, has no means to rank paths and does not provide dedicated methods to compare paths.

Pathfinding techniques have also been used to identify entity relationships, as the works proposed by (FANG *et al.*, 2011; DE VOCHT *et al.*, 2013). The last works use activation functions based on the Linked Data graph structure to drive informed searches (e.g. A*, random walks, node centrality), prioritizing nodes and pruning the search space. Their major limitation consist in exploiting Linked Data with an a priori knowledge, either by indexing and pre-processing datasets. However, even with such constraint, these techniques overcome the dependency of a SPARQL endpoint.

In general, some approaches explained in this section use ranking to exclude information, with the side effect that the user cannot infer the hidden meaning of each explanation, and thus he still has to decide which explanation to explore. Differently from these works, we introduce in Chapter 4 a connectivity profile for a pair of entities that uses the class hierarchy and the topic categories of the underlying knowledge base, summarizes a large set of explanations, retaining only the most representative ones (without losing relevant information), and classifies each explanation using topic categories.

In this thesis, considering the lessons learned using search strategies based on SPARQL queries, as (PIRRÒ, 2015; CHENG, *et al.* 2014; HEIM *et al*., 2009), in Chapter 5, we study and compare different approaches to find relevant paths, using a generic search strategy based on the backward search heuristic (LE *et al.*, 2014). We propose an online pathfinder process, using HTTP requests to recover the entity description. The activation criterion of the search uses similarity measure and node centrality to prioritize certain paths over others.

3.2. Evaluating the Relatedness in a Knowledge Base

Currently, there is no way to measure the effectiveness of the search strategies to explore the connectivity between entities automatically. The approaches available in the literature (CHENG *et al*., 2013; PIRRÒ, 2015; NUNES *et al.*; 2013; HULPUS *et al*., 2015) evaluate the effective of the approach with the help of users, and the comparison methods do not clearly define the capabilities of the approaches analyzed.

The work of (CHENG *et al*., 2013) proposed a methodology to evaluate approaches focused on the exploration of the connectivity of entities. The methodology proposed consisted in giving exploration tasks to users, and then asking users if the approaches had the capability to solve the assigned task. The users answered a questionnaire, through which the approaches compared are judged. The work of (PIRRÒ, 2015) uses the same evaluation criteria that (CHENG *et al*., 2013) and it compares different exploration approaches.

Works such as (NUNES *et al.*, 2013; HULPUS *et al*., 2015) focused on measuring the connectivity between entities. The evaluation process of these approaches were based on the peoples' preferences. In (NUNES *et al.*, 2013), a crowdsourcing platform was used to ensure the quality of the results. The user passes through a set of tests, where correct answers are already known. This process allows filtering out poor assessors. Hence, relevance judgements from untrusted users can be avoided. The work of (HULPUS *et al*., 2015) focuses on word and entity disambiguation and uses ground-truth datasets based on human assessed scores.

The work proposed by (DE VOCHT *et al.*, 2016) exposes that entity similarity heuristics increase the relevance of the links between nodes. The heuristics used for that work focused on the centrality of the nodes in a path, and do not use the semantic information of the relationships. In this last work, the authors compare different search strategies and though user judgments measure the effectiveness of the strategies.

In this thesis, Chapter 5 introduces a ground truth of entity pairs in two entertainment domains to automatically compare search strategies that explore the connectivity between entities. We use information from the Internet Movie Database (IMDb)[2] and last.fm[3] to generate specialized relationship-path rankings, for each entity pair of our ground truth in the movies and music domains, respectively.

3.3. Entity Recommendations

PageRank (BRIN *et al.*, 1998) and HITS (KLEINBERG *et al*., 1998) are prominent algorithms for ranking Web resources based on link analysis. The random surfing model underlying the PageRank algorithm has been widely accepted as the navigation model for the Web. This model is essentially a simple random walk modeled by a Markov chain. Based on this surfing model, the basic PageRank algorithm computes the rank for each Web resource by iteratively propagating the rank until convergence. Independent of (BRIN *et al.*, 1998), (KLEINBERG *et al*., 1998) proposed HITS. In this last work, the authors argued that it is not necessary that good authorities point to other good authorities. Instead, there are special nodes that act as hubs that contain collections of links to good authorities.

RDF graphs are structures to which one can apply classical strategies as PageRank and HITS. In this way, (BALMIN *et al*., 2004) proposed ObjectRank. ObjectRank is an adaptation of PageRank, where each link is represented by a particular weight. ObjectRank includes a concept called *authority transfer schema graphs*, which allows propagating the semantic through different type of links.

[2] http://www.imdb.com

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

Likewise, TripleRank (FRANZ *et al.*, 2009) introduces a tensor-based approach for ranking RDF triples. TripleRank takes knowledge about different links type, and can be seen as a multi-model counterpart to Web authority ranking with HITS. Because of the expressiveness of a tensor decomposition, TripleRank does not scale very well and only evaluates small graphs.

Research that investigate recommender systems applied to Linked Data includes (PASSANT, 2010; NGUYEN *et al.*, 2015; PIAO *et al.*, 2015). Such references compute the similarity between entities and assume that the properties of the entities have a relevance for a specific domain. The work of (PASSANT, 2010) used a music recommendation system based on DBpedia to offer recommendations for bands and solo artists. The work of (PASSANT, 2010) computed the similarity between entities using direct links and specialized relationship in the music domain. In the work of (NGUYEN *et al.*, 2015) different similarity measures in the music domain, using DBpedia and Freebase, are compared. In general, the measures used by (NGUYEN *et al.*, 2015) compute the relevance of the characteristics that describe the entities using a set of properties pre-selected in the music domain. Lastly, the work of (PIAO *et al.*, 2015) extended the research from (PASSANT, 2010) by providing a metric that computes the similarity between the properties that describe an entity.

In this thesis, Chapter 5 details search strategies to identify relevant paths between entities that use similarity measures to prioritize certain paths over others, without any domain restrictions, and path-ranking approaches to determine the relevance of the relationships paths that link two entities.

4
Profiling the Connectivity of Entity Pairs

4.1.
Introduction

This chapter explores knowledge bases to discover how two entities are related. For example, it is now (but probably not in 20 years from now) common knowledge that “Barack Obama” and “Michelle Obama” are related in at least two ways: they are married and they are both alumni of Harvard Law School. It is not so obvious, though, to discover how “Albert Einstein” and “Kurt Gödel” are related. By exploring DBpedia, for example, one may find that:

a) “Albert Einstein” was influenced by “Ernst Mach”, which was influenced by “David Hume”, which influenced “Edmund Hussert”, which influenced “Kurt Gödel”.

b) “Albert Einstein” was influenced by “Baruch Spinoza”, which was influenced by “Giordano Bruno”, which influenced “Gottfried W. Leibniz”, which influenced “Kurt Gödel”.

These two relationship paths can be abstracted by saying that

c) X was influenced by a philosopher, which was influenced by a philosopher, which influenced a philosopher, which influenced Y.

with X instantiated by “Albert Einstein” and Y by “Kurt Gödel”. Note that (c) provides a concise explanation of the connectedness (or relationship) between “Albert Einstein” and “Kurt Gödel” in the Philosophy domain.

When two entities are directly related, such as “married to”, or related by a short composition of relationships, such as “coauthor”, that have a natural language term to denote the composition, it is not too difficult to explain how the two entities are related. However, this is not the case with the example relating “Albert Einstein” and “Kurt Gödel” described in the previous paragraph. To address this issue, we introduce the concept of connectivity profile for a pair of entities, defined as a

concise explanation about how the entities are related in a knowledge base, and rephrase the goal of this work as:

- "Given a knowledge base and an entity pair, create a *connectivity profile* for the entity pair".

Several tools were developed (HEIM *et al*., 2009; FANG *et al*., 2011; PIRRÒ, 2015; CHENG *et al*., 2014; MOHAN *et al*., 2014) to explore knowledge bases and generate connectivity profiles. In general, such tools: (1) search for relationship paths between an entity pair– the larger the number of paths found, the stronger the connectivity between the entities is likely to be; (2) group the paths found; (3) summarize the groups of paths to present a connectivity profile of the entity pairs to the user. However, this simple strategy poses three challenges:

(1) How to find relationship paths between a given entity pair in a knowledge base?
(2) How to group the relationship paths in a meaningful way?
(3) What characteristics of the groups of paths must be selected to generate a connectivity profile?

In this chapter, we first describe a strategy to generate connectivity profiles for entity pairs represented in a knowledge base, organized as an RDF graph. To address the first challenge, the strategy resorts to specialized queries over the RDF graph to identify RDF paths that connect the given pair of entities. As for the second challenge, it adopts a strategy based on semantic annotations, as in (MEI *et al*., 2006), which uses similarity metrics to group annotations about an itemset and summarize the observed groups. The major difference lies in that our strategy works over sets of RDF paths rather than itemsets. The strategy covers the last challenge by resorting to the topic categories assigned to the entities and to the relationships between the entities found in the RDF paths. After defining the strategy, we detail the DBpedia Profiler tool, which implements the proposed strategy for DBpedia and adopts path streams (BARBIERI, *et al*., 2010) to collect and analyze the RDF paths. Finally, we compare DBpedia Profiler with RECAP (PIRRÒ, 2015), using DBpedia.

The contributions in this chapter can be summarized as: (1) definition of a strategy to generate connectivity profiles for entity pairs; (2) description of the

DBpedia profiler tool, which implements the strategy over DBpedia; (3) presentation of a comparison between DBpedia profiler and the state-of-the art tool RECAP.

The rest of this chapter is structured as follows. Section 4.2 provides an overview of the strategy. Section 4.3 describes the DBpedia profiler tool in detail. Section 4.4 compares DBpedia profiler with our implementation of RECAP. Finally, Section 4.5 presents the conclusions.

4.2. A Strategy to Generate Connectivity Profiles

4.2.1. An Example of a Connectivity Profile

We define a *connectivity profile* for an entity pair a and c as a tuple consisting of: (1) the *immediate classes* of a and c; (2) a *connectivity score* for a and c; (3) a set of *representative semantic explanation patterns*; (4) a set of *topic categories*; and (5) a set of *similar entity pairs*. The following example just illustrates a connectivity profile, without intending to indicate how it is computed.

Example 1. Figure 5 shows a connectivity profile for "Albert Einstein" and "Kurt Gödel", indicating: (1) the immediate classes of the entities, which are both dbo:Scientist ; (2) a connectivity score of 0.98; (3) three representative semantic explanation patterns; (4) the topic categories of the explanations: "Humanities", "Society", "Science" and "Geography"; (5) a set of similar entities pairs: "Niels Bohr" and "Aage Bohr", etc. Indeed, we observe that the pairs ("Albert Einstein", "Kurt Gödel") and ("Niels Bohr", "Aage Bohr") have in common the facts that they were awarded science prizes, died in the same place and were influenced by the same group of people. □

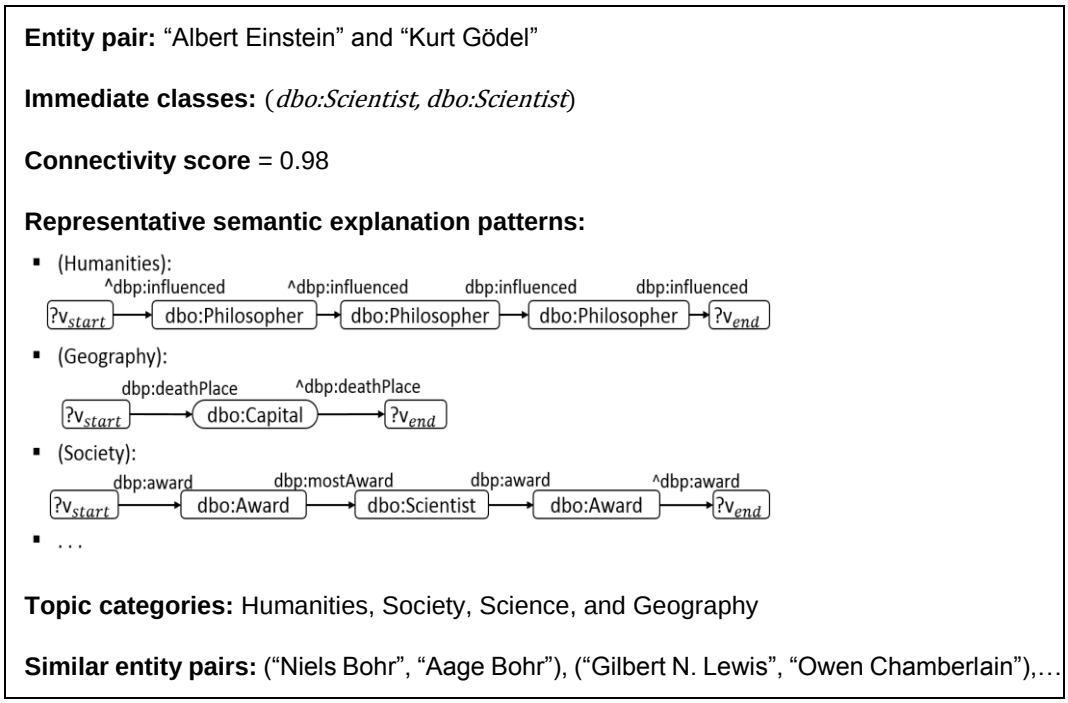

Figure 5: A schematic view of the connectivity profile of Albert Einstein and Kurt Gödel.

4.2.2. Exploring the Class Hierarchy of a Knowledge Base

In this and the next sections, let B be a knowledge base, $G_B=(N_B, E_B, e_B)$ be the RDF graph induced by B, EN_B be the set of entities of B, C_B be the set of classes defined in B, $H_B=(C_B, S_B)$ be the class hierarchy of B and $\mathcal{W}$ be a set of *topic categories*.

We use the class hierarchy of B and the set of topic categories to define some of the components of connectivity profiles. We start by defining the *entity class score* $ecs: EN_B \times C_B \rightarrow \mathbb{R}$ which assigns to each entity e in EN_B and each class C in C_B a *score* $ecs(e, C)$.

An *immediate class* of an entity e – the first component of a connectivity profile – is a class C that maximizes $ecs(e,C)$, which we note may not be unique. Defining the immediate class of an entity e is not as trivial as it seems. Indeed, some methods (e.g., CHENG *et al.*, 2014; MOHAN *et al.*, 2014) simply assume that each entity has only one class, whereas more sophisticated methods (MENG *et al.*, 2015; ASSIS *et al.*, 2015; BÖHM *et al.*, 2012) fully face the problem.

Example 2. Figures 6 (a) and 6(b) show the classes and the class hierarchy for the entity dbp:David_Hume, in DBpedia. Therefore, the entity class score assigns a value for dbo:Scientist, dbo:Writer, dbo:Economist, dbo:Historian, dbo:Philosopher, dbo:Person, dbo:Agent and owl:Thing, which would indicate their relevance to describe dbp:David_Hume. Using the entity class score, dbo:Philosopher has the highest score among all classes of dbp:David_Hume and therefore it is the immediate class. □

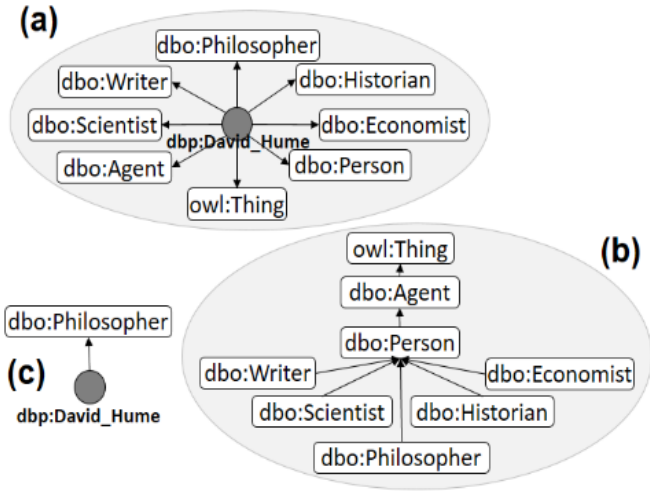

Figure 6: Entity dbp:David_Hume: (a) Classes; (b) Class Hierarchy; (c) Immediate Class.

In order to group the entities and generate semantic explanations patterns, we introduce two additional concepts. Let E be a set of entities and IC be the set of their immediate classes. We say that u is a *lowest superclass of IC* iff all classes in IC are subclasses of u and there is no subclass v of u such that all classes in IC are subclasses of v. Finally, we say that $\bar{u}$ is an *immediate class of E* iff $\bar{u}$ is a lowest superclass of IC that maximizes $\sum_{e \in E} ecs(e, \bar{u})$.

We define a *weighted class topic category function* $wcat: C_B \times \mathcal{W} \rightarrow \mathbb{R}$ that maps each class $C \in C_B$ and each topic category $t \in \mathcal{W}$ into a real number.

Based on this function, we define the *class topic category set function* $scat: C_B \rightarrow 2^{\mathcal{W}}$ such that $scat(C) = \{t \in \mathcal{W} / \ wcat(C, t) > \tau_t\}$, where τ_t is a *topic category threshold*.

Example 3. Figures 7(a) and 7(c) indicate the classes of dbp:David_Hume and dbp:Giordano_Bruno. The classes are assigned an entity class score. If the set entities does not share the same immediate class then it is necessary to find the lowest super class of the set, as in Figure 7(b). The entities grouped by the lowest superclass are used to compute the topic categories. Figure 7(d) presents the top 3 topic categories of dbo:Philosopher. □

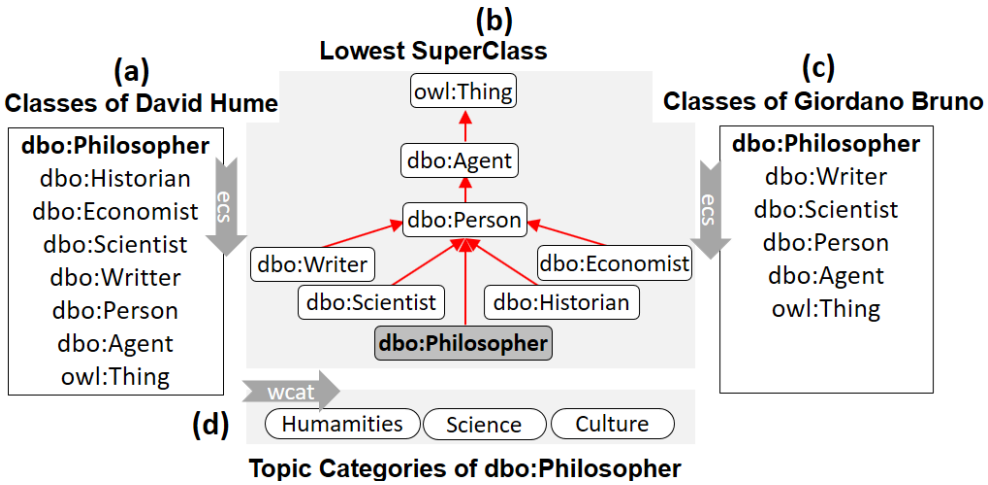

Figure 7: (a) Classes of David Hume; (c) Classes Giordano Bruno; (b) Lowest Superclass; (d) Topic Categories.

4.2.3. Exploring the RDF graph of a Knowledge Base

In this section, we explore the RDF graph to define the rest of the components of connectivity profiles. Let a and c be two entities in EN_B and k be a positive integer.

A *connectivity score* for B is a function $cs: EN_B \times EN_B \rightarrow \mathbb{R}$ that measures how a pair of entities a and c in EN_B are related in B; $cs(a, c)$ is called the *connectivity score of* a and c.

We focus on the construction of representative semantic explanation patterns that capture the connectivity of a and c, which may be a challenge, if there are many paths connecting a and c. For example, in DBpedia dbp:Albert_Einstein and dbp:Kurt_Gödel have more than 500 paths between them. We adopt a three-step strategy to address this challenge.

We first compute the set $P[a, c]$ of all paths of B that start on a and end on c with maximum length k. Then, we partition $P[a, c]$ in such a way that all paths in the same partition have exactly the same sequence of labels (i.e., properties or their inverse).

Let $\{P_1,...,P_n\}$ be a partition of $P[a, c]$ and assume that the sequence of labels of the paths in P_i is $l_{i_1}, \dots, l_{i_{k_i}}$. The *explanation pattern* EP_i for P_i of length k is then defined as the basic graph pattern of the form:

$$?v_{start}\ l_{i_1}\ ?o_1$$
$$?o_1\ l_{i_2}\ ?o_2$$
$$\dots$$
$$?o_{k-1}\ l_{i_{k_i}}\ ?v_{end}$$

where $?v_{start}$ and $?v_{end}$ are two special variables.

We use the classes of the entities involved in each path in P_i to extend EP_i into a *semantic explanation pattern* SEP_i as follows.

Given a path of length k (with $k+1$ nodes), we say that the j^{th} *entity* of the path is the j^{th} node of the path. Let $EN_{i,j}$ be the set of the j^{th} entities of the paths in EP_i and let $IC_{i,j}$ be the set of the immediate classes of the entities in $EN_{i,j}$, for $j=2,...,k$ (that is, we exclude the first and the last entities in a path, which are always bound to the given pair of entities). The *semantic explanation pattern* SEP_i for P_i of length k is defined as the basic graph pattern SEP_i with two parts, EP_i and TP_i, where

- EP_i is the explanation pattern for P_i of length k, as defined above
- for $j=2,...,k$, TP_i contains a triple pattern of the form "$?s_j$ **rdf:type** $b_{i,j}$", where "$?s_j$" is the subject of the j[th] triple pattern in EP_i and $b_{i,j}$ is an immediate class of $EN_{i,j}$ (if there is more than one, we arbitrarily select one of them)

Example 4. Consider paths p_1 and p_2 in DBpedia between dbp:Albert_Einstein and dbp:Kurt_Gödel shown in Figure 8(a) and 8(b), Figure 8(d) depicts the following semantic explanation pattern:

$?v_{start}$ **^dbp:influenced** $?o_1$.
$?o_1$ **^dbp:influenced** $?o_2$.
$?o_2$ **dbp:influenced** $?o_3$.
$?o_3$ **dbp:influenced** $?v_{end}$.
$?o_1$ **rdf:type** dbo:Philosopher .
$?o_2$ **rdf:type** dbo:Philosopher .
$?o_3$ **rdf:type** dbo:Philosopher .

For example, Figure 8(e) and 8(f) respectively show that dbo:Philosopher is the immediate class of dbp:David_Hume and also of dbp:Giordano_Bruno. In fact, dbo:Philosopher is an immediate class of the set {dbp:David_Hume, dbp:Giordano_Bruno} (class scores are not indicated in the figure). □

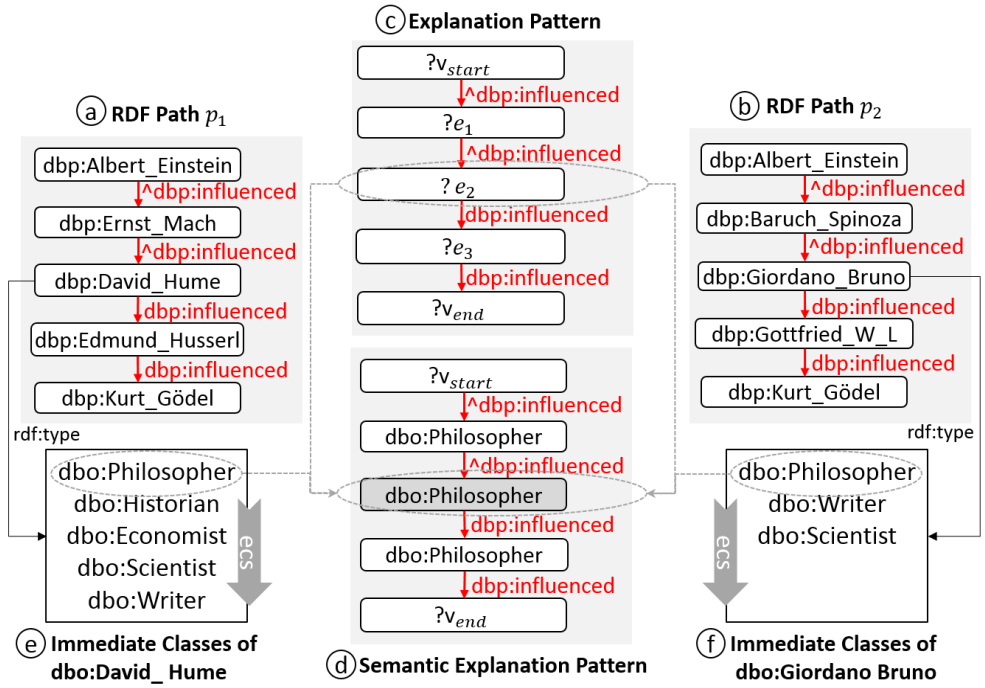

Figure 8: Generating a semantic explanation pattern between Albert Einstein and Kurt Gödel.

Even after the above process, the set of semantic explanation patterns may still be very large and redundant. Indeed, the process of finding explanations patterns followed by (FANG *et al.*, 2011; PIRRÒ, 2015;) uses a combinatorial strategy, which may lead to redundancies, and the processes described in (MOHAN *et al.*, 2014; MENG *et al.*, 2015) do not treat the redundant explanations. To face this problem, the last step of our process: filters semantic explanation patterns

which are not coherent; clusterizes the remaining semantic explanation patterns by similarity and, for each cluster, selects a representative semantic explanation pattern.

Let $\mathcal{SEP}$ be the set of semantic explanation patterns. Intuitively, a semantic explanation pattern *SEP* is *coherent* iff it satisfies the following two conditions: (i) the properties in SEP frequently co-occur in the knowledge graph; (ii) the classes in SEP have similar topic categories. At this point, we abstract the details by defining a *coherence measure* $coh : \mathcal{SEP} \rightarrow \mathbb{R}$ that maps each semantic explanation pattern into a real number. We say that SEP *is coherent* iff $coh(SEP) > \tau_c$, where τ_c is a given *coherence threshold*.

To cluster semantic explanation patterns, we introduce a *similarity metric* $sim : \mathcal{SEP} \times \mathcal{SEP} \rightarrow [0,1]$ that maps two semantic explanation patterns in $\mathcal{SEP}$ into a real number in the interval [0,1].

Let $P[a, c]$ again be the set of all paths of B that start on a and end on c with maximum length k, $\{P_1,\ldots,P_n\}$ be the partition of $P[a, c]$ based on label sequence and SEP_i be the semantic explanation pattern for P_i, as defined above, for $i = 1,...,n$. We eliminate all semantic explanation patterns which are not coherent, clusterize the remaining set, using the similarity measure, and select the medoid of each cluster. These medoids, called *representative semantic explanation patterns*, form the final set $EXP[a, c]$ of explanations for the connectivity (of length k) of a and c.

The last two components of a connectivity profile are defined as follows. Using the weighted class topic category set function $wcat : C_B \times \mathcal{W} \rightarrow \mathbb{R}$, we introduce the *pattern set topic category set function* $pcat : 2^{\mathcal{SEP}} \rightarrow 2^{\mathcal{W}}$ that maps a set of semantic explanation patterns into a set of topic categories in $\mathcal{W}$ and is defined as

$$pcat(P) = \{\, t \in \mathcal{W} \;/\; (\exists\, SEP \in P)(\exists C \in cl(SEP))(wcat(C,t) > \tau_t) \,\}$$

where $cl(SEP)$ is the set of classes that occur in the semantic explanation pattern SEP.

Given the set $EXP[a, c]$ of representative semantic explanation patterns for a and c, the topic categories in $pcat(EXP[a, c])$ are called the *topic categories* for a and c.

Finally, we find entity pairs which are similar to the given entity pair. Let SEP be a semantic explanation pattern in $EXP[a, c]$. The SPARQL query Q_p *induced by* SEP is the query of the form

$$\text{SELECT } ?v_{start}, ?v_{end} \text{ WHERE } SEP$$

An entity pair *is similar to* a *and* c *according to* $EXP[a, c]$ iff the pair is returned by a query induced by a semantic explanation pattern in $EXP[a, c]$.

4.2.4. Definition of the Connectivity Profile

We finally define a *connectivity profile* for a pair of entities a and c in B, with maximum distance k, as a tuple $\mathbb{C} = ((C_a, C_c), s, EXP[a, c], T, E)$, where C_a and C_c are the immediate classes of a and c, $s=cs(a, c)$ is the connectivity score of a and c, $EXP[a, c]$ is a set of representative semantic explanations patterns (of length at most k), T is the set of top weighted topic categories for a and c and E is a set of entity pairs similar to a and c.

To summarize, the construction of connectivity profiles depends on:

- Providing concrete implementations for the following functions:
  - the entity class score $ecs: EN_B \times C_B \rightarrow \mathbb{R}$
  - the *weighted topic category function* $wcat: C_B \times \mathcal{W} \rightarrow \mathbb{R}$
  - the *connectivity score* $cs: EN_B \times EN_B \rightarrow \mathbb{R}$
  - the *coherence measure* $coh: \mathcal{SEP} \rightarrow \mathbb{R}$
  - the *similarity measure* $sim: \mathcal{SEP} \times \mathcal{SEP} \rightarrow \mathbb{R}$
- Estimating two thresholds:
  - a *coherence threshold* τ_c
  - a *topic category threshold* τ_t
- Successively computing, for a pair of entities a and c:
  - RDF paths from a to c of length k
  - explanation patterns for a and c of length k

- semantic explanation patterns for a and c of length k
- representative semantic explanation patterns for a and c of length k

4.3. DBpedia Profiler Tool

DBpedia profiler implements the strategy described in Section 4.2. Therefore, in what follows, the knowledge base B is DBpedia, $G_B=(N_B, E_B, e_B)$ is the RDF graph corresponding to DBpedia, C_B is the set of classes defined in DBpedia, $H_B = (C_B,S_B)$ is the DBpedia class hierarchy and $\mathcal{W}$ is the set of 23 top-level categories of Wikipedia (KAWASE *et al*., 2014).

4.3.1. Path-stream approach

Figure 9 depicts the overall structure of the DBpedia profiler tool. Briefly, at the application layer, the user provides a pair of entities and interacts with the tool to browse the connectivity profile generated for the pair of entities.

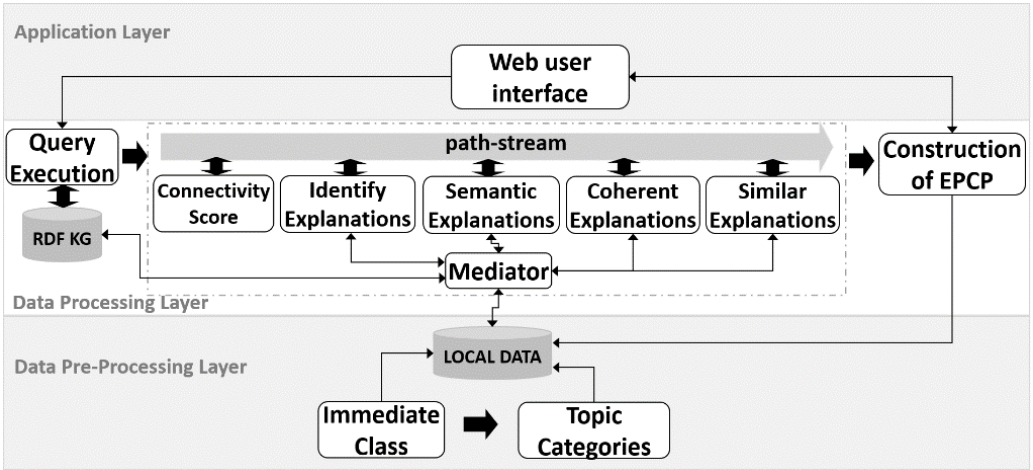

Figure 9: The DBpedia Profiler tool.

At the data processing layer, DBpedia profiler adopts a pipeline process to generate a connectivity profile for the given pair of entities (EPCP). First, it issues SPARQL queries to the DBpedia SPARQL endpoint to retrieve all RDF paths between the given pair of entities with a maximum predefined length, following the strategy defined in (HEIM *et al*., 2009). This process generates a stream of RDF

paths that is shared between the different operations that generate the profile. The operations in the data processing are independently applied to each RDF path. A mediator synchronizes the operation of the components and shares data between them, which also enables the parallel processing of the data flow.

At the data pre-processing layer, DBpedia profiler builds an index over the DBpedia class hierarchy to help identify the immediate classes of an entity (MENG *et al*., 2015; ASSIS *et al*., 2015; BÖHM *et al*., 2012) and map each of these classes into topic categories (KAWASE *et al*., 2014).

4.3.2. Exploring the Class Hierarchy of DBpedia

DBpedia profiler implements the entity class score $ecs: N_B \times C_B \rightarrow \mathbb{R}$ as

$$ecs(e, C) = distance(Root, C) * \frac{frequency(e, C)}{\log(frequency(C))}$$

where $distance(Root, C)$ computes the distance between a class C and the root class in H_B, $frequency(e, C)$ is the local count of a class C in the description text of e in B and $frequency(C)$ computes the overall count of C in H_B.

DBpedia profiler implements a weighted class topic category function based on the 23 top topic categories of Wikipedia as follows. We first recall that, in Wikipedia, articles are manually annotated with categories. Since each class C of DBpedia has a corresponding article A_C in Wikipedia (LEHMANN *et al*., 2013), we use the categories of A_C to relate C to the 23 main top categories of Wikipedia, as in (KAWASE *et al*., 2014).

In detail, the *weighted class topic category function* $wcat: C_B \times \mathcal{W} \rightarrow \mathbb{R}$ is defined as

$$wcat(C, t) = \sum_{w \in W(C)} weight(w, t)$$

where C is a class in the class hierarchy of DBpedia, t is a Wikipedia top category, $W(C)$ are the categories of the Wikipedia article associated with C and $weight$ measures the relation between a Wikipedia category w and a top category t and is defined as (KAWASE *et al*., 2014).

$$weight(w,t) = \frac{1}{popularity(t)} * \frac{1}{distance(w,t)}$$

where $popularity(t)$ indicates the popularity of a given main top topic category t and, $distance(t, w)$ is the distance of a category w to the main top category t.

4.3.3. Exploring the RDF graph of DBpedia

DBpedia profiler adopts the *semantic connectivity score* (*SCS*) (NUNES *et al*., 2013, 2014) as the connectivity score. SCS is a variation of the Katz index (KATZ, 1953), introduced to estimate the relatedness of actors in a social network, and is defined as

$$SCS(a,c) = \sum_{l=1}^{k} \beta^l * |L^{<l>}|$$

where $|L^{<l>}|$ is the number of RDF paths from a to c of length l and k is the maximum distance considered between a and c. The damping factor β is responsible for exponentially penalizing longer paths; the smaller this factor, the smaller the contribution of longer paths to the final score. The final score is normalized to fall in the interval [0,1].

To compute the set $P[a, c]$ of all paths of B that start on a and end on c with maximum length k, DBpedia profiler adopts a strategy similar to that of RelFinder (HEIM *et al*., 2009). Briefly, the tool issues SPARQL queries to the RDF graph of DBpedia to retrieve all paths that start on a and end on c and capture the length of each RDF path. Just as in (NUNES *et al*., 2013, 2014), DBpedia profiler allows the user to filter out entities that belong to certain classes from the RDF paths. The execution of these queries creates a path-stream processing, as shown in Figure 9.

DBpedia profiler computes the partition $\{P_1,\ldots,P_n\}$ of $P[a, c]$ and the set $\{SEP_1, \ldots, SEP_n\}$ of semantic explanation patterns for a and c of length k exactly as described in Section 4.2, using with the entity class score introduced in Section 4.3.2.

To compute the set of representative semantic explanation patterns, DBpedia profiler implements the coherence measure and the similarity metric for semantic explanation patterns as follows.

Recall that a semantic explanation pattern *SEP* for a given pair of entities is coherent iff it satisfies the following two conditions: (i) the properties in *SEP* frequently co-occur in the knowledge graph; (ii) the entity classes in *SEP* have similar topic categories. To satisfy the first condition, DBpedia profiler uses the Pointwise Mutual Information (PMI) of each property pair (CHURCH *et al*., 1990). PMI measures the co-occurrence strength between two items; it works by relating the probabilities of the individual occurrence of the items to the probability of both items occurring together. Based on the co-occurrence of the properties, we estimate the PMI score of each property pair in a semantic explanation pattern.

Let *SEP* be a semantic explanation pattern and r and s be two properties that occur in *SEP*. The PMI of r and s is defined as:

$$PMI(r,s) = \log\left(\frac{f(r,s)}{f(r) * f(s)}\right)$$

where $f(r,s)$ measures the co-occurrence frequency of r and s and $f(r)$ and $f(s)$, the individual frequency of r and s, as shown in Table 1.

Table 1: SPARQL Queries to compute the PMI.

f(*r*,*s*)	SELECT COUNT (DISTINCT *?e*) as *?count* WHERE { $?s_1$ **r** *?e* . *?e* **s** $?o_2$}
f(*r*)	SELECT COUNT (DISTINCT *?e*) as *?count* WHERE{ $?s_1$ **r** *?e* }
f(*s*)	SELECT COUNT (DISTINCT *?e*) as *?count* WHERE{ *?e* **s** $?o_2$ }

DBpedia Profiler covers the second condition with a strategy based on the intersection of the topic categories of the classes in the semantic explanation pattern *SEP*. Recall that the class topic category set function $scat : C_B \rightarrow 2^w$ maps each class into a set of topic categories. The *topic coherence* of a semantic explanation pattern *SEP* is defined as

$$th(SEP) = \frac{|\bigcap_{C \in cl(SEP)} scat(C)|}{|\bigcup_{C \in cl(SEP)} scat(C)|}$$

where *cl*(*SEP*), we recall, is the set of classes that occur in a semantic explanation pattern *SEP*. Then, the higher *th*(*SEP*) is, the more coherent *SEP* will be.

Finally, DBpedia Profiler computes the *coherent measure* of *SEP* as

$$coh(SEP) = \begin{matrix} (median\{PMI(r_i, r_j) / \ r_i \text{and } r_j \text{are properties in } SEP \text{ and } \ i < j\}) \\ + \qquad\qquad th(SEP) \end{matrix}$$

where the first factor is the PMI median score of the property pairs in *SEP* and the second factor is the coherence of the classes.

DBpedia Profiler adopts the Jaccard distance as the similarity measure for pairs of semantic explanation patterns. Given two semantic explanation patterns SEP_1 and SEP_2, the *Jaccard distance* between SEP_1 and SEP_2 is defined as

$$Jaccard(SEP_1, SEP_2) = \frac{|URIs(SEP_1) \cap URIs(SEP_2)|}{|URIs(SEP_1) \cup URIs(SEP_2)|}$$

where $URIs(SEP_i)$ denotes the set of RDF terms in a semantic explanation pattern SEP_i.

DBpedia Profiler computes the Jaccard distance in the path stream processing for each pair of semantic explanation patterns, after filtering. Then, it executes the One-Pass Microclustering algorithm, proposed in (MEI *et al*., 2006), to group the semantic explanation patterns. With the Jaccard distance, we expect to extract clusters such that the distance between inner-cluster explanations are bounded. The algorithm terminates when the minimal distance between clusters becomes larger than a user-specified threshold. The medoid of each cluster is selected as the representative semantic explanation pattern. This algorithm processes the set of semantic explanation patterns in one pass and, thus, is more efficient that those proposed in (MEI *et al*., 2006).

Using the set of representative semantic explanation patterns, DBpedia Profiler finally constructs the set of weighted topic categories and the set of similar entity pairs, as explained at the end of Section 4.2.

4.3.4. Construction of the Connectivity Profile

Finally, DBpedia Profiler constructs the connectivity profile for a pair of entities a and c in DBpedia, with maximum distance k, as the tuple $\mathbb{C} = ((C_a, C_c), s, EXP[a,c], T, E)$, where

- C_a and C_c are the immediate classes of a and c, as defined in Section 4.3.2
- $s = SCS(a, c)$, as defined in Section 4.3.3

- *EXP*[*a*, *c*] is the set of representative semantic explanations patterns, of length at most *k*, constructed as described in Section 4.3.3
- *T* is the set of top weighted topic categories for *P*, as defined in Section 4.3.3
- *E* is the set of similar entity pairs, as defined in Section 4.2.3, using the set of representative semantic explanation patterns in *P.*

4.4. Implementation and Evaluation

We compare and evaluate the process of explanation generation of DBpedia Profiler (available at http://lod2.inf.puc-rio.br/scs/proxs) with that of RECAP (available at http://lod2.inf.puc-rio.br/scs/recaps). We do not provide a detailed comparison of DBpedia Profiler with Explass (CHENG *et al*., 2014) and RelFinder (HEIM *et al*., 2009) because RECAP has already been shown to outperform these tools in (PIRRÒ, 2015; CHENG *et al*., 2014). However, we briefly summarize in Table 2 the functionality of these tools.

Table 2: Comparison of DBpedia Profiler with related tools

	RDF Graph	**Output**	**Querying Capabilities**	**Local Data**	**Connectivity Profile**
DBpedia Profiler	DBpedia	Graph, Paths, Explanation Patterns, Topic Categories	Yes	Yes	Yes
RECAP	Any	Graph, Paths	Yes	No	No
RelFinder	Any	Graph	No	Yes	No
Explass	DBpedia	Paths	No	Yes	No

The experiments were performed on an Intel Core i7 CPU 950 with 12 GB. The data processing layer of DBPEDIA Profiler was implemented in LUA and the application layer in PHP. The SPARQL query requests use LuaSocket and the parallel processing is supported by GNU parallel. The RDF data is encoded and stored in memory with Redis, for the effective processing of the path-stream. Since RECAP was originally not an on-line tool, we also implemented the explanation generation of RECAP, which is now available for the community scientific. DBpedia Profiler and RECAP were implemented to explore only DBpedia.

4.4.1. Performance Evaluating

We first compared the performance of DBpedia Profiler and RECAP for increasing values of k up to $k=5$ for the entire process of explanation generation, using the dataset with 26 pairs originally used to evaluate RECAP (PIRRÒ, 2015) and the Appendix 7.1 describes these entity pairs.

In the same way that (PIRRÒ, 2015), for each k, we generated all paths of length less than or equal to k. The evaluation considered the number of results returned for the SPARQL queries to retrieve the paths between two entities as a parameter, called LIMIT in the SPARQL query language (HARRIS *et al*., 2013). We do not have information about this parameter in the evaluation of (PIRRÒ, 2015). Since some queries require long processing times, LIMIT prevents of a possible timeout and improves the processing time. Considering a desired response time, this parameter was set to LIMIT=300 in DBpedia Profiler and to LIMIT=50 in RECAP. This configuration, apparently unjust to DBpedia Profiler, compensates for the extra time RECAP takes to de-duplicate and rank the patterns (FANG *et al*., 2011). DBpedia Profiler was set to disregard the 10% less coherent explanations, which is a different form of setting the coherence threshold; the Jaccard distance between the explanations was set to 0.92. RECAP computes the amount of information of a path and considers only the top-5 most informative paths.

Figures 10, and 11 show the average runtime of 5 runs for each of the 26 pairs. The average runtime for DBpedia Profiler was ~3.9 sec and that for RECAP, ~7.8 sec. Even with LIMIT=300, DBpedia Profiler achieved a shorter runtime than RECAP, with LIMIT=50. We recall that RECAP executes several online SPARQL query requests to compute the most informative path/pattern, whereas DBpedia Profiler has a pre-processing stage to build an index over the DBpedia class hierarchy to help compute the immediate classes, topic categories and the co-occurrence of pairs of properties.

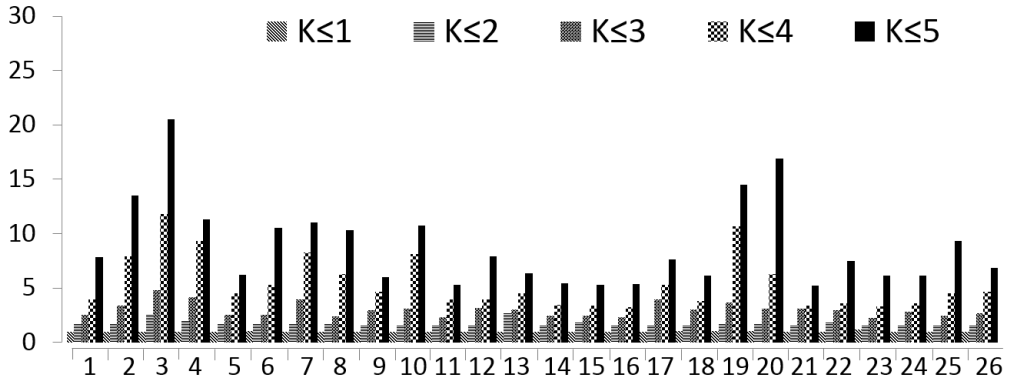

Figure 10: DBpedia Profiler process. Y-axis: time(s); X-axis: the entity pairs.

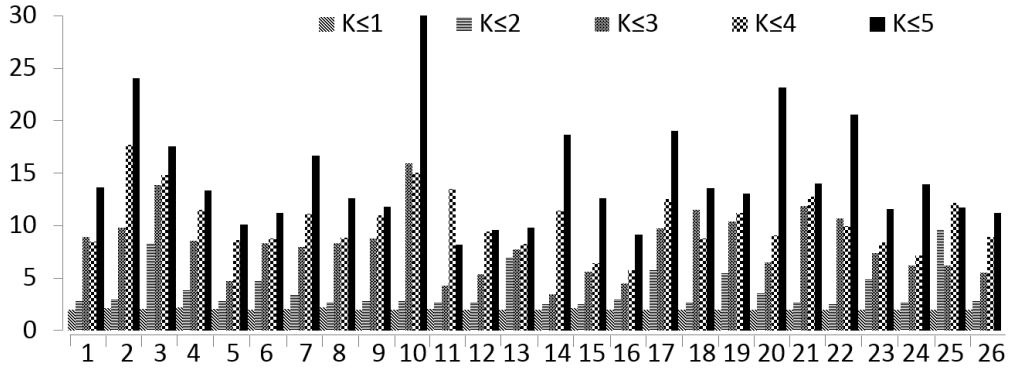

Figure 11: RECAP process. Y-axis: time(s); X-axis: the entity pairs.

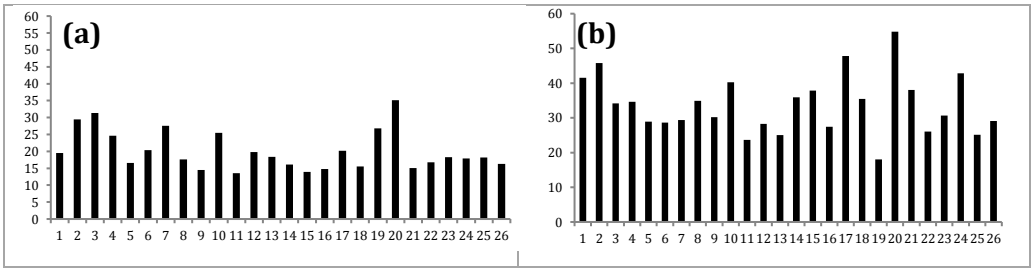

Figure 12: First Run of (a) DBpedia Profiler; (b) RECAP. Y-axis: time(s); X-axis: entity pairs.

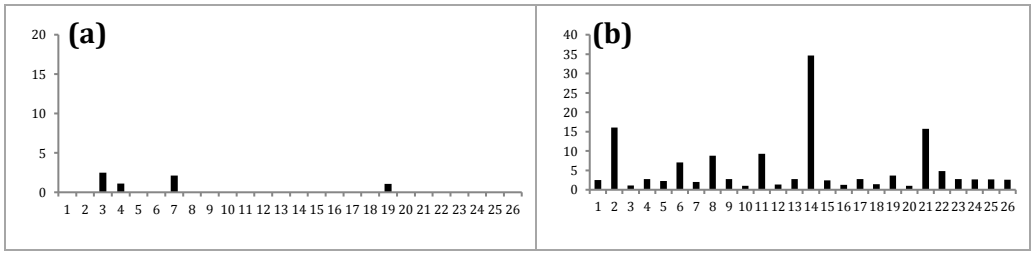

Figure 13: Computing Explanation to (a) DBpedia Profiler; (b) RECAP. Y-axis: time(s); X-axis: entity pair.

Figure 12 shows the runtime of the first run to generate the explanation patterns for DBpedia Profiler and RECAP, with *k*=5 and the configuration mentioned above. These measurements reflect the behavior of the tools with respect to a query user in real-time. The average runtime of DBpedia Profiler was ~20 secs and of RECAP, ~34 sec. For a second run, DBpedia responds with cached data, which explains the results in Figures 10 and 11. This experiment therefore reveals the efficiency of the stream processing of DBpedia Profiler, when compared with the parallel processing of RECAP.

Figure 13 plots the time to compute explanations patterns for DBpedia Profiler and RECAP, just for one run, with k=4 and LIMIT=300, for DBpedia Profiler, and LIMIT=50, for RECAP. These measurements were collected after the experiment of Figures 10 and 11; thus, both tools took advantage of the cached data. RECAP computed explanations in ~5.3 sec, on the average. We recall that RECAP spends time computing the most informative paths and then combines the paths using an approach similar to that adopted in (FANG *et al*., 2011). The combination of paths to compute explanations requires checking for duplicates (FANG *et al*., 2011), which has a high computational cost. Finally, RECAP computes the most informative patterns and present them to the user. DBpedia Profiler computed explanations in ~0.15 sec, on the average. We recall that DBpedia Profiler uses the One-Pass Microclustering algorithm to group the explanations. The complexity of this algorithm is linear, with respect to the number of explanations. The Jaccard distance (set to 0.92) was computed in the stream processing, together with the coherence score, using data computed at the pre-processing stage.

4.4.2. User Evaluation of the Explanations

This experiment aims at investigating whether DBpedia Profiler provides useful explanations to the user and comparing DBpedia Profiler with RECAP.

The experiment involved 20 graduate students, with some experience in Semantic Web technology. Each participant was assigned 6 random pairs among the 26 entity pairs. Prior to the experiment, the participants had a training stage to familiarize themselves with the tools. Following the methodology in (CHENG *et*

al., 2014), the participants were given a set of seven questions; the response to each question was in the form of an agreement value, ranging from 1 (min) to 5 (max). Q6 and Q7 were not considered in (CHENG *et al*., 2014). Q6 aimed at knowing whether the tools provided useful information to resolve the different tasks, since typically redundant information does not help the user. Q7 allowed the user to make a qualitative evaluation on the tool. We presented a brief description of this methodology in Appendix 7.1.

The results are reported in Table 3 and capture the experience of the users with the tools. We used Student's t-distribution to verify whether the mean ratings were statistically significant, with $p < 0.05$. The reliability analysis of questionnaire had a consistence of 0.88. According to Q2 and Q5, DBpedia Profiler provides better support to explain the tasks than RECAP. In Q7, the users expressed their satisfaction with the topic categories, immediate classes and the connectivity score returned by DBpedia Profiler. In Q6, even with LIMIT=300, DBpedia Profiler provided less redundancy than RECAP, with LIMIT=50 and returning only the top-10 paths. The SPARQL query patterns generated by RECAP were more complex and, in some cases, the entity pairs recovered were more specific that the pairs recovered by DBpedia Profiler.

Table 3: Questions/responses: mean (standard deviation).

Question	DBpedia Profiler	Recap
Q1: Information overview	4.03 (0.20)	2.96 (0.48)
Q2: Easiness in finding information	4.09 (0.19)	3.28 (0.47)
Q3: Easiness in comparing/synthesizing info	4.02 (0.22)	3.09 (0.61)
Q4: Comprehensive support	4.07 (0.21)	3.00 (0.43)
Q5: Sufficient support to the task	3.96 (0.26)	3.08 (0.44)
Q6: Redundancy in information	3.54 (0.36)	2.64 (0.58)
Q7: General Comments about the task and tool		

4.5. Discussion

In this chapter, we described a strategy to generate connectivity profiles for entity pairs in knowledge bases. The connectivity of two entities is explicitly described through a relationship paths set. We studied different process based on SPARQL queries to mine the paths between the entities and group them through their common characteristics. In this section, we discuss the mine process of RECAP and DBpedia Profiler, with the aim of justifying the results obtained in Section 4.4.

We developed a RECAP tool (PIRRÒ, 2015) and compared it with the DBpedia Profiler. RECAP applies the same graph mining strategy as REX (FANG *et al.*, 2011), which prunes the search space and generates explanation patterns with at least 1 instance. An explanation pattern is obtained by the combination of the paths found between two entities; if the paths have common entities and properties, then they can be combined. The main problem of this strategy, as the results in Section 4.4 show, is the computational cost to generate an explanation graph pattern and to check the graph isomorphism of the instances of one explanation pattern. Since graph isomorphism is an NP-complete problem, this make the generation of explanation patterns a high complexity process for on online knowledge databases (FANG *et al.*, 2011).

DBpedia Profiler executes a pipeline process to generate a connectivity profile for the given entity pair. The explanation patterns generated in this process are enriched with the Wikipedia Categories; these structures summarize the explanation patterns in a simple and clear way (KAWASE *et al.*, 2014). Wikipedia Categories are precomputed and are extracted in the online process. The users expressed their satisfaction with these structures in the user evaluation described in Section 4.4.2.

The class hierarchy of DBpedia is precomputed and it is used in the generation of semantic explanation patterns by the DBpedia Profiler. Explass (CHENG *et al.*, 2014) also used the classes of the entities to generate semantic explanation patterns, but this approach used all classes linked to one entity without defining the immediate class. In Explass, the process is computationally expensive and redundant since the total number of semantic explanation can be exponential. In DBpedia profiler, the semantic explanation patterns are generated using the

immediate class concept, which reduces the set of possible redundant classes (the process was defined in Section 4.2.2).

The mining process of the DBpedia Profiler uses a linear-clustering algorithm (MEI *et al*., 2006) to group the semantic explanation patterns by similarity, which helps identify representative explanations and summarize the set of explanations found between the entities. The representative semantic explanation patterns, defined in Section 4.3.3, help users find similar entity pairs, as explained at the end of Section 4.2. The similar entity pairs are examples that explain the relation between the target entity pair. These characteristics favor the performance and the user evaluation of the DBpedia Profiler.

In summary, the user of the DBpedia Profiler solved the search tasks easy and efficiently, RECAP generated complex explanation patterns and complicated the understanding of the relationships between the entities, but this approach does not need pre-processing of data.

4.6. Conclusions

In this chapter, we first introduced a strategy to generate connectivity profiles for entity pairs, which is independent of the specificities of the underlying knowledge base. Then, we described the DBpedia Profiler tool, which implements the strategy for DBpedia and follows an architecture based on RDF path stream processing. We compared the DBpedia Profiler with our implementation of RECAP and indicated the differences between the tools in terms of performance and quality of the explanations returned. Finally, we discussed the different mining processes compared in this Chapter.

The DBpedia Profiler tool explores the metadata aspects of DBpedia and the top main categories of Wikipedia. It implements an algorithm to find the immediate class of entities, and enriches the uncovered explanations. By exploring the DBpedia graph, DBpedia Profiler: computes a connectivity score to indicate the degree of relatedness of the entities; implements a new metric that measures the coherence of an explanation, through the co-occurrence of the properties and the intersection of topic categories; groups explanations using the Jaccard Distance and

the One-Pass Microclustering algorithm (MEI *et al*., 2006) to summarize sets of explanations.

Although the categories of Wikipedia add semantics to the explanation, some categories have a very broad context, which complicates the classification of explanations. For example, the categories People, Society and Humanities share many contexts with Person. A major goal for future research is to develop an optimized classifier to improve the semantics of the explanations.

5
Comparing Search Strategies to Understand the Connectivity of Entity Pairs

5.1.
Introduction

In this chapter, we are interested in exploring knowledge bases to understand how two entities are connected. More specifically, given an entity pair, we focus on the relationship paths that better describe the connectivity between them.

Several approaches (HEIM *et al*., 2009; FANG *et al*., 2011; PIRRÒ, 2015; CHENG *et al*., 2014; MOHAN *et al*.; 2014; HERRERA *et al*., 2016) have been proposed to explore and understand the connectivity of pairs of entities in a knowledge base. Such approaches apply a simple strategy:

(1) Search for relationship paths between pairs of entities – the larger the number of paths found, the stronger the connectivity between the entities is likely to be.
(2) Sort the paths found and select relevant paths.

However, this simple strategy poses two initial challenges:

(1) How to find relationship paths between a given pair of entities in a knowledge base?
(2) How to sort and select the most relevant relationship paths?

A third, important challenge that must also be addressed is:

(3) How to evaluate and compare search strategies that explore the connectivity of entity pairs?

To address the first challenge, we develop a generic search strategy based on the backward search heuristic (LE *et al*., 2014). Using simple HTTP requests, the backward search heuristic simultaneously starts from the vertices in the RDF graph that correspond to the pair of input entities, and recursively expands the search to their neighboring nodes until a candidate relationship path is generated. The expansion process uses entity similarity measures (e.g. Jaccard Similarity) to

prioritize certain paths over others. The similarity measure filters the entities less related to the target entities to help the search method identify more meaningful paths.

As for the second challenge, we use ranking approaches (PIRRÒ, 2015; HERRERA *et al.*, 2016; HULPUS *et al.*, 2015) which use the semantics of the relationships between the entities to assign a score to relationship paths. After sorting the set of relationship paths, the top-*k* paths are selected to describe the connectivity of an entity pair. The combination of similarity measures and ranking approaches therefore generate different search strategies that must be evaluated.

To address the third challenge, we create a ground truth for the music and movies domains. For each domain, the ground truth consists of a set of entity pairs and, for each pair, a set of relationship paths, ranked using domain-specific knowledge. The ranked list of relationship paths that each search strategy generates is then compared through the ground truth. After, the search strategy with the best results on music and movie domains is compared with the baseline chosen from the literature. Finally, we discussed some drawbacks of the strategies compared.

The contributions of this chapter can be summarized as: (1) A search process that combines entity similarity measures and ranking approaches; (2) The definition of a ground truth in two entertainment domains to compare the strategies; (2) A comparison between the different search strategies and a baseline from the literature.

The rest of this chapter is structured as follows. Section 5.2 introduces the search strategies compared in this work, and details a generic heuristic to find relevant relationship paths. Section 5.3 compares the different search strategies and discusses the results. Finally, Section 5.4 presents the conclusions.

5.2. Search Strategies to Find Relevant Paths

5.2.1. Overview

Let B be a knowledge base and $G_B = (N_B, E_B, e_B)$ be the *RDF graph* of B. We introduce a search process for G_B that receives as input an entity pair and outputs a list of paths in G_B between the entity pair. The search process has two basic steps:

(1) Find a set of meaningful[4] relationship paths between two entities.

(2) Sort the set of meaningful relationship paths by relevance.

The first step follows the backward search heuristic to generate meaningful paths between the entities in the pair. The heuristic uses a breadth-first search (BFS) to explore the neighbors of each input entity. Two BFS, that we call left and right, are executed in parallel to traverse the RDF graph. In each expansion step, a BFS uses *activation criteria* to prioritize and select entities that will be used to expand the partially constructed paths. A path is generated if both BFSs reach a common entity or a target entity.

The first step adopts two activation criteria: (1) Give priority to entities (nodes) with low degree in G_B; (2) Maintain entities that are similar to the last entity reached in a partially constructed path. We consider three different similarity measures, the Jaccard Distance, Wikipedia Link-based Measure, and SimRank, reviewed in, the Chapter 2, Section 2.4.

The second step receives as input the set of paths found in the first step and uses relationship-path ranking measures to sort the paths by relevance and recommend the top-*k* paths. We study three ranking measures, Predicate-Frequency Inverse-Triple-Frequency, Exclusivity-based Relatedness, and Pointwise Mutual Information, reviewed in Chapter 2, Section 2.5.

With the aim of creating search strategies and finding relevant paths, we combined similarity and ranking measures in the same process. We generate 9 search strategies listed in the Table 4. The first column is the short name of the strategies, which we will use in the evaluation in Section 5.5.

[4] The term "meaningful" is used here just to give an intuition that not all relationship paths will be considered.

Table 4: Search Strategies

Short Name	Name
J&I	Jaccard Distance & Predicate Frequency Inverse Triple Frequency
J&E	Jaccard Distance & Exclusivity-based Relatedness
J&P	Jaccard Distance & Pointwise Mutual Information
W&I	Wikipedia Link-based Measure & Predicate Frequency Inverse Triple Frequency
W&E	Wikipedia Link-based Measure & Exclusivity-based Relatedness
W&P	Wikipedia Link-based Measure & Pointwise Mutual Information
S&I	SimRank & Predicate Frequency Inverse Triple Frequency
S&E	SimRank & Exclusivity-based Relatedness
S&P	SimRank & Pointwise Mutual Information

5.2.2. Finding Relationship Paths between Entity Pairs

In this section, we describe a process for discovering paths that link two entities in an RDF graph. We analyze the structure of the RDF graph and generate relevant relationship paths.

5.2.2.1. Activation Criterion

As already mentioned, the first step of the search process uses two activation criteria: (1) give priority to entities (nodes) with low degree in G_B; (2) maintain entities that are similar to the last entity reached in a partially constructed path.

As for the first criterion, if the number of links (in and out) of an entity is large, then it might be very expensive to expand the search from that entity. Therefore, waiting for the expansion from the other entity might be less expensive. Hence, entities with a low degree (in and out) should be given priority to create relevant paths.

With respect to the second criterion, finding relevant paths means connecting semantically close entities, by prioritizing the selection of similar entities (DE VOCHT et al., 2013).

Example 5. Consider the RDF graph in Figure 14, a backward search is executed from each of the target entities, dbr:Michael_Jackson and dbr:Whitney_Houston.

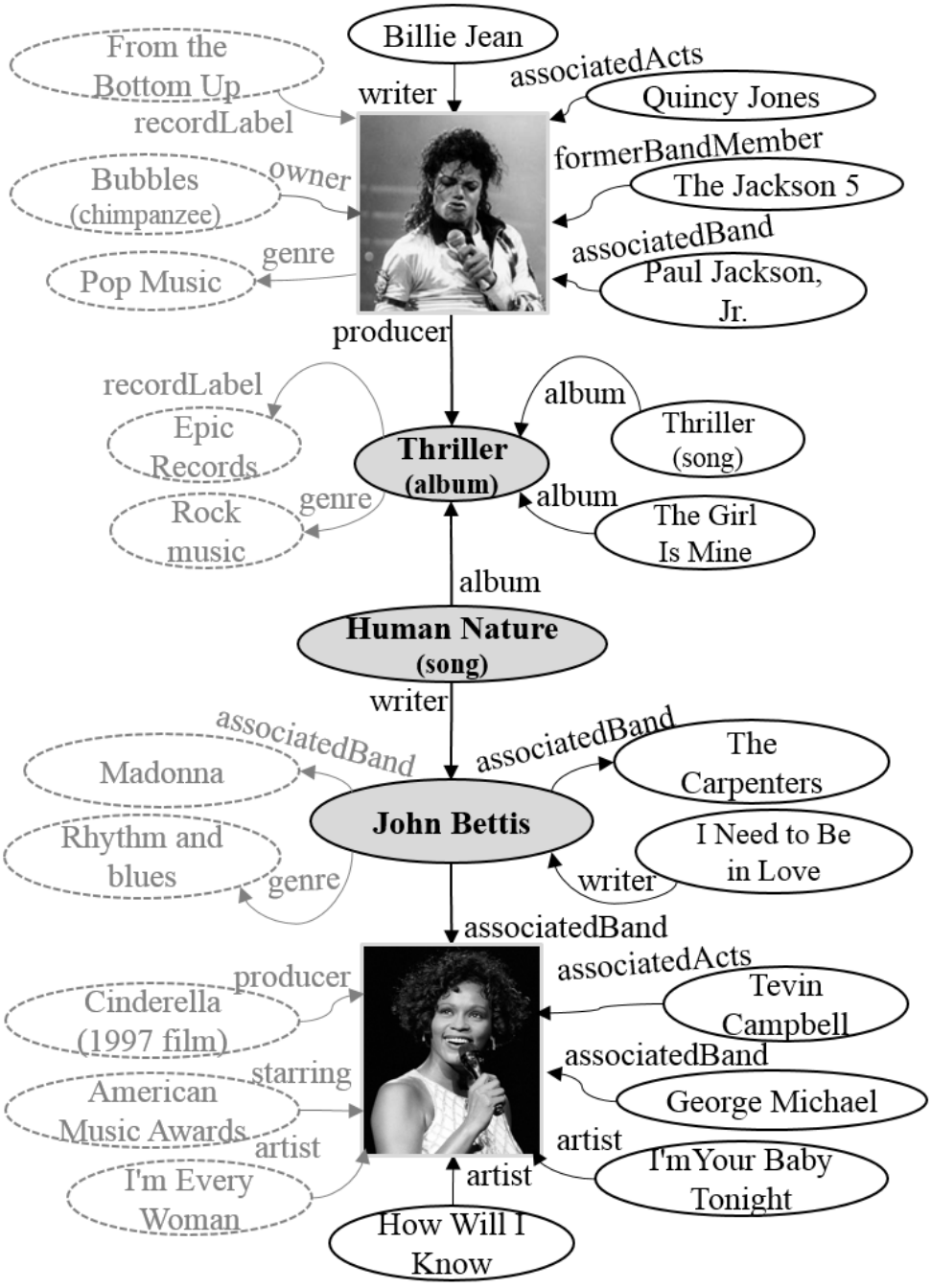

Figure 14: Finding paths between Michael Jackson and Whitney Houston using the activation criterion.

The backward search from dbr:Michael_Jackson first analyses the adjacency list of dbr:Michael_Jackson to identify which entities should be included in the search. According to the first activation criterion, the search process prioritizes the expansion, for example, of dbr:Thriller_(album), with 111 links, over dbr:The_Jackson_5, with 340 links. To apply the second activation criterion, the

search process uses a similarity measure (e.g., Jaccard distance) to prioritize the entities shown in solid line circles, such as dbr:Thriller_(album) and dbr:The_Jackson_5. At the second round, the search process analyses, for example, the adjacency list of dbr:Thriller_(album). Here, the activation criteria prioritize dbr:Human_Nature_(song).

The backward search from dbr:Whitney_Houston is similar.

The search process finds a path when the two backward searches reach one of the target entities, or when they collide on the same entity, in this case, dbr:Human_Nature_(song).

5.2.2.2. Backward Search on Knowledge Graphs

Algorithm 1 describes, in pseudocode, the backward search heuristic and its main components. The input consists of an entity pair v_{start} and v_{end}, a maximum distance k and an activation function $\boldsymbol{\tau}$. The sub-paths generated are stored in main memory in hash tables, R_{left} and R_{right}. A path is generated if there are sub-paths of R_{left} and R_{right} which collide in the same entity, or if one of the sub-paths reaches a target entity. When the sub-paths reach the same entity, they are indexed by the entity. For example, if e is the entity reached, the process will check sub-paths $r_{left} \in R_{left}[e]$ and $r_{right} \in R_{right}[e]$ to generate complete paths. The output is a set of paths R that link the given entity pair.

Main Loop (Line 6 – Line 14). The main loop expands paths from v_{start} and v_{end}. The loop executes the operations *extraction*, *expansion* and *intersection*. In a parallel fashion, we also structure the collection of entities V as a queue with the entities tagged as *left* and *right*. This enables grouping the *extraction*, *expansion*, and *intersection* operations in only one process, which are then executed in parallel. Finally, the paths are stored in main memory in table R, so that they can be consumed without waiting for the completion of the backward search.

Extraction (Line 15 – Line 19). The description of an entity consists of a set of specific property values and relationships with others entities. The auxiliary function *extract* retrieves the description of an entity e from the RDF graph G and stores it in a hash table T (Line 17).

Algorithm 1. BackwardSearch (G, v_{start}, v_{end}, k, τ): R
Input: an entity pair v_{start} and v_{end} , maximum distance k and a function of activation τ
Output: a set of paths R that link the given pair of entities

1: $R = \emptyset$, $V = \emptyset$, $T = \emptyset$, $expanding = 0$
2: $side = 0$, $left = 0$, $right = 1$
3: $V_{left} = \emptyset$, $V_{right} = \emptyset$, $R_{left} = \emptyset$, $R_{right} = \emptyset$
4: $V_{left} = \{v_{start}\}$, $V_{right} = \{v_{end}\}$
5: append v_{start}, v_{end} to V

6: **repeat**
7: $extraction(V_{side}, T)$
8: $R_{side}, V_{side} = expansion(V_{side}, R_{side}, T, \tau)$
9: append the entities of V_{side} to V (duplicates are discarded)
10: append the paths of $intersection(R_{left}, R_{right}, V, v_{start}, v_{end})$ to R
11: $expanding = expanding + 1$
12: $side = expanding \% 2$
13: **until** $expanding <= k$
14: return R

15: **function** extraction(V_{side}, T)
16: **for** $e \in V_{side}$ **do**
17: append new entities of $extract(e)$ to T (if the content of the entity e is not in the Local Database, then get it from G)
18: **end for**
19: **end function**

20: **function** expansion(V_{side}, R_{side}, T, τ):R_{new}, V_{new}
21: $R_{new} = \emptyset, V_{new} = \emptyset, V_{adjacency} = \emptyset$
22: **for** $e \in V_{side}$ **do**
23: $V_{adjacency} = activation(T[e], e, \tau)$
24: **for** $e_t \in V_{adjacency}$ **do**
25: append e_t to V_{new} (duplicates are discarded)
26: append sub-paths $getSubPaths(e, e_t, R_{side}[e])$ to R_{new}
27: **end if**
28: **end for**
29: **end for**
30: return R_{new}, V_{new}
31: **end function**

32: **function** intersection(R_{left}, R_{right}, V, v_{start}, v_{end}): R_{new}
33: $R_{new} = \emptyset$
34: **for** $e \in V$ **do**
35: **for** $r_{left} \in R_{left}[e]$ **do**
36: **for** $r_{right} \in R_{right}[e]$ **do**
37: append $r_{new} = join(r_{right}, r_{left}, v_{start}, v_{end})$ to R_{new}
38: **end for**
39: **end for**
40: **end for**
41: return R_{new}
42: **end function**

Expansion (Line 20 – Line 31). Line 23 of the *expansion* function contains a call to the function *activation*, which applies the activation function τ to the adjacency list of an entity e. An entity e_t will participate in the next expansion, if it meets the activation criteria. The call returns a subset $V_{adjacency}$ of entities recovered for an entity e through the hash table T stored in the memory.

The function *getSubPaths* in Line 26 extends the sub-paths that end at an entity e; a new sub-path is indexed by an entity $e_t \in V_{adjacency}$ reached in the expansion. The new sub-paths are stored in R_{side} at the end of the call of the expansion function.

Intersection (Line 32 - Line 42). The *intersection* function generates single paths through the function *join*. The function *intersection* receives the collections R_{left}, R_{right} and V. The call of function *join* considers two case: (*i*) if two sub-paths $r_{left} \in R_{left}[e]$ and $r_{right} \in R_{right}[e]$ reach the same entity e, they are joined in a single path with maximum length k. (*ii*) If a path r_{left} (or r_{right}) reaches the target entity v_{end} (or v_{start}), then a single path that ends in the target entity is created. Finally, the paths are stored in the hash table R.

Final Remarks. In the backward search heuristic, the activation function can be used in at least two ways: (*i*) *one-to-one*, when a new entity *f* adjacent to the current entity e is activated because the value of the activation score for e and *f* is above a given threshold; (*ii*) *one-to-many*, when the adjacency list of an entity e is sorted by the activation score and entities with higher scores are considered for expansion. We adopted the second approach.

5.3. Evaluation and Results

In this section, we compare the quality of the 9 search strategies introduced in Section 4 and analyze the results. We first detail the parameter setting adopted in the experiments. Then, we describe how we constructed the ground truth. Next, we present the baselines adopted. Finally, we compare the 9 search strategies with a ground truth and with the baselines adopted.

We used DBpedia as the knowledge base and selected two domains, music and movies, which are well-represented in DBpedia. We performed the experiments for one domain and then repeated them for the other domain to corroborate the

findings. To construct the ground truth, we resorted to the Internet Movie Database – IMDb, the most popular and authoritative source for movies, TV and celebrity content, and to last.fm, an online music catalog with free music streaming.

5.3.1. Evaluation Setting

5.3.1.1. Search Strategy settings

As we have already pointed, the search process studied in this chapter has two basics step: finding and sorting meaningful paths.

Finding paths. In this step, we use a backward search heuristic with two activation criterions: the number of links of an entity and the similarity between entities. We set the number of links to expand the entities to 200. This value is deduced from statistics published by DBpedia[5], which indicate that 90% of the entities have less than 200 links. As in (MOORE *et al.*, 2014), we assume that nodes with high degree influence the pathfinding process with potentially very unspecific information.

In the computation of the similarity score, we must face two problems: the similarity measures use different artifacts to compute the score and the entities can be of different types (for example, a movie and an actor). To get around this problem, the adjacency list of each entity is sorted by similarity and only the top 50% of the entities are considered, independently of the size of the list and the similarity scores. We considered 50% of the list because it is a moderate factor to maintain the connectivity between entities and propagate the similar score in the graph (COHEN, 2010).

The maximum path length between two entities is set to 4, since it is a value backed up by the small world (WATTS *et al.*, 1998) phenomenon, that says that a pair of nodes is separated by a small number of connections, and since it was confirmed in previous experiments (NUNES *et al.*, 2013).

Sorting paths. After finding the relationship paths between the entities, we use a ranking measure to sort the paths and recommend the top-*k* paths in the

[5] http://downloads.dbpedia.org/2015-04/ext/pagerank/

ranking. We considered the first 50 paths, because this is the minimum number of path recovered in the finding path process of the search strategies. Also, as reported in (FANG *et al*., 2011; PIRRÒ, 2015; CHENG *et al*., 2014; HULPUS *et al.*, 2015), this value suffices to explore the connectivity between the entities.

Entity dereferencing. We assume that the entities in a knowledge base are identified by URIs that use the http:// scheme. These entities can be looked up simply by dereferencing the URI using the HTTP protocol. Thus, given a URI *u* on a knowledge base, if *u* is RDF dereferencable, then dereferencing *u* should retrieve RDF triples whose subject is *u* (HARTIG *et al*., 2009).

5.3.1.2. Data Quality

In DBpedia, many properties *p* are redundantly defined as http://dbpedia.org/ontology/*p* and at the same time as http://dbpedia.org/property/*p*. After the relationship paths are found, these duplicates must be removed, a cleaning up process that reduced the number of paths in at least 50%, as also reported in (PASSANT, 2010).

5.3.1.3. Ranking Quality

We adopted the *Discounted Cumulative Gain* – DCG (JÄRVELIN et al., 2002) to measure the quality of the rankings obtained. DCG is a well-known measure used in Information Retrieval to assess ranking quality. This measure accumulates the gain from the top of a ranked list to the bottom, penalizing lower ranks, and can be parameterized to consider only the top-*k* elements of the ranked list.

Consider a list with n documents with *ratings* $rel_1,\ldots,rel_n$ and let The *discounted cumulative gain* of the top-*k* results, with $1 \leq k \leq n$, denoted DCG_k, is defined as

$$\mathrm{DCG}_k = rel_1 + \sum_{i=2}^{k} \frac{rel_i}{log_2\, i}$$

In our experiments, the relationship paths will play the role of the documents.

5.3.2.
Constructing the Ground Truth

The construction of the ground truth poses three major challenges: (1) how to select entity pairs; (2) how to find relationship paths for the entity pairs selected; and (3) how to rank the relationship paths. We addressed these challenges in the movies and music domains.

The Ground Truth is summarized in the Appendix 7.2.

5.3.2.1.
Selecting Entity Pairs

We focused on best-selling music artists[6], in the music domain, and on famous classic actors and actresses[7], in the movies domain. We considered the box office sales and the actor's fame as relevance criteria for the music and movies domains.

After selecting a list of entities from each of these two domains, we submitted each entity to Google Search to select a set of related entities. Then, for the possible entity pair, we computed their semantic connectivity score[8] (NUNES *et al.* 2014) in DBpedia, with maximum length 4, to discover entity pairs with high connectivity. The maximum path length between two entities was set to 4, since it is a value backed up by the small world (WATTS *et al.*, 1998) phenomenon, which says that a pair of nodes is separated by a small number of connections, and since it was confirmed in previous experiments (NUNES *et al.*, 2013). Table 5 and Table 6 show the entity pairs for the music and movies domains, respectively.

[6] https://en.wikipedia.org/wiki/List_of_best-selling_music_artists

[7] http://www.imdb.com/list/ls000035399/

[8] http://lod2.inf.puc-rio.br/scs/SemConnectivities

Table 5: Entity pairs for Music Domain.

#	Entity Pair	#	Entity Pair
1	Michael Jackson, Whitney Houston	**11**	Eminem, Dr. Dre
2	The Beatles, The Rolling Stones	**12**	Don Henley, Eagles (band)
3	Elton John, George Michael	**13**	Phil Collins, Peter Gabriel
4	Led Zeppelin, The Who	**14**	Lil Wayne, 2 Chainz
5	Pink Floyd, David Gilmour	**15**	Rod Stewart, Faces (band)
6	U2, R.E.M.	**16**	Bob Dylan, Stevie Nicks
7	Metallica, Anthrax (American band)	**17**	Fleetwood Mac, Don Henley
8	Rihanna, Nicki Minaj	**18**	Paul McCartney, Ringo Starr
9	Guns N' Roses, Velvet Revolver	**19**	Jay Z, Kanye West
10	Bob Dylan, The Band	**20**	Genesis (band), Steve Hackett

Table 6: Entity pairs for Movie Domain.

#	Entity Pair	#	Entity Pair
21	Elizabeth Taylor, Richard Burton	**31**	Humphrey Bogart, Lauren Bacall
22	Cary Grant, Katharine Hepburn	**32**	Judy Garland, Mickey Rooney
23	Laurence Olivier, Ralph Richardson	**33**	Miriam Hopkins, Ernst Lubitsch
24	Errol Flynn, Olivia de Havilland	**34**	Myrna Loy, Jean Harlow
25	William Powell, Myrna Loy	**35**	Jean Harlow, Clark Gable
26	James Stewart, Henry Fonda	**36**	Grace Kelly, Alfred Hitchcock
27	Paul Newman, Joanne Woodward	**37**	Vivien Leigh, Laurence Olivier
28	Bette Davis, Joan Crawford	**38**	Joan Blondell, James Cagney
29	John Wayne, Kirk Douglas	**39**	Ronald Colman, Samuel Goldwyn
30	Charlie Chaplin, Frank D. Williams	**40**	Audrey Hepburn, Gregory Peck

5.3.2.2. Mapping Entities

As a preparation for the path finding process, we mapped classes in specialized ontologies to classes of the DBpedia ontology[9], and to classes of the reference datasets, IMDb and last.fm. For the music domain, we adopted the Music Ontology[10] and, for the movies domain, we selected the Movie Ontology[11], related to people and movies. Tables 7 and 8 show the mappings adopted.

Music domain. To map DBpedia entities to last.fm, we used the keyword search API of last.fm[12]: api:artist.getInfo, api:album.getInfo and api:track.getInfo.

We first determined whether the entity represented an artist or a musical content by analyzing the rdf:type property, as in (HERRERA *et al.*, 2016). For example, the entity dbr:Michael_Jackson has type dbo:Artist. If the entity represented an artist, we extracted keywords from its URI (such as "Michael + Jackson") and submitted them to api:artist.getInfo[13] to search for the entity. If the search was successful, we had an exact mapping, otherwise we used other keywords. It the entity represented musical content (an album, song or single), we had to identify its main artist in DBpedia, through the property dbp:artist. For example, the main artist of dbr:Thriller_(album) is dbr:Michael_Jackson. If the entity represented a musical album, we called api:album.getInfo[14] to search for the entity. Similarly, it the entity represented a song or a single, we called api:track.getInfo.

[9] http://mappings.dbpedia.org/server/ontology/classes/

[10] http://musicontology.com/

[11] http://www.movieontology.org/

[12] http://www.last.fm/api

[13] api?method=artist.getinfo&artist=Michael+Jackson

[14] api?method=album.getInfo&artist=Michael+Jackson&album=Thriller

Table 7: Mapping in the Music Domain

Music Ontology	DBpedia Ontology	last.fm
mso:MusicGroupArtist	dbo:Band	Artist
mso:MusicArtist	dbo:Artist, dbo:Singer dbo:MusicalArtist, dbo:MusicDirector, dbo:ClassicalMusicArtist, dbo:MusicComposer, dbo:SongWriter	
mso:album	dbo:Album	Album
mso:single, mso:soundtrack, mso:track, mso:MusicalWork	dbo:Single, dbo:Song, dbo:MusicalWork	Track

Table 8: Mapping in the Movie Domain

Movie Ontology	DBpedia Ontology	IMDb
mvo:Director	dbo:MovieDirector	Artist
mvo:Actor	dpo:Actor, dbo:Artist	
mvo:Producer	dbo:Producer	
mvo:Writer	dbo:Writer, dbo:MusicComposer	
mvo:Musical_Artist	dbo:MusicalArtist	
mvo:Movie	dbo:Film	Movie

Movies domain. In DBpedia, to find out if an entity is a movie, we used the property rdf:type; in any other case, the entity is considered as a participant of a movie (see Table 8). The immediate type of an entity is identified using the method proposed in Chapter 4.

To map DBpedia entities to IMDb, we imported the database of IMDb[15] to a local PostgreSQL database and recreated information about Name, Movie and Cast (people who worked on a movie). Usually, the entities in DBpedia have an auto description in the URI; for example, the movie dbr:Cleopatra_(1963_film) has the name and release year in its URI. We used this basic description to find the same entity in IMDb through classic SQL wildcards, as for example:

SELECT id, title from **movie** where
{ title like '%Cleopatra%' and year = 1963 }

[15] ftp://ftp.fu-berlin.de/pub/misc/movies/database/

SELECT id, name from **name** where
{ name like '%Elizabeth%Taylor%' }

For those cases where the queries returned more than one result, we used the Levenshtein Distance (LEVENSHTEIN *et al.*, 1966) to choose the most similar IMDb entity to the DBpedia entity.

5.3.2.3. Finding Relationship Paths

We created 40 ranked sets of relationship paths – one for each of the 40 entity pairs of Tables 5 and 6 – to generate the ground truth in two entertainment domains

We used the Algorithm 1 to identify the paths between each the entity pairs listed in Tables 5 and 6. The classes or types of an entity in DBpedia are defined through the rdf:type property. The classes of the DBpedia ontology in music and movie domains are defined in Tables 7 and 8, respectively. The entities that belong to previous classes are considered in the generations of relationship paths in DBpedia. In Algorithm 1, the backward search use as single activation function the classes of the DBpedia ontology in the domain concerned, the expansion process analyses the types of each entity in *one-to-one* way, if an entity belongs to a class of the ontology domain, then it is prioritized to generate relationship paths. The other considerations of the backward search process are the same, as defined in Section 5.2.2.2.

5.3.2.4. Ranking the Relationship Paths

We ranked the paths in each of the 40 sets using semantic information extracted from IMDb and last.fm to compute entity ratings, and information extracted from DBpedia to compute property relevance scores (the Appendix 7.3 presents examples of path rankings using the ground truth).

To obtain the ranked lists, we first computed the *score* of each path π as the average of the *rating* of the entities involved in the path. Recall that π is a path in the DBpedia graph. Each entity e used in π was first mapped to an equivalent entity e' in IMDb or last.fm, as explained in Section 5.3.2.2; the rating of e' was computed

from data in IMDb or last.fm, as described below, and assigned to e. Finally, the score of π was computed as the average of the ratings of the entities that occur in π.

For each entity pair, we ranked the paths using their scores and retained the top 50 paths. However, since the path score ignores the relevance of the properties, paths that involve the same entities will have the same score. As a further step, we inspected each ranked list and used the *relevance scores* of the properties, computed in DBpedia, to help rank the paths with the same entities.

This ranking process is justified for two basic reasons. On one hand, we intended to create a dataset that would help evaluate approaches that address the entity relatedness problem, which typically involve a path ranking measure. Therefore, it would not be reasonable to adopt a path ranking measure from the literature (which would create ranked lists biased to that measure). On the other hand, it would be infeasible to manually rank the relationship paths that connect two entities (in DBpedia), whose number is typically very high. Hence, we opted to: (1) select two domains – music and movies – for which specialized data were available; (2) filter the paths in DBpedia so that they traverse only entities in each of these domains; (3) use specialized domain data to pre-rank the paths found; (4) manually inspect and sanction the pre-ranking, which proved to be a feasible task. The computation of entity ratings and property relevance scores is detailed below.

Entity rating in the music domain. In last.fm, each artist and musical content, has two scores of relevance: listeners and play count. This information is recovered by the search api of last.fm. The listeners score represents the number of different users who listen a song, and the play count is the number of times a person listens a song. An album, depending on the number of songs, receives as play count score (or listener score) the sum of the play count scores (or listeners scores) of all songs in the album. Similarly, an artist receives a play count and listener score. In this chapter, we use the play count to create a rating of the entities in the music domain and to generate a semantic score for the relationship paths between entities, which better expresses the people's preferences. If the entity is not identified in the mapping, we assign as a zero score.

Entity rating in the movies domain. IMDb publishes user-generated ratings for movies; an IMDb registered user can cast a vote (from 1 to 10) on every released

movie in the database. Users can vote as many times as they want, but each vote will overwrite the previous one so it is one vote per movie per user. In the case of people involved in a movie, we compute the average rating of the movies where the person participated to generate his/her rating. We imported the movies ratings to our local database and, with the table Cast, we related movies and actors to compute the artist rating. Again, if the entity is not identified in the mapping, we assign a zero score.

Property relevance score in DBpedia. We used the *inverse triple frequency* (ITF) (PIRRÒ, 2015). as the property relevance score, defined as $itf(p, G_B) = log \frac{|B|}{|B_p|}$, where $|B|$ is the number of triples in the knowledge base B and $|B_p|$ is the number of triples in B whose property is p.

Example: Consider the following paths of the DBpedia RDF graph:

P_1. Elizabeth_Taylor **^producer** The_Taming_of_the_Shrew **starring** Richard_Burton

P_2. Elizabeth_Taylor **^starring** The_Taming_of_the_Shrew **starring** Richard_Burton

where "Elizabeth_Taylor", "Richard_Burton" and "The_Taming_of_the_Shrew" actually are abbreviations for the URIs of these DBpedia entities, and likewise for the properties.

The first step is to compute the entity rating of these entities using information from IMDb, which involves finding these DBpedia entities in IMDb. The path scores are computed as the average of the rating of the entities in the path. Since these two paths involve the same entities, they will have the same score. The second step is then to compute the ITF in DBpedia of the properties "^starring" and "^producer" to help disambiguate the ranking. Since "^producer" is less frequent in DBpedia than "^starring", it has a higher ITF. Path P_1 should then be ranked before P_2. However, this is subjected to manual inspection to confirm the preference of P_1 over P_2, which was the final decision in this case, on the grounds that P_1 is perhaps more informative to the user than P_2.

5.3.3. Baselines

We adopt RECAP (PIRRÒ, 2015) as the baseline, where the RDF paths between entity pairs are found using SPARQL queries, no activation criteria is used, and the

maximum distance considered between entities is set to 4 (NUNES *et al.* 2014). Therefore, the paths are generated by the following set of SPARQL queries:

SELECT * WHERE { V_{start} (p_1| ^p_1) ?e_1. ?e_1 (p_2 | ^p_2) ?e_2 . ?e_2 (p_3 | ^p_3) ?e_3 . ?e_3 (p_4 |^p_4) V_{end} . } (*i*)

SELECT * WHERE { V_{start} (p_1| ^p_1) ?e_1. ?e_1 (p_2 | ^p_2) ?e_2 . ?e_2 (p_3 | ^p_3) V_{end} . } (*ii*)

SELECT * WHERE { V_{start} (p_1| ^p_1) ?e_1 . ?e_1 (p_2 | ^p_2) V_{end} . } (*iii*)

SELECT * WHERE { V_{start} (p_1| ^p_1) V_{end} . } (*iv*)

We note again that, since an RDF graph is a directed graph, to find undirected paths between entities, one should also consider the inverse predicates. The assumption behind this decision is that all predicates can be considered to have a semantically sound inverse (HULPUS *et al*., 2015).

After recovering the relationship paths with the SPARQL queries, they are sorted by Predicate-Frequency Inverse-Triple-Frequency measures, as defined in (PIRRÒ, 2015).

We create a second baseline by replacing the Predicate-Frequency Inverse-Triple-Frequency by the Exclusivity-based Relatedness ranking measure.

In the following, we identified the first baseline as “Q&I” and the second, as “Q&E”.

5.3.4. Comparison of the Search Strategies with the Ground Truth

For the movies and music domains, respectively, Tables 9 and 10 show the results of comparing the 9 search strategies with the ground truth. The first row of each table identifies the strategies by their short name, defined in Table 4, and the last row shows the average DCG score obtained by each strategy. The first column of each table contains the identifier of the entity pairs defined in Tables 5 and 6. The second column indicates the search strategy with the best DCG for each entity pair.

For the movies and music domains, Tables 8 and 9 respectively present pairwise comparisons between the strategies, using Student's t-distribution, to verify whether the average scores were statistically significant, with $p < 0.05$.

For the music domain, Table 9 indicates that W&E has the best average DCG score. But, the second column indicates that the J&E strategy provided the best DCG results for 8 entity pairs. However, Table 9 reveals that the W&E and J&E strategies are statistically similar, with $p = 0.3925$. Table 11 also shows that the differences between the average scores of the W&E and J&E strategies and the other strategies were all statistically significant, with $p < 0.05$.

We also observe that, in the music domain, the differences between the strategies are evident, probably because the entity ranking system of last.fm is not restricted to intervals, as in IMDb.

For the movies domain, Table 10 indicates that J&E has the best average DCG score. The second column confirms that J&E provided the best DCG results for 7 entity pairs. A similar behavior is obtained by the W&E search strategy for 5 entity pairs, possibly because the strategies share the same ranking measure. Table 12 reveals that the W&E and J&E strategies are statistically similar, with $p = 0.1382$. Table 10 also shows that the differences between the average scores of the J&E strategy and the other strategies were all statistically significant, with $p < 0.05$. The pairwise comparison also reveal that W&E does not have differences which are statistically significant from J&I ($p = 0.3344$), J&P ($p = 0.2178$), S&I ($p = 0.08179$) and S&E ($p = 0.9569$).

Thus, among the 9 search strategies compared, we may conclude that the J&E strategy (Jaccard Distance & Exclusivity-based Relatedness) should be preferred, in the two domains. In addition, the results of Tables 9 and 10 indicate that the Exclusivity-based Relatedness is the best path ranking measure.

Table 9: DGC@50 in the music domain

#	>	J&I	J&E	J&P	W&I	W&E	W&P	S&I	S&E	S&P
1	**J&E**	458,988,458.17	**633,392,974.66**	447,360,722.94	487,470,716.28	636,710,925.70	454,857,077.44	476,783,713.90	606,142,751.11	455,998,796.03
2	**W&E**	1,971,091,562.27	2,753,779,539.02	2,105,604,575.92	1,888,437,083.83	**2,825,550,196.74**	2,106,116,034.85	1,888,193,453.76	2,097,662,415.43	1,912,962,889.62
3	**W&E**	192,897,579.76	273,016,513.27	212,556,639.60	200,567,193.24	**289,912,884.80**	212,556,639.60	194,453,367.92	248,234,663.69	224,355,769.55
4	**W&E**	755,491,062.97	785,988,538.61	780,574,857.82	742,430,361.84	**1,044,802,160.50**	821,536,674.24	729,455,894.68	725,343,469.44	726,398,443.98
5	**W&E**	921,703,241.26	1,396,563,131.34	1,098,682,569.68	914,687,851.76	**1,514,877,550.42**	1,085,748,624.36	915,779,475.67	1,462,045,002.05	1,020,322,186.01
6	**J&E**	851,827,235.94	**919,424,412.72**	851,246,006.74	664,830,289.30	853,422,248.55	755,259,837.34	666,715,248.93	666,985,610.03	666,988,252.21
7	**J&E**	1,011,399,241.82	**1,841,263,491.63**	1,294,038,404.90	983,890,885.72	1,798,683,542.18	1,249,317,430.48	860,861,459.43	911,440,947.64	863,542,440.58
8	**W&I**	1,084,840,725.10	1,076,130,862.96	832,181,918.66	**1,081,288,421.72**	946,526,642.25	821,053,607.15	751,805,120.20	755,408,694.42	746,073,006.37
9	**J&E**	371,802,505.42	**700,329,417.24**	552,151,283.35	371,833,723.59	652,272,377.45	590,239,249.90	372,020,753.28	389,253,840.67	499,413,293.83
10	**W&E**	547,855,988.86	626,189,114.09	430,325,794.89	556,466,379.96	**706,286,206.09**	504,552,633.00	82,528,902.67	87,540,031.14	86,568,602.03
11	**J&E**	812,472,375.33	**1,070,308,761.17**	786,914,578.52	752,285,518.71	1,046,159,694.09	797,506,972.55	669,216,426.43	866,787,122.98	739,841,393.93
12	**S&E**	125,841,471.35	155,857,220.71	125,506,183.50	112,022,493.22	149,835,138.09	119,497,823.49	112,543,736.25	**176,564,542.57**	113,905,837.23
13	**W&P**	232,491,213.85	244,123,259.04	249,654,814.29	217,398,569.58	239,652,391.16	**249,733,109.41**	150,126,706.54	150,126,485.11	152,368,221.21
14	**J&E**	28,575,990.41	**33,812,977.20**	27,768,014.04	21,862,895.46	31,422,186.87	28,785,988.40	21,694,200.14	24,633,128.36	27,989,915.98
15	**S&E**	59,526,993.73	75,043,180.69	62,064,591.82	59,232,611.50	68,912,202.08	57,183,141.35	69,977,164.21	**76,084,431.90**	64,393,566.83
16	**J&I**	**703,744,024.40**	522,725,361.57	502,475,192.63	586,039,740.32	463,948,521.31	516,977,735.38	533,852,133.06	525,268,060.61	519,667,489.17
17	**J&E**	186,315,107.88	**248,770,727.64**	233,411,462.03	183,469,048.77	229,433,533.00	212,435,439.37	182,732,518.40	188,354,789.00	188,114,142.67
18	**W&E**	240,527,373.95	342,536,102.62	359,172,274.09	138,973,669.15	**573,785,694.93**	528,223,562.11	93,441,452.83	138,324,077.91	109,581,531.22
19	**J&E**	653,306,298.06	**820,057,007.22**	694,284,176.70	689,376,026.93	818,620,591.36	706,888,094.24	622,057,308.17	767,800,378.36	626,572,893.25
20	**S&E**	123,811,206.30	155,577,749.99	150,071,038.53	127,738,525.72	153,483,479.59	162,962,871.21	110,381,037.37	**168,985,902.50**	124,162,680.47
μ	W&E	566,725,482.84	733,744,517.17	589,802,255.03	539,015,100.33	**752,214,908.36**	599,071,627.29	475,231,003.69	551,649,317.25	493,461,067.61

Table 10: DGC@50 in the movies domain

#	>	J&I	J&E	J&P	W&I	W&E	W&P	S&I	S&E	S&P
21	**J&E**	101.7179125	**109.1541017**	104.3419068	101.4510003	108.5850107	102.2158939	102.0210924	108.0105891	103.3331381
22	**J&E**	106.6521713	**109.0513266**	105.9637564	106.276836	108.7539213	106.1621848	104.578776	108.2741378	104.1508907
23	**W&E**	108.1722847	108.0604545	103.709266	108.0420709	**108.3788082**	105.4364741	103.2010738	104.4358906	100.8668424
24	**W&E**	101.7096559	103.7847925	102.4639776	102.3425463	**104.0055413**	103.055964	102.4149949	102.5335141	102.820858
25	**W&E**	104.0736712	106.9539711	104.8887741	102.7234664	**107.2300199**	104.7580523	100.7321834	104.6730483	103.7103855
26	**J&E**	102.0311325	**103.1782995**	101.957734	101.8840695	102.9303069	100.9014823	100.9806091	99.31307808	99.70808772
27	**S&E**	97.23690249	100.4068862	97.86047221	96.84653007	100.4068862	97.73147766	100.7031477	**103.1472917**	100.9662278
28	**J&I**	**102.0472326**	101.7916208	101.9990848	99.63941859	101.6401097	100.3995797	100.9133985	100.2252951	99.22522337
29	**J&I**	**100.9532697**	99.55538396	99.87624295	100.2161413	99.98497721	100.1163419	98.92596983	95.92275889	96.82610409
30	**J&P**	82.61289874	81.25409495	**84.0045402**	82.84323864	81.18492123	83.2172422	81.66736088	81.12845478	78.489955
31	**W&E**	102.6135246	**106.4563473**	103.0232158	102.6079602	**106.5278606**	103.8627853	102.5392181	105.9714704	102.7870055
32	**J&E**	97.27891418	100.5423449	98.36869698	97.63675066	98.16200402	97.92520742	98.01975344	98.6239517	98.36668486
33	**S&E**	107.4761682	108.1942068	106.525463	107.7203443	109.2114947	106.9146716	106.6664435	**110.9466064**	106.3331044
34	**S&E**	101.2091958	103.7449972	101.5170346	100.4245276	103.9013642	101.193589	102.0804978	**104.2773939**	102.1652831
35	**W&E**	102.002413	**104.3030497**	101.0468399	102.1708923	**104.5389308**	101.2802103	103.2740652	104.1158341	102.3060204
36	**J&E**	104.2643337	**105.2305254**	103.7788353	97.52554569	98.24146754	96.81223765	102.3718547	103.3116149	102.165056
37	**J&E**	103.3063993	104.3369872	101.4630031	92.6505531	94.45138953	92.55124446	102.5503218	103.2709757	102.5226448
38	**S&E**	100.5040952	101.017853	101.3986788	100.3946898	100.5801129	100.9378317	101.3091842	**101.7931757**	99.564
39	**J&E**	105.5153047	**106.1661686**	102.8360103	105.7322839	105.8648132	102.927369	102.7563709	104.0726017	99.23865507
40	**W&P**	103.6087108	105.4994412	105.3910246	104.0285533	105.4994412	105.508895	104.4004286	105.286901	105.3021323
M	**J&E**	101.74931	**103.43414**	101.62073	100.65787	102.50397	100.69544	101.10534	102.46673	100.54241

Table 11. Pairwise Comparison in the music domain

	J&I	J&E	J&P	W&I	W&E	W&P	S&I	S&E	S&P
J&I		0.0094	0.4140	0.0370	0.0085	0.2913	0.0044	0.7404	0.0301
J&E			0.0026	0.0039	**0.3925**	0.0080	0.0006	0.0056	0.0009
J&P				0.1035	0.0018	0.4207	0.0007	0.3238	0.0025
W&I					0.0037	0.0696	0.0320	0.7751	01496
W&E						0.0031	0.0005	0.0036	0.0008
W&P							0.0011	0.2641	0.0031
S&I								0.0161	0.0481
S&E									0.0326
S&P									

Table 12: Pairwise Comparison in the movies domain

	J&I	J&E	J&P	W&I	W&E	W&P	S&I	S&E	S&P
J&I		0.001306	0.7262	0.09354	0.3344	0.1399	0.1458	0.2921	0.06771
J&E			0.000296	0.000962	**0.1382**	0.000984	9.44E-05	0.02804	4.33E-06
J&P				0.1469	0.2178	0.1225	0.1546	0.1448	0.02524
W&I					0.000509	0.8934	0.5273	0.04159	0.8948
W&E						2.05E-04	0.08179	0.9569	0.02118
W&P							0.5502	0.034	0.843
S&I								0.01081	0.1245
S&E									0.000132
S&P									

5.3.1. Comparison of the Best Search Strategy with the Baselines

We recall that the Q&I and Q&E strategies are the baselines and that Q&I corresponds to RECAP (PIRRÒ, 2015). Also, J&E was the preferred strategy. Tables 13 and 14 then compare the J&E, Q&I and Q&E strategies for the music and movies domains, respectively.

In the music domain, the average DCG score of Q&I is less than half of the score obtained by J&E, and in the movies domain the average DCG score of J&E

is also greater that Q&I. In the music domain, the average DCG score of Q&E is higher that the score of Q&I. In the movies domain Q&E shows a slight improvement in the average DCG score, when compared with Q&I.

In a pairwise comparison, J&E and Q&I have statistically significant differences in the two domains (p = 0.0240 in music, and p = 2.24E-08 in movies).

However, in the music domain, J&E does not have statistically significant differences with Q&E ($p < 0.05$), with p = 0.0695. In addition, in the movies domain, J&E has statistically significant differences with Q&I and Q&E, with p = 2.24E-08 and p = 3.10E-08, respectively.

Therefore, we may conclude that the J&E strategy performs better than the baselines, for the music and movies domains. We may also conclude that Q&E – classical SPARQL queries with the Exclusivity-based Relatedness path ranking measure – performs better than Q&I – classical SPARQL queries with the Predicate-Frequency Inverse-Triple-Frequency measure (the original RECAP strategy), for the music and movies domains.

Table 13: Comparing with baseline in the music domain

#	J&E	Q&I	Q&E
1	633,392,974.66	444,807,776.86	480,907,819.25
2	2,753,779,539.02	15,195,889.92	38,943,310.32
3	273,016,513.27	250,516,706.72	255,314,533.68
4	785,988,538.61	557,170,887.23	685,358,058.40
5	1,396,563,131.34	9,495,218.29	24,480,158.16
6	919,424,412.72	776,557,816.12	824,168,944.87
7	1,841,263,491.63	62,933,086.19	116,693,570.,40
8	1,076,130,862.96	856,119,129.32	961,706,389.52
9	700,329,417.24	472,047,465.13	589,643,590.53
10	626,189,114.09	562,671,912.47	611,756,486.33
11	1,070,308,761.17	133,712,886.64	169,574,334.16
12	155,857,220.71	23,657,563.47	28,230,501.03
13	244,123,259.04	562,836,077.49	263,535,631.88
14	33,812,977.20	32,321,213.20	24,922,943.78
15	75,043,180.69	62,446,723.43	17,232,752.72
16	522,725,361.57	579,330,640.47	529,932,360.79
17	248,770,727.64	4,618,945.53	41,961,802.00
18	342,536,102.62	144,299,854.08	1,101,761,151.45
19	820,057,007.22	931,583,723.23	1,177,262,059.20
20	155,577,749.99	4,178,125.72	530,036.16
μ	733,744,517.17	324,325,082.08	397,195,821.73

Table 14: Comparing with baseline in the movies domain

#	J&E	Q&I	Q&E
21	109.1541017	74.04989766	69.46827321
22	109.0513266	17.66088243	14.6878794
23	108.0604545	71.44311491	71.52806626
24	103.7847925	79.17713582	75.30431868
25	106.9539711	91.25503521	90.76558024
26	103.1782995	61.27873807	63.56980208
27	100.4068862	66.84597973	61.95518338
28	101.7916208	62.25771412	56.40864151
29	99.55538396	50.62729921	53.11990538
30	81.25409495	72.86037327	71.33436023
31	106.4563473	78.80105822	72.95852578
32	100.5423449	51.69226531	84.10840061
33	108.1942068	65.86269644	67.94526909
34	103.7449972	84.81788749	82.34229576
35	104.3030497	85.92968156	87.18178703
36	105.2305254	54.01190478	52.89485027
37	104.3369872	64.18796705	66.89397555
38	101.017853	72.35328389	70.87365334
39	106.1661686	78.41639509	73.59506078
40	105.4994412	69.37020673	67.93328785
μ	103.43414	67.64498	67.74346

5.3.2. Discussion

In this section, we analyze examples that explain the results obtained in Tables 9, 10, 13 and 14, and discuss the advantages and disadvantages of the search strategies studied in this chapter.

5.3.2.1. High Centrality and Generic Information

As defined in Section 5.3.3, we adopted two baselines using SPARQL Queries to find relationship paths in the movies and music domains.

The baselines, Q&I and Q&E, use a set of SPARQL queries that are sent to DBpedia, which resolves the queries and prioritizes some immediate results. The results obtained have a direct relation with the restrictions of DBpedia, the request time limit and the size of query result. Short paths are prioritized and some

relationships generated are rather confusing and of little significance. Some short examples help understand this situation.

Some paths recovered are composed of generic information that does not explain why the entity pair are connected. An example is the following RDF path, which links dbr:Cary_Grant and dbr:Katharine_Hepburn, and is composed of entity types:

dbr:Cary_Grant **rdf:type** dbo:Actor .
dbo:Actor **^rdf:type** dbr:Katharine_Hepburn .

Another source of problem are entities with high centrality. For example, dbr:United_States is an entity with 16,928 links in DBpedia. In fact, this entity is a hub that does not clearly express, for example, why dbr:Dorothy_Gale and dbr:Rip_Taylor are connected. This is illustrated in the following RDF path:

dbr:Judy_Garland **^dbp:portrayer** dbr:Dorothy_Gale .
dbr:Dorothy_Gale **dbp:nationality** dbr:United_States .
dbr:United_States **^dbp:nationality** dbr:Rip_Taylor .
dbr:Rip_Taylor **dbp:influences** dbr:Mickey_Rooney .

In the Music domain, there are similar situations. The record companies are entities with high centrality and hardly lead to obvious explanations. The next RDF path illustrate this problem:

dbr:Led_Zeppelin **^dbp:associatedActs** dbr:John_Entwistle .
dbr:John_Entwistle **dbp:label** dbr:Track_Records .
dbr:Track_Records **dbp:founder** dbr:Chris_Stamp .
dbr:Chris_Stamp **dbp:associatedActs** dbr:The_Who .

Also, awards and nominations do not clearly define a relationship between entities. For example, in the following RDF path, dbr:Grammy_Nominees does not have relevant information that explains why dbr:U2 and dbr:R.E.M. are related:

dbr:U2 **^dbp:music** dbr:Grammy_Nominees .
dbr:Grammy_Nominees **dbp:music** dbr:Mary_Chapin_Carpenter .
dbr:Mary_Chapin_Carpenter **dbp:associatedActs** dbr:Indigo_Girls .
dbr:Indigo_Girls **dbp:associatedMusicalArtist** dbr:R.E.M .

5.3.2.2. Ideal Path

There are relationship paths that belong to the music or movies domains but which cannot be found by search strategies that use an activation function, as those defined

in Section 5.2. These paths are composed of influential entities in the domain but with high centrality.

For example, the following RDF path connects dbr:Bob_Dylan and dbr:The_Band and is composed of dbr:The_Beastles and dbr:Ringo_Starr,

dbr:Bob_Dylan **^dbp:pastMembers** dbr:Traveling_Wilburys .
dbr:Traveling_Wilburys **dbp:associatedBand**, dbr:The_Beatles .
dbr:The_Beatles **dbp:formerBandMember**, dbr:Ringo_Starr .
dbr:Ringo_Starr **dbp:associatedMusicalArtist** dbr:The_Band .

But dbr:The_Beatles and dbr:Ringo_Starr are influential entities in the music domain. Indeed, last.fm reports 479 million and 3,1 million play counts for The Beatles and Ringo Starr, respectively. However, we detected 1,198 and 564 links (as a centrality measure) for The Beatles and Ringo Starr, respectively. In fact, the search strategies that use as activation criterion the number of links do not generate the above path.

The following path, between dbr:Jay_Z and dbr:Kanye_West, has similar characteristics:

dbr:Jay_Z **^dbp:associatedBand** dbr:Linkin_Park .
dbr:Linkin_Park **^dbp:after** dbr:Nicki_Minaj .
dbr:Nicki_Minaj **^dbp:writer** dbr:Monster_(song) .
dbr:Monster_(song) **dbp:writer** dbr:Kanye_West .

In last.fm, Linkin Park and Nicki Minaj have 265,4 million and 54,6 million play counts, respectively, and we detected 550 and 542 links as the centrality of Linkin Park and Nicki Minaj, respectively.

To summarize, we presented simple cases where, in multi-domain bases such as DBpedia, the similarity measures and the individual centrality of an entity can be relevant criteria to leave out entities with generic information in the generation of meaningful paths. However, in specific domains, such as music or movies, the search strategies need extra information to understand the connectivity between the entities, as PageRank or Hubs and Authorities, which provide a global importance of each entity in the knowledge base.

5.4. Conclusions

We combined entity similarity and path ranking measures to generate search strategies to find meaningful relationship paths between entities in a knowledge base. We created a dataset of entity pairs in the movies and music domains and found the relationship paths, for each pair in these domains, though the Movie and Music ontology. We generated reliable relationship path rankings using the information found in IMDb and last.fm to create a ground truth, in a semi-automated process. We used the DCG measure to compare the quality of the search strategies on two entertainment domains and discussed their differences. We created two baselines using classic SPARQL queries, which follows the pathfinding process defined in (PIRRÒ, 2015).

The search strategies are applied using the same pathfinding process, orchestrated by the backward heuristic. The results reveal that the search strategy composed of the Jaccard similarity and the Exclusivity-based Relatedness ranking measure has a favorable behavior in the recommendation of relationship paths in the two entertainment domains, the best of the 9 search strategies compared, and with much better results than the baselines.

The construction process of the Ground Truth can be replicated to other domains where, intuitively: (1) entities with high reputation help select "meaningful" paths; (2) less frequent properties, or more discriminatory properties, also help select "meaningful" paths. In fact, the construction process described in Section 5.4.2 is as interesting as the resulting Ground Truth.

Future work will focus on other domains, such as Sport, Video Games or Academic Publication to create specialized rankings, to increase the size of the entity pair dataset and enhance the results of this work. Additionally, we plan to test the search strategies in other knowledge bases, such as Wikidata, and analyze the processing times of the search strategies.

6
Conclusions and Future Work

In this thesis, we focused on the development of search strategies to address the entity relatedness problem in knowledge bases in RDF format. In order to show the potential usefulness of the research, we proposed strategies that explore the connectivity between entities, studied Web-based approaches that mine knowledge bases and evaluated our contributions through user participation and experimental tests.

In Chapter 4, we addressed the problem of how and why entities are related in a knowledge base. Although substantial research attempted to solve this problem, most of the work focused on specific parts of the solution. A few approaches were concerned with generating a ranked list of the most meaningful relationship paths (HEIM *et al*., 2009, CHENG *et al*., 2013), some approaches generated complex graphs, but without explanations (FANG *et al*., 2011), while others described tools that are not available online to carry out effective comparisons (PIRRÒ, 2015). Therefore, we introduced a strategy to generate connectivity profiles for entity pairs represented in a knowledge base, and developed the DBpedia profiler tool, which implements the proposed strategy.

The connectivity profile strategy implemented in the DBpedia Profiler tool was compared with the state-of-the-art tool RECAP (PIRRÒ, 2015). The experiments shows that DBpedia Profiler outperforms RECAP in terms of performance and usability. The performance behavior of DBpedia Profiler, with respect to a user query in real-time, was ~20 seconds, on the average, which compares favorably with the ~34 seconds of RECAP. This can be explained in part because RECAP uses an expensive process to create relationship paths (FANG *et al*., 2011). In the user evaluation, the results of the proposed questionnaire reveal that DBpedia Profiler provides a more satisfactory user experience, where the mean score of DBpedia Profiler was 3.95, which again compares favorably with 3.01 score of RECAP, and the value of the standard deviation scores of DBpedia Profiler

captured a stable behavior in solving the search tasks by the end users. The users expressed their satisfaction with DBpedia Profiler to reach the expected results through the semantic annotations, particularly with the usage of topic categories, which provide a simple way to explain the connectivity between the entities.

In Chapter 5, we put into practice the lessons learned from the previous chapter to explore the connectivity between two entities. In general, pathfinding processes found in the literature on RDF knowledge bases use SPARQL queries (HEIN *et al*., 2009; CHENG *et al*., 2014; PIRRÒ, 2015; HERRERA *et al*., 2017). Typically, querying knowledge bases via SPARQL queries requires familiarity with SPARQL syntax and the structure of the underlying knowledge base, because that the data can be published in different ways (FÄRBER, *et al.* 2015). Therefore, we adopted a backward search heuristic with the aim of covering any RDF graph of an online knowledge base. In the backward search, the expansion process uses simple HTTP requests to recover the entity description. The expansion process uses an entity similarity measure and the entity degree to prioritize certain paths over others.

Moreover, there currently is no way to measure the effectiveness of the search strategies that explore the connectivity between entities automatically. The approaches available in the literature (CHENG *et al*., 2014; NUNES *et al.*; 2013; PIRRÒ, 2015; HULPUS *et al*., 2015; HERRERA *et al*., 2017) widely adopted user valuations to compare the effectiveness of the proposed methods, but the evaluations do not clearly define the capabilities of the approaches analyzed. To address this challenge, in Chapter 5, we created a ground truth from two entertainment domains to compare the search strategies that explore the connectivity between entities.

In Chapter 5, we combined similarity and ranking measures in the same search process, orchestrated by the backward heuristic, with the aim of creating search strategies and finding relevant relationship paths between two entities. We applied the search strategies to DBpedia and compared the relationship paths found with a ground truth, consisting of a set of entity pairs and, for each pair, a set of relationship paths, ranked using domain-specific knowledge. The last.fm and Internet Movie Database (IMDb) databases were used to generate specialized lists of relationship paths to the given entity pairs in the music and the movies domains, respectively.

The comparison among the search strategies reveals that the "J&E" strategy composed of the Jaccard similarity and the Exclusivity-based Relatedness ranking measure has a favorable behavior in the recommendation of the relevant relationship paths in the two entertainment domains. We compared J&E with two baselines, the first one generated using the approach proposed by (PIRRÒ, 2015) and the second generated using the Exclusivity-based Relatedness ranking measure. The results of the last comparison show that, "J&E" executes the search tasks with better DCG scores than the baselines. Finally, we conclude that the search strategies based on the similarity measures and the individual centrality of an entity can be interesting alternatives to generate relevant paths in multi-domain bases, such as DBpedia. However, there are situations which require extra information to understand the connectivity between the entities.

During the development of this research, we identified the following opportunities for future work:

i) Complementing the evaluation of the strategy to generate Connectivity Profiles, proposed in the Chapter 4, with the Ground Truth introduced in the Chapter 5. The semantic explanation patterns can be compared individually based on the meaning of the paths recovered by each explanation, and thus create a comparison framework to evaluate explanation patterns.
ii) Developing a framework for the entity relatedness problem, considering as basic hot-spots the similarity measure between entities and the ranking-path measure to identify relevant paths.
iii) Node similarity is a method that provides information about the structure of any given network, and it has been well studied (ZHOU, *et al*., 2009; THIEL, *et al*., 2010; YUAN, *et al*., 2012). The last work was concerned with maximizing the similarity score in a subgraph, to determine spreading activation patterns between the nodes of a network. These works provide the opportunities to optimize the activation criteria in the pathfinding process of the backward search heuristic.
iv) A recurring need in the evaluation of search strategies to explore entity connectivity is the generation of a reliable ground truth. We will continue to improve the ground truth proposed in this thesis, which includes

supporting pathfinding processes on any knowledge base and incrementing our entity pair datasets for different domains.

Bibliography

ASSIS, Pedro *et al*. Improving Relation Extraction by Using an Ontology Class Hierarchy Feature. **Web Information Systems Engineering**, v. 9419, p. 241-249, 2015.

BALMIN, Andrey; HRISTIDIS, Vagelis; PAPAKONSTANTINOU, Yannis. Objectrank: Authority-based keyword search in databases. **VLDB Endowment**, v. 30, p. 564–575, 2004.

BARBIERI, Davide *et al*. Querying rdf streams with C-SPARQL. **SIGMOD Record**, v. 39, n. 1, p. 20-26, 2010.

BERNERS-LEE, Tim. Design Issues: Linked Data. 2006. Disponível em: <http://www.w3.org/DesignIssues/LinkedData.html>. Acesso em : 30 mar. 2017.

BIZER, Christian; HEATH, Tom; BERNERS-LEE, Tim. Linked data - the story so far. **Journal on Semantic Web and Information Systems**, v. 5, n. 3, p. 1-22, 2009.

BÖHM, Christoph; GJERGJI, Kasneci, NAUMANN, Felix. Latent topics in graph-structured data. **CIKM**, p. 2663-2666, 2012.

BOLLES, Andre; GRAWUNDER, Marco; JACOBI, Jonas. Streaming SPARQL - extending SPARQL to pro-cess data streams. **In European Semantic Web Conference**, v. 5021, p. 448-462, 2008.

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

BUNKE, Horst; SHEARERB Shearerb. A graph distance metric based on the maximal common subgraph. **Pattern Recognition Letters**, v. 19, p. 255–259, 1998.

CHENG, Gong; ZHANG, Yanan; QU, Yuzhong. Explass: Exploring Associations between Entities via Top-K Ontological Patterns and Facets. **International Semantic Web Conference**, v. 8797, pp. 422-437, 2014.

CHURCH, Kenneth; HANKs, Patrick. Word association norms, mutual information, and lexicogra-phy. **Computational Linguistics**, v. 16, n. 1, p 22-29, 1990.

COHEN, William W. Graph walks and graphical models. Tech. Report CMU-ML-10-102, School Comp. Sci., Carnegie Mellon Univ., 2010.

CYGANIAK, Richard; WOOD, David; LANTHALER, Markus. **RDF 1.1 Concepts and Abstract Syntax**. 2014. Disponível em: <http://www.w3.org/TR/rdf11-concepts/>. Acesso em : 30 mar. 2017.

DE VOCHT, Laurens *et al*. Discovering Meaningful Connections between Resources in the Web of Data. **In:LDOW CEUR-WS.org**, v. 996, 2013.

DE VOCHT, Laurens *et al*. Effect of Heuristics on Serendipity in Path-Based Storytelling with Linked Data. **International Conference on Human Interface and the Management of Information**,v. 9734, p. 238-251, 2016

DONG, Xin, *et al*. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. **SIGKDD**, p. 601–610, 2014.

EHRIG, Marc *et al*. Similarity for Ontologies - A Comprehensive Framework. **European Conference on Information Systems**, p. 1509-1518, 2005.

FANG, Lujun *et al*. REX: Explaining Relationships be-tween Entity Pairs. **PVLDB**, v. 5, n. 3, p. 241-252. 2011.

FÄRBER, Michael *et al*.. A Comparative Survey of DBpedia, Freebase, OpenCyc, Wikidata. and YAGO. **Semantic Web Journal**, 2015.

FRANKLIN, Michael; HALEVY, ALON; MAIER, David. From databases to dataspaces: A new abstraction for information management. **SIGMOD Record**, v. 34, n. 4, p. 27–33, 2005.

FRANZ, Thomas *et al*. Triplerank: Ranking semantic web data by tensor decomposition. **International Semantic Web Conference**, v. 5823, p. 213–228, 2009.

GUESSOUM, Djamel; MIRAOUI, Moeiz; TADJ, Chakib. Survey of semantic similarity measures in pervasive computing. **International Journal on Smart Sensing and Intelligent Systems**, v. 8 n. 1, p. 125-158, 2015 .

HARISPE, Sébastien *et al*. Semantic similarity from natural language and ontology analysis. **Synthesis Lectures on Human Language Technologies**, v. 8, n. 1, p. 1-254, 2015.

HARRIS, Steve; SEABORNE, Andy SPARQL1.1 **Query Language W3C Recommendation**, 2013. Disponível em: <http://www.w3.org/TR/sparql11-overview/>. Acesso em : 30 mar. 2017.

HARTIG, Olaf, BIZER, Christian and FREYTAG, Johann-Christoph. Executing sparql queries over the Web of Linked Data. **International Semantic Web Conference**, v. 5823, p. 293–309, 2009.

HEIM, Philipp *et al*. RelFinder: Revealing Relationships in RDF Knowledge Bases. **Proceedings of the 3rd International Conference on Semantic and Media Technologies**, v. 5887, p. 182-187, 2009.

HERRERA, José *et al*. DBpedia Profiler Tool: Profiling the Connectivity of Entity Pairs in DBpedia. **Proceedings of the Intelligent Exploration of Semantic Data**, 2016.

HERRERA, José *et al*. **Entity Relatedness Test Dataset** - V2. figshare. Disponível em: https://doi.org/10.6084/m9.figshare.5007983.v1. Acesso em: 15 May. 2017.

HULPUS, Ioana; PRANGNAWARAT, Narumol; HAYES, Conor. Path-based Semantic Relatedness on Linked Data and its use to Word and Entity Disambiguation. **The Semantic Web Conference**. v. 9366, p. 442-457, 2015.

JÄRVELIN, Kalervo; KEKÄLÄINEN, Jaana. Cumulated gain-based evaluation of IR techniques. **ACM Transactions on Information Systems**. v. 20, p. 422-446, 2002.

JEH, Glen; WIDOM, Jennifer. Simrank: A measure of structural context similarity. **In KDD**, p. 538–543, 2002.

KALERVO, Järvelin, KEKÄLÄINEN, Jaana. Cumulated gain-based evaluation of IR techniques. **ACM Transactions on Information Systems**, v.20, n. 4, p 422-446, 2002.

KATZ, Leo. A new status index derived from sociometric analysis. **Psychometrika**, v. 18, n. 1, p. 39–43, 1953.

KAWASE, Ricardo *et al*. Exploiting the wisdom of the crowds for characterizing and connecting heterogeneous resources. **Conference on Hypertext and Social Media**, p. 56-65, 2014.

KLEINBERG, Jon M. Authoritative sources in a hyperlinked environment. **Journal of the ACM**, v. 46, n. 5, p. 604–632. 1999.

LE, Wangchao et al. Scalable keyword search on large RDF data. **IEEE Trans. Services Computing**, v. 26, n. 11, p. 2774-2788, 2014.

LEHMANN, Jens *et al*. DBpedia - a large-scale, multilingual knowledge base extracted from Wikipedia. **Semantic Web Journal**, v. 6, n. 2, p. 167-195, 2015.

LEVENSHTEIN, Vladimir. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. **Soviet Physics Doklady**, v. 10, n. 8, p. 707-710, 1966.

MENG, Changping *et al*. Discovering meta-paths in large heterogeneous information networks. **World Wide Web**, p. 754–764, 2015.

MILNE David; WITTEN Ian H. An efective, low-cost measure of semantic relatedness obtained from wikipedia links. **In Proceedings of AAAI**, p. 25-30, 2008.

MOHAN, Yang *et al*. Finding patterns in a knowledge base using keywords to compose table answers. **PVLDB**, v. 7, n. 14, p 1809-1820, 2014.

MOORE, Joshua; STEINKE, Florian; TRESP, Volker. A Novel Metric for Information Retrieval in Semantic Networks. **In European Semantic Web Conference Workshops**, v.7117, p. 65-79, 2011.

NGUYEN Nguyen *et al*. Content-based recommendations via DBpedia and Freebase: a case study in the music domain. **International Semantic Web Conference**, v. 9366, p. 605-621, 2015.

NUNES, Bernardo *et al*. Combining a co-occurrence based and a semantic measure for entity linking. **In European Semantic Web Conference**, v. 7882, 2013.

NUNES, Bernardo *et al*. S. SCS connector - Quantifying and visualising semantic paths between entity Pairs. **The Semantic Web: ESWC Satellite Events**, v. 8798, 2014.

PASSANT, Alexandre. Dbrec: Music Recommendations Using DBpedia. **International Semantic Web Conference**, v. 6497 p. 209-224, 2010.

PIAO, Guangyuan; ARA, Safina showkat; BRESLIN, John G. Computing the Semantic Similarity of Resources in DBpedia for Recommendation Purposes. **Joint**

International Semantic Technology Conference, v. 9544, p. 185-200, 2015

PIRRÒ, Giuseppe. Explaining and Suggesting Relatedness in Knowledge Graphs. **International Semantic Web Conference**, v. 9366, p. 622-639, 2015.

POUND, Jeffrey; MIKA, Peter; ZARAGOZA. Hugo. Ad-hoc object retrieval in the web of data. **World Wide Web**, p. 771-780, 2010.

QIAOZHU, Mei et al. Generating semantic annotations for fre-quent patterns with context analysis. **SIGKDD international conference on Knowledge discovery and data mining**, p. 337-346, 2006.

THIEL, Kilian; BERTHOLD, Michael R. Node Similarities from Spreading Activation. International Conference on Data Mining, pp. 1085-1090, 2010.

WATTS, Duncan J.; STROGATZ, Steven. Collective dynamics of 'small-world' networks. **Nature**, v. 393, n. 6684, p. 440–442, 1998.

YUAN, Ye et al. Efficient Subgraph Similarity Search on Large Probabilistic Graph Databases. VLDB, v. 5, n. 9, pp. 800-811, 2012.

ZHOU, Yang; CHENG, Hong; XU YU, Jeffrey. Graph clustering based on structural/attribute similarities. VLDB, v. 2, n. 1, pp. 718-729, 2009.

7
Appendix

7.1.
Methodology to Evaluate Exploration Tools

In this section, we briefly describe the methodology adopted to evaluate the exploration tools, based on (FANG *et al*., 2011), and executed in Chapter 4.

This methodology basically defines an entity pairs set, a questionnaire for the user and a procedure to evaluate the exploration tools. The entity pairs are listed in Table 15, and the questions to create the questionnaire are described in Table 16. The procedure uses the entity pairs and the questionnaire, and it is used to evaluate and compare exploration tools.

In Chapter 4, the entities pairs set is used in the performance evaluation, as described in Section 4.4.1. In the user evaluation, the procedure is executed to compare the exploration tools, as described in Section 4.4.2. We describe this procedure in the what follows.

The evaluation involves users with some experience with Semantic Web technology. Each participant (user) is assigned 6 entity pairs, selected randomly from the Table 15. Prior to the experiment, the participants had a training stage to familiarize themselves with the tools.

For each entity pair, the user must execute a search task using the exploration tools. The search task adopted to evaluate the tools is defined as follows:

"*Suppose you will write an article about the associations between X and Y. Use the given system to explore their associations and identify several themes to discuss in the article*."

where (*X*,*Y*) represents an entity pair. For each tool to be evaluated, the user must execute the search task within a maximum of 10 minutes.

After the user execute a search task in each exploratory tool, he must answer the questions defined in the Table 16. The response to each question was in the form

of an agreement value, ranging from 1 (min) to 5 (max). Questions Q1-Q5 were defined in (FANG *et al*., 2011), to which we added Questions Q6 and Q7.

Table 15: Entity Pairs to generate the Search Tasks

#	Entity Pair	#	Entity Pair
1	Ingrid_Bergman, Isabella_Rossellini	**14**	Danielle_Steel, Nora_Roberts
2	John_F._Kennedy, Jacqueline_Kennedy_Onassis	**15**	Richard_Gere, Carey_Lowell
3	Barack_Obama, George_W._Bush	**16**	Leonardo_DiCaprio, Kate_Winslet
4	Walt_Disney, Roy_O._Disney	**17**	Capcom, Sega
5	Julia_Roberts, Emma_Roberts	**18**	Aldi, Lidl
6	Garry_Marshall, Héctor_Elizondo	**19**	Universal_Studios, Paramount_Pictures
7	Andrew_Jackson, John_Quincy_Adams	**20**	IBM, Hewlett-Packard
8	Frank_Herbert, Brian_Herbert	**21**	Last_Action_Hero, Terminator_2:_Judgment_Day
9	Bruce_Carver, Christopher_Jones	**22**	Forbes, Bloomberg_L.P.
10	Abraham_Lincoln, George_Washington	**23**	Charmed, Buffy_the_Vampire_Slayer
11	Tom_Cruise, Katie_Holmes	**24**	The_Sopranos, Law_&_Order
12	Christian_Bale , Christopher_Nolan	**25**	Nanga_Parbat, Broad_Peak
13	Hal_Roach, Stan_Laurel	**26**	Manhattan_Bridge, Brooklyn_Bridge

Table 16: Questions and Responses about Exploration Effectiveness.

Question	Description
Q1: Information overview	The system helped me get an overview of all the information
Q2: Easiness in finding information	The system helped me easily find information relevant to this task.
Q3: Easiness in comparing/synthesizing info	The system helped me easily compare and synthesize all kinds of relevant information.
Q4: Comprehensive support	The system provided me with much support for carrying out this task.
Q5: Sufficient support to the task	The system provided me with sufficient support for carrying out this task (Different from Q4, this question targets the functions that are expected but missing).
Q6: Redundancy in information	The system provided redundancy information. We assumed that redundant information does not help the user in the exploration.
Q7: General Comments about the task and tool	Allowed the user to make a qualitative evaluation on the tool

7.2. Ground Truth Summarized

For each of the 40 entity pairs of our entity pairs dataset, we used the path finding algorithm, described in Section 5.2.2.2, to create 40 sets, each with 50 relationship

paths. The dataset is summarized in **Error! Not a valid bookmark self-reference.**7, and it is available at (HERRERA *et al.*, 2017).

Table 17: Ground Truth Description.

Contents (for each domain)	Description
<domain>_entity_pairs	List of 20 entity pairs for the domain
<domain>_ranked_paths	For each entity pair, a ranked list of relationship path
<domain>_class_mappings	Mappings between DBpedia classes and domain classes
<domain>_entity_mappings	Mappings between DBpedia entities and domain entities
<domain>_entity_scores	Entity scores computed from the domain reference dataset
<domain>_property_relevance_scores	Property relevance scores (ITF) computed from DBpedia

7.3. Examples of Path Ranking using the Ground Truth

In the examples that follow, we considered the structure "ID, Path description, score" to represent the lists of the relationships paths between the entities. For simplicity, we avoid showing the namespace of the URIs of the entities.

Example 6: Top-20 relationship paths between Michael Jackson and Whitney Houston generated by Jaccard Distance & Exclusivity-based Relatedness (J&E) in the Music Domain.

ID	PATH DESCRIPTION	SCORE
1	"Michael_Jackson ^influencedBy Jacky_Cheung ^influencedBy Raymond_Lam influences Whitney_Houston"	0.055555556
2	"Michael_Jackson ^producer The_Simpsons_Sing_the_Blues producer Bryan_Loren associatedActs Whitney_Houston"	0.014492754
3	"Michael_Jackson ^producer Thriller_(Michael_Jackson_album) ^album Human_Nature_(Michael_Jackson_song) writer John_Bettis associatedMusicalArtist Whitney_Houston"	0.012048193
4	"Michael_Jackson ^producer Thriller_(Michael_Jackson_album) ^album Human_Nature_(Michael_Jackson_song) writer John_Bettis associatedBand Whitney_Houston"	0.012048193
5	"Michael_Jackson ^associatedActs Jermaine_Jackson ^producer Whitney_Dancin'_Special previousWork Whitney_(album) artist Whitney_Houston"	0.010989011

6	"Michael_Jackson ^associatedActs Jermaine_Jackson ^producer Whitney_Dancin'_Special ^subsequentWork Whitney_Houston_(album) artist Whitney_Houston"	0.010989011
7	"Michael_Jackson ^producer Love_Never_Felt_So_Good producer Justin_Timberlake ^associatedMusicalArtist Harvey_Mason_Jr. associatedMusicalArtist Whitney_Houston"	0.010416667
8	"Michael_Jackson ^producer Love_Never_Felt_So_Good producer Justin_Timberlake ^associatedBand Harvey_Mason_Jr. associatedMusicalArtist Whitney_Houston"	0.010416667
9	"Michael_Jackson ^associatedBand Jermaine_Jackson ^producer Whitney_Dancin'_Special previousWork Whitney_(album) artist Whitney_Houston"	0.01010101
10	"Michael_Jackson ^associatedBand Jermaine_Jackson ^producer Whitney_Dancin'_Special previousWork Whitney_Houston_(album) artist Whitney_Houston"	0.01010101
11	"Michael_Jackson ^associatedActs Paul_Jackson_Jr. associatedActs Whitney_Houston"	0.01
12	"Michael_Jackson ^associatedActs Paul_Jackson_Jr. associatedBand Whitney_Houston"	0.009803922
13	"Michael_Jackson ^associatedActs Paul_Jackson_Jr. associatedMusicalArtist Whitney_Houston"	0.009803922
14	"Michael_Jackson ^associatedActs Jermaine_Jackson associatedBand Whitney_Houston"	0.009803922
15	"Michael_Jackson ^producer Love_Never_Felt_So_Good musicalArtist Justin_Timberlake ^associatedActs Harvey_Mason_Jr. associatedMusicalArtist Whitney_Houston"	0.009345794
16	"Michael_Jackson ^associatedMusicalArtist Paul_Jackson_Jr. associatedActs Whitney_Houston"	0.009259259
17	"Michael_Jackson ^associatedActs Ricky_Lawson associatedActs Whitney_Houston"	0.009259259
18	"Michael_Jackson ^associatedMusicalArtist Jermaine_Jackson associatedActs Whitney_Houston"	0.009259259
19	"Michael_Jackson ^associatedActs Ricky_Lawson associatedBand Whitney_Houston"	0.009090909
20	"Michael_Jackson ^associatedActs Pattie_Howard associatedActs Whitney_Houston"	0.008928571

Example 7: Top-20 relationship paths between Michael Jackson and Whitney Houston generated by J&E and sorted by the Ground Truth in the Music Domain.

ID	PATH DESCRIPTION	SCORE
1	"Michael_Jackson ^associatedActs Jermaine_Jackson associatedBand Whitney_Houston"	46118820
2	"Michael_Jackson ^associatedMusicalArtist Jermaine_Jackson associatedActs Whitney_Houston"	46118820
3	"Michael_Jackson ^associatedActs Paul_Jackson_Jr. associatedActs Whitney_Houston"	45952366
4	"Michael_Jackson ^associatedActs Paul_Jackson_Jr. associatedMusicalArtist Whitney_Houston"	45952366
5	"Michael_Jackson ^associatedActs Paul_Jackson_Jr. associatedBand Whitney_Houston"	45952366
6	"Michael_Jackson ^associatedMusicalArtist Paul_Jackson_Jr. associatedActs Whitney_Houston"	45952366
7	"Michael_Jackson ^associatedActs Ricky_Lawson associatedActs Whitney_Houston"	45919506
8	"Michael_Jackson ^associatedActs Ricky_Lawson associatedBand Whitney_Houston"	45919506
9	"Michael_Jackson ^associatedActs Pattie_Howard associatedActs Whitney_Houston"	45919011
10	"Michael_Jackson ^producer Love_Never_Felt_So_Good musicalArtist Justin_Timberlake ^associatedActs Harvey_Mason_Jr. associatedMusicalArtist Whitney_Houston"	41134991.2
11	"Michael_Jackson ^producer Love_Never_Felt_So_Good producer Justin_Timberlake ^associatedBand Harvey_Mason_Jr. associatedMusicalArtist Whitney_Houston"	41134991.2
12	"Michael_Jackson ^producer Love_Never_Felt_So_Good producer Justin_Timberlake ^associatedMusicalArtist Harvey_Mason_Jr. associatedMusicalArtist Whitney_Houston"	41134991.2
13	"Michael_Jackson ^producer The_Simpsons_Sing_the_Blues producer Bryan_Loren associatedActs Whitney_Houston"	34440497.25
14	"Michael_Jackson ^influencedBy Jacky_Cheung ^influencedBy Raymond_Lam influences Whitney_Houston"	34439248.25
15	"Michael_Jackson ^producer Thriller_(Michael_Jackson_album) ^album Human_Nature_(Michael_Jackson_song) writer John_Bettis associatedMusicalArtist Whitney_Houston"	30075771
16	"Michael_Jackson ^producer Thriller_(Michael_Jackson_album) ^album Human_Nature_(Michael_Jackson_song) writer John_Bettis associatedBand Whitney_Houston"	30075771

17	"Michael_Jackson ^associatedActs Jermaine_Jackson ^producer Whitney_Dancin'_Special previousWork Whitney_(album) artist Whitney_Houston"	27977903.2
18	"Michael_Jackson ^associatedBand Jermaine_Jackson ^producer Whitney_Dancin'_Special previousWork Whitney_(album) artist Whitney_Houston"	27977903.2
19	"Michael_Jackson ^associatedActs Jermaine_Jackson ^producer Whitney_Dancin'_Special ^subsequentWork Whitney_Houston_(album) artist Whitney_Houston"	27908769.2
20	"Michael_Jackson ^associatedBand Jermaine_Jackson ^producer Whitney_Dancin'_Special previousWork Whitney_Houston_(album) artist Whitney_Houston"	27908769.2

Example 8: Top-20 relationship paths between Elizabeth Taylor and Richard Burton generated by Jaccard Distance & Exclusivity-based Relatedness (J&E) in the Movies Domain.

ID	PATH DESCRIPTION	SCORE
1	"Elizabeth_Taylor spouse Richard_Burton"	0.125
2	"Elizabeth_Taylor ^spouse Richard_Burton"	0.1
3	"Elizabeth_Taylor ^starring Doctor_Faustus producer Richard_Burton"	0.017857142857143
4	"Elizabeth_Taylor ^starring Doctor_Faustus director Richard_Burton"	0.017543859649123
5	"Elizabeth_Taylor ^producer The_Taming_of_the_Shrew starring Richard_Burton"	0.017543859649123
6	"Elizabeth_Taylor ^producer The_Taming_of_the_Shrew screenplay Paul_Dehn ^screenplay The_Spy_Who_Came_in_from_the_Cold starring Richard_Burton"	0.014084507042254
7	"Elizabeth_Taylor ^producer The_Taming_of_the_Shrew screenplay Paul_Dehn ^writer The_Spy_Who_Came_in_from_the_Cold starring Richard_Burton"	0.013157894736842
8	"Elizabeth_Taylor ^producer The_Taming_of_the_Shrew writer Paul_Dehn ^screenplay The_Spy_Who_Came_in_from_the_Cold starring Richard_Burton"	0.012987012987013
9	"Elizabeth_Taylor ^producer The_Taming_of_the_Shrew writer Paul_Dehn ^writer The_Spy_Who_Came_in_from_the_Cold starring Richard_Burton"	0.01219512195122
10	"Elizabeth_Taylor ^narrator Genocide narrator Orson_Welles ^writer Battle_of_Sutjeska starring Richard_Burton"	0.011111111111111

11	"Elizabeth_Taylor ^starring The_V.I.P.s starring Richard_Burton"	0.0097087378640777
12	"Elizabeth_Taylor ^starring Divorce_His_Divorce_Hers starring Richard_Burton"	0.0093457943925234
13	"Elizabeth_Taylor ^starring The_Taming_of_the_Shrew starring Richard_Burton"	0.009009009009009
14	"Elizabeth_Taylor ^starring The_Sandpiper starring Richard_Burton"	0.009009009009009
15	"Elizabeth_Taylor ^starring The_Comedians starring Richard_Burton"	0.009009009009009
16	"Elizabeth_Taylor ^starring Doctor_Faustus starring Richard_Burton"	0.009009009009009
17	"Elizabeth_Taylor ^starring Boom! starring Richard_Burton"	0.009009009009009
18	"Elizabeth_Taylor ^starring Hammersmith_Is_Out starring Richard_Burton"	0.0088495575221239
19	"Elizabeth_Taylor ^producer The_Taming_of_the_Shrew editing Peter_Taylor ^editing Sea_Wife starring Richard_Burton"	0.0088495575221239
20	"Elizabeth_Taylor ^starring Cleopatra starring Richard_Burton"	0.0085470085470085

Example 9: Top-20 relationship paths between Elizabeth Taylor and Richard Burton generated by J&E and sorted by the Ground Truth in the Movie Domain.

ID	PATH DESCRIPTION	SCORE
1	"Elizabeth_Taylor ^producer The_Taming_of_the_Shrew screenplay Paul_Dehn ^screenplay The_Spy_Who_Came_in_from_the_Cold starring Richard_Burton"	6.704322
2	"Elizabeth_Taylor ^producer The_Taming_of_the_Shrew writer Paul_Dehn ^screenplay The_Spy_Who_Came_in_from_the_Cold starring Richard_Burton"	6.704322
3	"Elizabeth_Taylor ^producer The_Taming_of_the_Shrew screenplay Paul_Dehn ^writer The_Spy_Who_Came_in_from_the_Cold starring Richard_Burton"	6.704322
4	"Elizabeth_Taylor ^producer The_Taming_of_the_Shrew writer Paul_Dehn ^writer The_Spy_Who_Came_in_from_the_Cold starring Richard_Burton"	6.704322
5	"Elizabeth_Taylor ^narrator Genocide narrator Orson_Welles ^writer Battle_of_Sutjeska starring Richard_Burton"	6.452606
6	"Elizabeth_Taylor ^producer The_Taming_of_the_Shrew starring Richard_Burton"	6.4424966666667
7	"Elizabeth_Taylor ^starring The_Taming_of_the_Shrew starring Richard_Burton"	6.4424966666667

8	"Elizabeth_Taylor ^starring Cleopatra starring Richard_Burton"	6.37583
9	"Elizabeth_Taylor ^producer The_Taming_of_the_Shrew editing Peter_Taylor ^editing Sea_Wife starring Richard_Burton"	6.216378
10	"Elizabeth_Taylor ^starring The_Comedians starring Richard_Burton"	6.2091633333333
11	"Elizabeth_Taylor ^starring The_V.I.P.s starring Richard_Burton"	6.1424966666667
12	"Elizabeth_Taylor ^starring The_Sandpiper starring Richard_Burton"	6.1091633333333
13	"Elizabeth_Taylor spouse Richard_Burton"	6.063745
14	"Elizabeth_Taylor ^spouse Richard_Burton"	6.063745
15	"Elizabeth_Taylor ^starring Hammersmith_Is_Out starring Richard_Burton"	6.0091633333333
16	"Elizabeth_Taylor ^starring Doctor_Faustus director Richard_Burton"	5.9424966666667
17	"Elizabeth_Taylor ^starring Doctor_Faustus producer Richard_Burton"	5.9424966666667
18	"Elizabeth_Taylor ^starring Doctor_Faustus starring Richard_Burton"	5.9424966666667
19	"Elizabeth_Taylor ^starring Boom! starring Richard_Burton"	5.9091633333333
20	"Elizabeth_Taylor ^starring Divorce_His_Divorce_Hers starring Richard_Burton"	5.9091633333333