Javier Guillot Jiménez

Strategies to Understand 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
September 2021

Javier Guillot Jiménez

Strategies to Understand 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 Examination Committee:

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

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

Profa. Melissa Lemos Cavaliére
Instituto Tecgraf – PUC-Rio

Prof. Luiz André Portes Paes Leme
UFF

Profa. Giseli Rabello Lopes
UFRJ

Rio de Janeiro, September 17th, 2021

Javier Guillot Jiménez

Javier Guillot Jiménez holds a master in computer science degree and a bachelor in computer science degree both from the University of Havana (UH). Javier worked as an assistant professor for eight years at the Faculty of Mathematics and Computer Science of UH. He also worked as a system analyst at the Department of Informatics of the University of Arts (ISA) in Cuba, and the Central Coordination for Planning and Evaluation (CCPA) of the Pontifical Catholic University of Rio de Janeiro (PUC-Rio). His main research topic areas include Semantic Web, Information Retrieval and Linked Data.

Bibliographic data

Guillot Jiménez, Javier

Strategies to Understand the Connectivity of Entity Pairs in Knowledge Bases / Javier Guillot Jiménez; advisor: Marco Antonio Casanova. – 2021.

94 f: il. color. ; 30 cm

Tese (doutorado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2021.

Inclui bibliografia

1. Informática – Teses. 2. Relacionamento de entidades. 3. Medida de similaridade. 4. Ranqueamento de caminhos de relacionamento. 5. Base de Conhecimento. I. Casanova, Marco Antonio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

To my family,
for their support and encouragement.

Acknowledgments

First I would like to thank my advisor Prof. Marco Antonio Casanova for his guidance and his full support to carry out this work. I was very lucky to arrive in a new country to face a great professional challenge and to be welcomed by an excellent professor and researcher and a wonderful person like him.

I would also like to thank Prof. Antonio Luz Furtado, Prof. Luiz André Portes Paes Leme, Yenier Torres Izquierdo, Angelo Batista Neves, and Prof. Giseli Rabello Lopes for their collaboration and for helping me finish this research and publish part of the results.

I would also like to thank all my professors and colleagues from the Faculty of Mathematics and Computer Science of the University of Havana for all the years of learning and experiences as a student and professor, especially dear Prof. Lucina García, Martha Montes de Oca, Carmen Fernández, and Rafael Oliva.

I would like to especially thank my wife Ive who has patiently accompanied me on this long and hard journey from which we leave feeling more in love than ever.

I would also like to thank my family for all the support that they give me from the distance. My parents cannot imagine how much impact a few words in a simple text or audio message have on me. Thank you for the love and for always trusting my decisions and ways of facing life. Thanks to my little sister Haydée for being my driving force and getting me out of my comfort zone, thanks for having shared with me the first years of this adventure. Thank you all for everything.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) - CNPq/146411/2018-8, and by the PUC-Rio. Thank you for the aids granted, without which this work does not could have been accomplished.

Abstract

Guillot Jiménez, Javier; Casanova, Marco Antonio (Advisor). **Strategies to Understand the Connectivity of Entity Pairs in Knowledge Bases**. Rio de Janeiro, 2021. 94p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The entity relatedness problem refers to the question of exploring a knowledge base, represented as an RDF graph, to discover and understand how two entities are connected. This question can be addressed by implementing a path search strategy that combines an entity similarity measure with an entity degree limit and an expansion limit to reduce the path search space and a path ranking measure to order the relevant paths between a given pair of entities in the RDF graph. This thesis first introduces a framework, called CoEPinKB, together with an implementation, to experiment with path search strategies. The framework features as hot spots the entity similarity measure, the entity degree limit, the expansion limit, the path ranking measure, and the knowledge base. The thesis moves on to present a performance evaluation of nine path search strategies using a benchmark from two entertainment domains over the OpenLink Virtuoso SPARQL protocol endpoint of the DBpedia. The thesis then introduces DCoEPinKB, a distributed version of the framework based on Apache Spark, that supports the empirical evaluation of path search strategies, and presents an evaluation of six path search strategies over two entertainment domains over real-data collected from DBpedia. The results provide insights about the performance of the path search strategies and suggest that the framework implementation, instantiated with the best performing pair of measures, can be used, for example, to expand the results of search engines over knowledge bases to include related entities.

Keywords

Entity Relatedness; Similarity Measure; Relationship Path Ranking; Backward Search; Knowledge Base.

Resumo

Guillot Jiménez, Javier; Casanova, Marco Antonio. **Estratégias para entender a conectividade de pares de entidades em bases de conhecimento**. Rio de Janeiro, 2021. 94p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

O problema do relacionamento de entidades refere-se à questão de explorar uma base de conhecimento, representada como um grafo RDF, para descobrir e entender como duas entidades estão conectadas. Esta questão pode ser resolvida implementando-se uma estratégia de busca de caminhos que combina uma medida de similaridade de entidades, um limite para o grau das entidades, e um limite de expansão para reduzir o espaço de busca de caminhos, e uma medida de ranqueamento de caminhos para ordenar os caminhos relevantes entre um determinado par de entidades no grafo RDF. Esta tese inicialmente apresenta um *framework*, chamado CoEPinKB, juntamente com uma implementação, para experimentar estratégias de busca de caminhos. O *framework* apresenta como pontos de flexibilização a medida de similaridade entre entidades, o limite máximo do grau das entidades, o limite de expansão, a medida de classificação de caminhos, e a base de conhecimento. Em seguida, a tese apresenta uma avaliação de desempenho de nove estratégias de busca de caminhos usando um *benchmark* envolvendo dois domínios de entretenimento sobre o *OpenLink Virtuoso SPARQL protocol endpoint* da DBpedia. Por fim, a tese apresenta o DCoEPinKB, uma versão distribuída do *framework* baseado em Apache Spark, que suporta a avaliação empírica de estratégias de busca de caminhos, e apresenta uma avaliação de seis estratégias de busca de caminhos em dois domínios de entretenimento sobre dados reais coletados da DBpedia. Os resultados fornecem intuições sobre o desempenho das estratégias de busca de caminhos e sugerem que a implementação do *framework*, instanciado com o par de medidas de melhor desempenho, pode ser usado, por exemplo, para expandir os resultados dos motores de busca em bases de conhecimento para incluir entidades relacionadas.

Palavras-chave

Relacionamento de entidades; Medida de similaridade; Ranqueamento de caminhos de relacionamento; Base de Conhecimento.

Table of contents

List of figures

List of tables

List of algorithms

List of codes

List of Abreviations

BFS – Breadth-first Search

DCG – Discounted Cumulative Gain

EBR – Exclusivity-based Relatedness

HDFS – Hadoop Distributed File System

IRI – Internationalized Resource Identifier

KB – Knowledge Base

PF-ITF – Predicate Frequency Inverse Triple Frequency

PMI - Pointwise Mutual Information

PT – Property Table

RDF – Resource Description Framework

ST – Statement Table

Turtle – Terse RDF Triple Language

UI – User Interface

VP – Vertical Partitioning

WLM – Wikipedia Link-based Measure

All we have to decide is what to do
with the time that is given us.

J. R. R. Tolkien, *The Fellowship of the Ring.*

1 Introduction

1.1 Context and Motivation

The expansion of data available on the Web grew enormously in recent years. With the rise of the Semantic Web, much of this data is encoded using the RDF data model. Knowledge bases, such as DBpedia (LEHMANN et al., 2015), are expressed using the RDF data model and can be viewed as graphs whose nodes represent entities and whose edges denote relationships. The *entity relatedness problem* refers to the question of exploring a knowledge base, represented as an RDF graph, to discover and understand how two entities are connected. More precisely, this problem can be defined as: "Given an RDF graph G and a pair of entities a and b, represented in G, compute the paths in G from a to b that best describe the connectivity between a and b".

Searching for relevant relationship paths between two entities has applications in several areas. For example, security agencies may be interested in analyzing terrorist networks to discover complex relationships between two suspected terrorists. In the business area, identifying implicit relationships between products and customers helps improve recommendations of new products and generate effective advertisements for potential customers. The academic community may also be interested in finding interrelationships between researchers in co-authorship networks, and a historian may also want to identify the relationships between two politicians in History. Large knowledge bases describe entities and the relations between them and can be used to search for these kinds of relationships. However, entities may share too many direct relations and relationship paths, making it challenging to identify relevant relationship paths between a pair of entities.

Several strategies and tools have been proposed to discover the semantic associations between a pair of entities in a knowledge base. Some approaches first identify all possible relationships between two entities using SPARQL queries to retrieve paths up to a certain length (HEIM et al., 2009; PIRRò, 2015; HERRERA et al., 2016) and then rank the results based on some predefined informativeness measures. Pathfinding techniques have also been used to identify entity relationships (FANG et al., 2011; VOCHT et al., 2013; CHENG; ZHANG; QU, 2014; HERRERA, 2017).

One approach is to apply a two-step strategy: (1) search for relationship paths between a pair of entities; and (2) rank the paths found and select those that are relevant. This strategy must, however, be refined to avoid generating and ranking a very large number of paths. A particular refinement for this approach, which we call *path search strategy*, goes as follows.

The first step adopts backward search (LE et al., 2014), which is a breadth-first search strategy that expands the paths starting from each input entity, in parallel, until a candidate relationship path is generated. The expansion process uses activation criteria to prioritize certain paths over others and to filter the entities less related to the target entities so that it can be easier to identify more meaningful paths. These activation criteria give priority to entities with a low degree in the graph and maintain entities that are similar to the last entity reached in a partially constructed path, using some similarity measure. The second step adopts ranking approaches that use the semantics of the relationships between the entities to assign a score to relationship paths. After sorting the set of relationship paths found in the first step, the top-k paths are selected to describe the connectivity of an entity pair.

Many approaches evaluate the accuracy of relationship path rankings with the help of user experiments (FANG et al., 2011; VOCHT et al., 2013; CHENG; ZHANG; QU, 2014; PIRRò, 2015; HERRERA et al., 2016), while others use a ground truth (HERRERA et al., 2017) to evaluate different strategies that address the entity relatedness problem.

Large public knowledge bases, such as DBpedia, have billions of facts in RDF format that can be queried using semantic query languages, such as SPARQL. However, as the volume of RDF data increases, the computational complexity of indexing and querying large RDF datasets becomes a significant challenge. The distributed nature of the Semantic Web infrastructure itself suggests querying RDF datasets in a parallel and distributed way. Indeed, many distributed SPARQL query engines, built on top of distributed data processing frameworks, have been introduced to overcome this problem (ROHLOFF; SCHANTZ, 2010; HUSAIN et al., 2011; HUANG; ABADI; REN, 2011; PRZYJACIEL-ZABLOCKI et al., 2012; VIRGILIO; MACCIONI, 2014; SCHäTZLE et al., 2016; ABDELAZIZ et al., 2017; SUN et al., 2019; RAGAB et al., 2021). Hence, the entity relatedness problem may also benefit from parallel and distributed approaches.

1.2 Goal and Contributions

The main goal of this thesis is to investigate flexible and extensible approaches to the entity relatedness problem that scale to large knowledge bases.

The first contribution of this thesis is the proposal and implementation of an approach and a framework, called CoEPinKB, that helps address the entity relatedness problem. CoEPinKB differs from the solution proposed in Herrera (2017) in three main aspects. First, CoEPinKB was designed to make it easy for developers to add new entity similarity and relationship path ranking measures to generate new path search strategies. Second, CoEPinKB has a simple and practical Web user interface that facilitates the interaction of the users with the framework and provides an API that facilitates executing different experiments and analyzing the results. Lastly, CoEPinKB was engineered to work with any knowledge base accessible using a remote SPARQL query endpoint over HTTP.

The analysis in Herrera (2017) evaluated nine relationship path search strategies on two entertainment domains. However, the analysis did not evaluate the performance of these strategies concerning execution time. The second contribution of this thesis is the use of CoEPinKB to evaluate the performance of these different strategies concerning execution time on two entertainment domains in DBpedia.

The third contribution of this thesis is the conception of a novel approach to address the entity relatedness problem in a distributed manner. The thesis describes a proof-of-concept implementation of the approach as a framework, called DCoEPinKB. To the best of our knowledge, this is the first work addressing the entity relatedness problem using a distributed strategy.

The fourth contribution of this thesis is the construction of test datasets from the music and movies domains, collected from two subsets of the English DBpedia corpus, that support the evaluation of the accuracy of path search strategies.

Finally, the fifth contribution is an extensive experimental evaluation of the correctness and performance of DCoEPinKB, comparing it with CoEPinKB, using the previously constructed test datasets.

The contributions of this thesis were partly published in Jiménez, Leme & Casanova (2021) and presented at the Brazilian Symposium on Databases (SBBD 2021) (JIMéNEZ et al., 2021).

1.3 Structure of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 summarizes background information about RDF, the representation of RDF in relational databases, entity similarity measures, relationship path ranking measures, and ranking accuracy. Chapter 3 reviews related work. Chapter 4 describes the proposed solutions and the process of finding relevant relationship paths between entity pairs through a backward search algorithm. Chapter 5 presents the architecture and the implementation of the CoEPinKB and DCoEPinKB frameworks and shows how they can be used. Chapter 6 presents a performance evaluation of a family of path search strategies using the proposed frameworks. Finally, Chapter 7 presents the conclusions and future work of the thesis.

2
Background

This chapter provides the required background information to understand the basic principles of RDF and the use of similarity and path ranking measures in knowledge graphs to find relevant relationship paths between an entity pair.

2.1
RDF

The *Resource Description Framework* (RDF) is a flexible and extensible data model for representing information about resources (SCHREIBER; RAIMOND, 2014). Resources can be anything, including documents, people, physical objects, and abstract concepts. RDF allows representing this variety of resources and their relationships through RDF triples, which are statements about resources that have the form (s, p, o), where s stands for the subject, p for the predicate, and o for the object.

The subject of an RDF triple is an IRI or a blank node, the predicate is an IRI, and the object is an IRI, a literal or a blank node. The abbreviation IRI is short for *Internationalized Resource Identifier*. An IRI identifies a resource of the real world and is a generalization of URI (Uniform Resource Identifier) that permits a wider range of Unicode characters. An example of IRI is `dbr:Paul_Newman`[1]. A literal is a basic value of a specific data type, that defines the range of possible values, such as strings, numbers, and dates. Unlike IRIs and literals, blank nodes do not identify specific resources and can always be replaced by Skolem IRIs (CYGANIAK; WOOD; LANTHALER, 2014).

The triple `(dbr:Paul_Newman, dbo:spouse, dbr:Joanne_Woodward)` is an example that states that Paul Newman was married to Joanne Woodward. The RDF triple can also be denoted without parentheses and commas and ending with a dot as:

```
dbr:Paul_Newman dbo:spouse dbr:Joanne_Woodward .
```

The term *resource* is synonymous with *entity* as it is used in the RDF Semantics specification (HAYES; PATEL-SCHNEIDER, 2014), such that, in this thesis, both terms are used interchangeably. The predicate represents the nature of the relationship between the subject and the object and is called in

[1]https://dbpedia.org/resource/Paul_Newman

RDF a *property*. While *object properties* relate resources to resources, *datatype properties* assign literals to resources.

An RDF dataset R is a set of RDF triples. It can be modeled as a labeled graph $G_R = (V_R, E_R, L_R)$, where V_R is the set of resources and literals that occur as subject or object of the triples in R and there is an edge (s, o) in E_R labeled with $p = L_R(s, o)$ iff the triple (s, p, o) occurs in R. An IRI in V_R will be referred to as an entity occurring in G.

An *undirected path* π in an RDF graph G_R between entities w_0 and w_k is an expression of the form $(w_0, p_1, w_1, p_2, w_2, \ldots, p_{k-1}, w_{k-1}, p_k, w_k)$, where: k is the length of the path; w_i is an entity in G_R such that w_i and w_j are different, for $0 \leq i \neq j \leq k$; and either (w_i, w_{i+1}) or (w_{i+1}, w_i) are edges of G labeled with p_{i+1}, for $0 \leq i < k$. Note that, since a path is undirected, but G is a directed graph, we allow either (w_i, w_{i+1}) or (w_{i+1}, w_i) to be used to generate a path. For example, the following expression represents a path that connects Paul Newman and Joanne Woodward, where $k = 2$: `(dbr:Paul_Newman, dbo:starring, dbr:Our_Town_(2003_film), dbo:executiveProducer, dbr:Joanne_Woodward)`. This path exists because the following triples occurs in R (we recall that "`^p`" denotes the inverse of a property `p` in SPARQL notation):

```
dbr:Paul_Newman ^dbo:starring dbr:Our_Town_(2003_film) .
dbr:Our_Town_(2003_film) dbo:executiveProducer dbr:Joanne_Woodward .
```

A large number of RDF datasets are available and interlinked on the Web as Linked Data (BERNERS-LEE, 2006), and many of them offer a querying facility through SPARQL (PRUD'HOMMEAUX; SEABORNE, 2008), a query language for RDF. The results of SPARQL queries can be result sets or RDF graphs. An example of a SPARQL query that returns direct RDF paths of length 4 between `dbr:Paul_Newman` and `dbr:Joanne_Woodward` is shown below:

Code 1: SPARQL query example

```
SELECT *
WHERE { dbr:Paul_Newman ?p1 ?e1 .
        ?e1 ?p2 ?e2 .
        ?e2 ?p3 ?e3 .
        ?e3 ?p4 dbr:Joanne_Woodward . }
```

The `WHERE` clause contains triple patterns that are matched with an RDF graph. A triple pattern is similar to an RDF triple, except that the subject, predicate, or object can be a query variable (denoted in SPARQL prefixed with "?", such as `?e1` or `?p3`).

As the volume of RDF data grows, the computational complexity of indexing and querying large datasets becomes challenging. Distributed data processing frameworks like Apache Spark (ZAHARIA et al., 2010) are not designed to perform native RDF processing. However, relational schemas can be used to represent RDF data in an efficient and scalable way.

The *Statement Table* (ST), also known as *triple table* (FAYE; CURé; BLIN, 2012; ABDELAZIZ et al., 2017), can be considered the most straightforward way of storing RDF triples. It directly maps RDF data onto a table with three columns (subject, predicate, object), in which each tuple corresponds to an RDF statement. This solution is the simplest, and it has been adopted by several existing RDF triplestores (MAHRIA; CHAKER; ZAHI, 2021). However, depending on the kind of queries executed on the dataset, the storage of RDF triples in a single table makes the queries very slow to execute due to expensive self-joins. Table 2.1 shows an example of the ST representation schema of a small RDF graph.

Table 2.1: Example of a Statement Table representation

subject	**predicate**	**object**
`dbr:Paul_Newman`	`dbo:type`	`dbo:Person`
`dbr:Paul_Newman`	`dbo:spouse`	`dbr:Joanne_Woodward`
`dbr:Paul_Newman`	`dbo:child`	`dbr:Melissa_Newman`
`dbr:Paul_Newman`	`dbo:child`	`dbr:Nell_Newman`
`dbr:Joanne_Woodward`	`dbo:type`	`dbo:Person`
`dbr:Joanne_Woodward`	`dbo:child`	`dbr:Melissa_Newman`
`dbr:Our_Town_(2003_film)`	`dbo:type`	`dbo:Film`
`dbr:Our_Town_(2003_film)`	`dbo:starring`	`dbr:Paul_Newman`
`dbr:Our_Town_(2003_film)`	`dbo:executiveProducer`	`dbr:Joanne_Woodward`
...	...	...

The *Property Table* (PT) was a proposal to try to overcome some of the scalability limits that persist in the ST approach and it can be classified into two types: *clustered property table* and *property-class table* (ABDELAZIZ et al., 2017). The basic idea behind the property table is to discover clusters of subjects that share the same properties and to group them into a table. For each tuple of the property table, one column contains the subject of the triples and one or more columns contain the property values for that subject.

The *clustered property table* consists of a unique table whose dimensions are determined by the number of distinct subjects and predicates. Table 2.2 shows the clustered property table representation schema of the same example in Table 2.1.

On the other hand, the *property-class table* approach uses different tables, one for each "type" of subject, and exploits the `rdf:type` predicate to cluster

Table 2.2: Example of a Clustered Property Table representation

subject	rdf:type	rdf:spouse	rdf:child	rdf:starring	rdf:executiveProducer
dbr:Paul_Newman	dbo:Person	dbr:Joanne_Woodward	dbr:Melissa_Newman	NULL	NULL
dbr:Joanne_Woodward	dbo:Person	dbr:Paul_Newman	dbr:Melissa_Newman	NULL	NULL
dbr:Our_Town_(2003_film)	dbo:Film	NULL	NULL	dbr:Paul_Newman	dbr:Joanne_Woodward
...	...	...	...	...	...

similar sets of subjects in the same table. Table 2.3 shows the property-class table representation schema of the same example in Table 2.1.

Table 2.3: Example of a Property-Class Table representation

Person		
subject	**rdf:spouse**	**rdf:child**
dbr:Paul_Newman	dbr:Joanne_Woodward	dbr:Melissa_Newman
dbr:Joanne_Woodward	dbr:Paul_Newman	dbr:Melissa_Newman
...	...	...

Film		
subject	**rdf:starring**	**rdf:executiveProducer**
dbr:Our_Town_(2003_film)	dbr:Paul_Newman	dbr:Joanne_Woodward
...	...	...

The main benefit of using PT is that complex queries can be executed avoiding expensive self-joins. However, it is still expensive, because some complex queries need to combine data from several tables. Additionally, this approach generates many `NULL` values since, for a given cluster, not all properties will be defined for all subjects (FAYE; CURé; BLIN, 2012). Due to its sparse tables representation, it has a high storage overhead, when a large number of predicates is present in the knowledge base (ABDELAZIZ et al., 2017). This RDF relational schema is also less flexible than the ST approach as the clustered properties might need to be rearranged as data changes and it can not represent multi-valued properties.

Vertical Partitioning (VP) is an alternative relational schema for RDF data proposed by Abadi et al. (2009) in which the RDF triples table is decomposed into a table of two columns (subject, object) for each unique property in the knowledge base. The first column stores the subjects of the triples that have the property defined for the table, and the second column contains the object values for those subjects. Table 2.4 shows the vertical partitioning representation schema of the same example in Table 2.1.

This approach admits multi-valued attributes and does not store `NULL` values. However, Sidirourgos et al. (2008) pointed out potential scalability problems for the VP approach, when the dataset contains a large number of properties. Indeed, it leads to a large number of tables, which is problematic

Table 2.4: Example of a Vertical Partitioning representation

rdf:type	
subject	**object**
dbr:Paul_Newman	dbo:Person
dbr:Joanne_Woodward	dbo:Person
dbr:Our_Town_(2003_film)	dbo:Film
...	...

rdf:child	
subject	**object**
dbr:Paul_Newman	dbr:Melissa_Newman
dbr:Paul_Newman	dbr:Nell_Newman
dbr:Joanne_Woodward	dbr:Melissa_Newman
...	...

rdf:spouse	
subject	**object**
dbr:Paul_Newman	dbr:Joanne_Woodward
dbr:Joanne_Woodward	dbr:Paul_Newman
...	...

rdf:starring	
subject	**object**
dbr:Our_Town_(2003_film)	dbr:Paul_Newman
...	...

rdf:executiveProducer	
subject	**object**
dbr:Our_Town_(2003_film)	Joanne_Woodward
...	...

when executing queries that require scanning multiple tables to reconstruct information related to a single entity.

In this thesis, the ST schema was used. For our problem, this schema is the most suitable among the three presented above due to the type of queries that our framework performs mostly and the particularities of the graphs used, in which, for example, there are a large number of properties.

2.2 Similarity Measures

2.2.1 Overview

A similarity measure is a real-valued function σ that quantifies the similarity between two objects e and f, such that $\sigma(e, f) \in [0, 1]$, $\sigma(e, e) = 1$, and $\sigma(e, f) = \sigma(f, e)$. Similarity measures can be classified into four main categories (MEYMANDPOUR; DAVIS, 2016): (i) distance-based models, which are based on the structural representation of the underlying context; (ii) feature-based models, which define concepts or entities as sets of features; (iii) statistical methods, which consider statistics derived from the underlying context; and (iv) hybrid models, which comprise combinations of the three basic categories.

The remainder of this section describes the similarity measures that will be used in the proposed solution to address the entity relatedness problem, i.e., the Jaccard index (JACCARD, 1901), the Wikipedia Link-based Measure (MILNE; WITTEN, 2008) and SimRank (JEH; WIDOM, 2002).

Feature-based similarity measures, such as the Jaccard index, assume that concepts can be represented as sets of features and assess the similarity of concepts based on the commonalities among their feature sets. In this thesis, the set of features of an entity is modeled as a set of entities in its surroundings.

Let T be an RDF dataset and G be the RDF graph induced by T. For two entities a and b in G, two abstract walkers are deployed to traverse the graph at a specific depth d to acquire features. At each depth, a walker collects entities, after visiting depth d, the walkers return the sets of features A_d and B_d of entities a and b, respectively. Note that an entity $z \in A_d$ iff there is a path from a to z of length less than or equal to d.

2.2.2 Jaccard Index

The Jaccard index (JACCARD, 1901), also known as the *Jaccard similarity coefficient*, is a classical similarity measure with several practical applications in information retrieval, data mining, machine learning, and many more areas (LESKOVEC; RAJARAMAN; ULLMAN, 2014). The Jaccard index between two entities a and b in G is defined as the cardinality of the intersection of their sets of features A_d and B_d divided by the cardinality of their union:

$$J(a,b) = \frac{|A_d \cap B_d|}{|A_d \cup B_d|} = \frac{|A_d \cap B_d|}{|A_d| + |B_d| - |A_d \cap B_d|} \tag{2-1}$$

If $a = b$ or $A_d \cup B_d = \emptyset$, we define $J(a,b) = 1$.

2.2.3 Wikipedia Link-based Measure

The Wikipedia Link-based Measure (WLM), proposed by Milne & Witten (2008), was initially defined to compute the semantic similarity of two Wikipedia pages by comparing their incoming and outgoing links. The main difference between this similarity measure and other contemporary Wikipedia-based approaches is the use of Wikipedia's hyperlink structure to define relatedness rather than its category hierarchy or textual content. Wikipedia pages that link common pages indicate relatedness, while pages that link disparate pages suggest the opposite. This measure can be also used to measure the similarity between two entities a and b in G and is defined as:

$$WLM(a,b) = 1 - \frac{\log(\max(|A_d|, |B_d|)) - \log(|A_d \cap B_d|)}{\log(|V|) - \log(\min(|A_d|, |B_d|))} \tag{2-2}$$

where V is the set of entities of G.

2.2.4 SimRank

SimRank (JEH; WIDOM, 2002) is a general link-based similarity measure that is based on a simple and intuitive graph-theoretic model and is applicable in any domain with object-to-object relationships. It measures the similarity of the structural context in which objects occur based on their relationships with other objects.

The intuition behind SimRank is that, in many domains, two objects are considered similar if they are referenced by similar objects. Entities a and b are similar if they are related to entities x and y, respectively, and x and y are themselves similar. The base case considers that entities are similar to themselves. This similarity measure can be inferred by recursively considering the similarity of the neighbors of two objects.

Let G be an RDF graph and w be an entity (node) in G. A node v is an *in-neighbor* of w iff there is an edge (v, w) in G. Let $I(w)$ denote the set of *in-neighbors* of w in G. The SimRank score $s(a, b)$ between entities a and b is defined as follows:

$$s(a,b) = \frac{\lambda}{|I(a)||I(b)|} \sum_{c \in I(a)} \sum_{d \in I(b)} s(c,d) \tag{2-3}$$

where λ is a confidence level between 0 and 1. If $a = b$, we define $s(a, b) = 1$. If $I(a) \cup I(b) = \emptyset$, we define $s(a, b) = 0$.

The two main limitations of many algorithms that compute SimRank are that the computing cost can be very high in practice and that they can only be applied to static graphs. Due to the computational complexity of SimRank, there are many studies to speed up such calculations (LIZORKIN; VELIKHOV, 2008; LI et al., 2010; LI et al., 2020; HAMEDANI; KIM, 2021).

2.3 Relationship Path Ranking Measures

2.3.1 Overview

After finding relationship paths between two entities, one important step is to rank the paths and consider only the top-k most relevant ones. A relationship path ranking measure r considers each path π between two entities a and b in an RDF graph G as a sequence of properties and nodes, as defined in Section 2.1, analyses each component in π, and generates a score. A higher score indicates a greater relevance.

One simple measure ranks the paths according to the distance between the two entities and prioritizes the shortest paths. However, there are more complex and accurate relationship path ranking measures that consider the semantic of the relationships that link the entities.

The remainder of this section describes the relationship path ranking measures that will be used in this thesis to address the entity relatedness problem, i.e., the PREDICATE FREQUENCY INVERSE TRIPLE FREQUENCY (PIRRò, 2015), the EXCLUSIVITY-BASED RELATEDNESS (HULPUş; PRANGNAWARAT; HAYES, 2015) and the POINTWISE MUTUAL INFORMATION (CHURCH; HANKS, 1990) to evaluate different factors of the constituent RDF predicates of each relationship path. For the next subsections, let T be an RDF dataset, G be the RDF graph induced by T, p be a predicate in T (a property in G) and w be an entity in G.

2.3.2 Predicate Frequency Inverse Triple Frequency

The PREDICATE FREQUENCY INVERSE TRIPLE FREQUENCY (PF-ITF), proposed by Pirrò (2015), is an adaptation of the original TF-IDF used in information retrieval that considers the participation of a predicate p in all triples in an RDF dataset and can be defined as follows. The frequency of p incoming to (outgoing from) w in G, $pf_i^w(p,G)$ and $pf_o^w(p,G)$, are shown in Equation 2-4 and Equation 2-5, respectively. The *inverse triple frequency* of p in G, $itf(p,G)$, and the *predicate frequency inverse triple frequency*, $pfitf_x^w(p,G)$, are shown in Equation 2-6 and Equation 2-7, respectively.

$$pf_i^w(p,G) = \frac{|* \xrightarrow{p} w|}{|* \rightarrow w|} \tag{2-4}$$

$$pf_o^w(p,G) = \frac{|w \xrightarrow{p} *|}{|w \rightarrow *|} \tag{2-5}$$

$$itf(p,G) = \log \frac{|T|}{|* \xrightarrow{p} *|} \tag{2-6}$$

$$pfitf_x^w(p,G) = pf_x^w(p,G) \times itf(p,G) \tag{2-7}$$

where $|* \xrightarrow{p} w|$ is the number of triples in G where the predicate p is incoming to w, $|w \xrightarrow{p} *|$ is the number of triples where the predicate p is outgoing from w, $|* \rightarrow w|$ is the total number of triples incoming to w, $|w \rightarrow *|$ is the total number of triples outgoing from w, and $|* \xrightarrow{p} *|$ is the total number of triples including p. Note that, in Equation 2-7, $pfitf_x^w(p,G)$ can use $pf_i^w(p,G)$ or $pf_o^w(p,G)$.

Let $\pi(w_0, w_1) = (w_0, p_1, w_1)$ be a path between w_0 and w_1 in G of length $k = 1$. The score of π is defined as:

$$score(\pi(w_0, w_1), G) = \frac{pfitf_o^{w_0}(p_1, G) + pfitf_i^{w_1}(p_1, G)}{2} \tag{2-8}$$

The score of a path $\pi(w_0, w_k) = (w_0, p_1, w_1, \ldots, w_{k-1}, p_k, w_k)$, for $k > 1$, is defined as:

$$score(\pi(w_0, w_k), G) = \frac{score(\pi(w_0, w_1), G) + \ldots + score(\pi(w_{k-1}, w_k), G)}{k} \tag{2-9}$$

2.3.3 Exclusivity-based Relatedness

The Exclusivity-based Relatedness (EBR), introduced by Hulpuş, Prangnawarat & Hayes (2015), claims that a relation between two concepts is stronger if each of the concepts is related through the same type of relationship to fewer other concepts. This property of relations is called by the authors extitexclusivity. Using the notation introduced to define PF-ITF, the *exclusivity* of a relationship $a \xrightarrow{p} b$ is defined as:

$$exclusivity(a \xrightarrow{p} b) = \frac{1}{|a \xrightarrow{p} *| + |* \xrightarrow{p} b| - 1} \tag{2-10}$$

The denominator is subtracted by 1 because the relationship $a \xrightarrow{p} b$ is otherwise counted twice, once for the relationships outgoing from a and once for the relationships incoming to b. The score of a path $\pi(w_0, w_k) = (w_0, p_1, w_1, \ldots, w_{k-1}, p_k, w_k)$ in G is defined as:

$$score(\pi(w_0, w_k), G) = \frac{1}{\sum_{i=1}^{k} 1/exclusivity(w_{i-1} \xrightarrow{p_i} w_i)} \tag{2-11}$$

2.3.4 Pointwise Mutual Information

The Pointwise Mutual Information (PMI), proposed by Church & Hanks (1990), measures the co-occurrence strength between two items. It relates the probabilities of the individual occurrence of the items to the probability of both items occurring together. The PMI score of a path is estimated based on the co-occurrence of the properties and entities in the path. We consider three cases in the computation of the relevance of a path:

1. Co-occurrence of two properties p_r and p_s, when they are properties of the same entity. (Equation 2-12)

2. Co-occurrence of a property p and an entity w, when p is outgoing from w. (Equation 2-13)

3. Co-occurrence of a property p and an entity w, when p is incoming to w. (Equation 2-14)

These cases can be formalized as follows:

$$PMI(p_r, p_s) = \log \frac{f(p_r, p_s)}{f(p_r) * f(p_s)} \tag{2-12}$$

$$PMI(w, p) = \log \frac{f(w, p)}{f(w) * f(p)} \tag{2-13}$$

$$PMI(p, w) = \log \frac{f(p, w)}{f(p) * f(w)} \tag{2-14}$$

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 score of a path $\pi(w_0, w_k) = (w_0, p_1, w_1, \ldots, w_{k-1}, p_k, w_k)$ in G is defined as:

$$\begin{aligned} score(\pi(w_0, w_k), G) = {} & median\{PMI(p_i, p_j) | 1 \leq i \neq j \leq k\} \\ & + (1/2k) * (PMI(w_0, p_1) + \ldots + PMI(w_{k-1}, p_k) \\ & + PMI(p_1, w_1) + \ldots + PMI(p_k, w_k)) \end{aligned} \tag{2-15}$$

2.4 Measuring Ranking Accuracy

The list of the top-k most relevant relationship paths between two entities, obtained after using some relationship path ranking measure, can be compared to a *golden standard*. This section recalls the definition of normalized discounted cumulative gain, which will be used in Chapter 6 to measure the accuracy of the different path search strategies.

The Discounted Cumulative Gain (DCG) is a well-known measure proposed by Järvelin & Kekäläinen (2002) and used in information retrieval to assess ranking accuracy. 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 of n documents with ratings $rel_1, \ldots, rel_n$. The *discounted cumulative gain* of the top-k results, with $1 \leq k \leq n$, denoted DCG_k, is defined as:

$$DCG_k = rel_1 + \sum_{i=2}^{k} \frac{rel_i}{\log_2(i+1)} \tag{2-16}$$

DCG_k is normalized by $IDCG_k$, the discounted cumulative gain for an ideal ranking of the top-k results. Then, the *normalized discounted cumulative gain* is defined as:

$$nDCG_k = \frac{DCG_k}{IDCG_k} \tag{2-17}$$

Note that in a perfect ranking algorithm, the DCG_k will be the same as the $IDCG_k$, producing an $nDCG_k$ equal to 1.

3
Related Work

In this chapter, we review research that introduced approaches for discovering and ranking entity relationship paths in knowledge bases, ways to measure the effectiveness of path search strategies, as well as distributed query processing systems that allow similarity-based operations, and distributed engines for processing large RDF datasets.

3.1
Entity Relationship Discovery and Ranking in Knowledge Bases

Several strategies and tools have been proposed to discover the semantic associations between a pair of entities in a knowledge base. Some approaches first identify all possible relationships between two entities, using SPARQL queries to retrieve paths up to a certain length (HEIM et al., 2009; PIRRò, 2015; HERRERA et al., 2016), and then rank the results based on some predefined informativeness measures. Pathfinding techniques have also been used to identify entity relationships (FANG et al., 2011; MOORE; STEINKE; TRESP, 2012; VOCHT et al., 2013; CHENG; ZHANG; QU, 2014; HERRERA, 2017).

Heim et al. (2009) proposed an approach that automatically reveals relationships between two known entities and displays them as a graph. The relationship paths are found by an algorithm based on the concept of *decomposition* of an RDF graph (LEHMANN; SCHüPPEL; AUER, 2007) and composed of several SPARQL queries that search iteratively for paths with increasing length, starting from zero, between the input entities. The authors presented RelFinder, an implementation of this approach, and demonstrated its applicability using an example from the DBpedia. However, this approach does not provide mechanisms for ranking or comparing paths.

REX (FANG et al., 2011) is a system implemented in Python that takes a pair of entities in a given knowledge base as input and produces a ranked list of *relationship explanations*. The authors consider a relationship explanation as a constrained graph pattern and its associated graph instances derivable from the underlying knowledge base. REX implements different algorithms for finding the relationship explanations, adapted from solutions proposed for the keyword search problem in databases. The PathEnumBasic algorithm is based on the backward expansion search introduced in BANKS (BHALOTIA et al., 2002) and generates partial paths from input entities concurrently, with shorter paths being generated first. The second path enumeration algorithm PathEnumPrioritized is a direct adaption of the bidirectional search (KACHOLIA et al., 2005), an improved version of BANKS, and instead of always expanding the shortest partial paths, the degree of the nodes is used as an activation score to prioritize the expansion. The authors also proposed some *interestingness* measures for ranking relationship explanations and performed user experiments to demonstrate the effectiveness of the algorithms.

Moore, Steinke & Tresp (2012) proposed an approach that can find informative paths between two specified nodes. It performs a shortest paths search between the two nodes using a metric that just depends on the degrees of adjacent nodes and favors paths via low-degree nodes, thus ensuring that the paths prefer more specific and informative relationships over general ones.

Vocht et al. (2013) introduced an approach for pathfinding that takes into account the meaning of the connections and uses a distance metric based on Jaccard. It applies the measure to estimate the similarity between two nodes and to assign a weight based on the random walk, which ranks the rarest resources higher. Vocht et al. (2016) proposed an in-depth extension of this algorithm which reduces arbitrariness by increasing the relevance of links between nodes through additional pre-selection and refinement steps. The authors also compared and measured the effectiveness of different search strategies through user experiments.

Explass is an approach proposed by Cheng, Zhang & Qu (2014) that explores a knowledge base searching for associations and provides a list of the top-k clusters, which are labeled with an association pattern that gives users a conceptual summary of the associations in the cluster. The clusters are obtained by formulating and solving a data mining problem, and then the top-k ones are found by formulating and solving an optimization problem. Explass integrates patterns with facet values, which are classes of entities and relationships that appear in associations, and that can be used by users to refine the search and better explore associations. The authors compared Explass with two existing related approaches by conducting a user study and tested the statistical significance of the results. Cheng, Shao & Qu (2017) examined existing techniques for ranking semantic associations and proposed two new techniques based on the heterogeneity or homogeneity of the constituents of a semantic association. Cheng, Liu & Qu (2021) presented a fast algorithm for semantic associations search by enumerating and joining paths, which proved a tighter bound and allowed more effective distance-based pruning of the search space than previous work.

Pirrò (2015) introduced RECAP, a framework to generate different types of relatedness explanations between entities, possibly combining information from multiple knowledge bases. RECAP goes beyond related approaches such as REX and EXPLASS, as it allows to build different types of explanations (for example, graphs and sets of paths), thus controlling the amount of information displayed. The author first formalizes the notion of *relatedness explanation* and introduces different criteria to build explanations based on information theory, diversity, and their combinations. The first approach that the author proposed for ranking paths between a pair of entities is based on the informativeness of a path and uses the novel PF-ITF measure to calculate the score of a path. The author conducted experiments to investigate whether RECAP provides useful explanations to the user.

DBPEDIA PROFILER is a tool proposed by Herrera et al. (2016) which implements a strategy to generate connectivity profiles for entities represented in DBpedia. The tool uses SPARQL queries to identify relationship paths that connect the given pair of entities and adopts a strategy based on semantic annotations, which use a similarity measure, to group and summarize the collected paths. The authors made experiments to compare DBPEDIA PROFILER with RECAP and the results showed that DBPEDIA PROFILER outperforms RECAP in terms of performance and usability.

As stated above, many approaches (FANG et al., 2011; VOCHT et al., 2013; CHENG; ZHANG; QU, 2014; PIRRò, 2015; HERRERA et al., 2016) evaluate relationship paths rankings with the help of user experiments. Herrera et al. (2017), by contrast, proposed the ENTITY RELATEDNESS TEST DATASET, a ground truth of paths between pairs of entities in two entertainment domains in the DBpedia that supports the evaluation of different strategies that address the entity relatedness problem. The authors used information from the Internet Movie Database (IMDb)[1] and last.fm[2] to generate specialized relationship path rankings between entity pairs in the movies and music domains, respectively. For each domain, the dataset contains 20 pairs of entities, each with a ranked list with 50 relationship paths based on information about their entities found in IMDb and last.fm, and on information about their properties, computed from DBpedia. However, the authors do not specify which version of the DBpedia they relied on to generate the ground truth, making it difficult to use their test dataset in our experiments considering that the content of the DBpedia has changed since then. Section 6.2.1.8 of this thesis circumvents this problem by introducing a strategy to construct ground truths for any given version of the DBpedia.

[1]https://www.imdb.com/
[2]https://www.last.fm/

Herrera (2017) introduced a generic search strategy, based on the backward search heuristic proposed by Le et al. (2014) for keyword search, which combines SPARQL queries, activation criteria, similarity, and ranking measures to find relevant paths between a pair of entities in alternative ways. This approach expands the paths starting from two source entities and prioritizes certain partial paths over others until relationship paths between these entities are generated. The activation criteria consider the degree of the entities and use similarity measures, such as JACCARD INDEX, WLM, and SIMRANK. For ranking the paths found and selecting those that are relevant, the author used ranking measures, such as PF-ITF, EBR, and PMI. Finally, the author evaluated the accuracy of the results of the different strategies with the help of the ground truth proposed by Herrera et al. (2017). However, this work lacks an evaluation of the performance, in terms of execution time, of each of the different path search strategies, as well as a tool with a graphical user interface that facilitates evaluating these strategies. The author also identified some opportunities for future work, such as to develop a framework for the entity relatedness problem, considering as points of flexibility the similarity measure between entities and the path-ranking measure to identify relevant paths. This thesis aims at filling this gap, as described in the following chapters.

3.2 Similarity-based operations on Distributed Query Processing Systems

DIMA (SUN et al., 2017) is a distributed in-memory similarity-based query processing system built on top of Spark (ZAHARIA et al., 2010). DIMA extends the Spark SQL programming interface and the Catalyst optimizer for users to easily invoke similarity selection and similarity join query operations in their data processing tasks. Similarity selection extends traditional exact selection by tolerating errors and similarity join extends traditional exact join by tolerating errors between records. DIMA supports various data sources, such as CSV, JSON, and Parquet, and implements two types of similarity: set-based similarity, using the Jaccard index, and character-based similarity, using edit distance. The authors proposed an approach for similarity-based queries that employs offline distributed indexing.

Sun et al. (2019) improved DIMA to support additional similarity operations, such as top-k selection and top-k join. Top-k selection computes the k most similar records and top-k join computes the k most similar pairs. To avoid expensive data transmission, the authors proposed DIMA+, an approach that uses a balance-aware signature selection to balance the workload in distributed environments.

Unlike DIMA and DIMA+, Kim et al. (2020) focused on handling very large datasets that do not fit in memory. The authors extended Apache AsterixDB, an open-source parallel data management system for semi-structured data, to allow users to specify a similarity query, either by using a system-provided function or specifying their logic as a user-defined function. They presented an experimental study based on several large datasets on a parallel computing cluster to evaluate the proposed techniques for supporting similarity queries and presented a performance comparison with three other parallel systems.

3.3 Processing Large RDF Datasets in Distributed Environments

The continuous growth of knowledge bases has led to the search for new approaches and technologies to store, access, and querying RDF data. Many distributed RDF systems have been introduced to overcome the problem of indexing and querying large RDF datasets (HUSAIN et al., 2009; ROHLOFF; SCHANTZ, 2010; HUSAIN et al., 2011; HUANG; ABADI; REN, 2011; PRZYJACIEL-ZABLOCKI et al., 2012; VIRGILIO; MACCIONI, 2014; SCHäTZLE et al., 2016; SCHäTZLE et al., 2016; ABDELAZIZ et al., 2017; RAGAB; TOMMASINI; SAKR, 2019; RAGAB et al., 2020; RAGAB et al., 2021). These systems are generally built on top of distributed data processing frameworks, such as MapReduce (DEAN; GHEMAWAT, 2008) or Apache Spark (ZAHARIA et al., 2010), partition the RDF graphs among multiple machines to handle big datasets, and parallelize query execution to reduce query runtime.

SHARD is an open-source, horizontally scalable triplestore system proposed by Rohloff & Schantz (2010) and built using the Hadoop implementation of MapReduce. SHARD persists graph data as RDF triples and responds to queries over this data in the SPARQL query language. Husain et al. (2011) presented HADOOPRDF, an scalable and fault-tolerant framework based on Hadoop that supports data-intensive query processing. The authors proposed a storage schema to store RDF data in HDFS[3] and a new greedy algorithm that overcomes the limitations of the algorithm previously introduced by Husain et al. (2009). The algorithm uses the MapReduce programming model and produces a query plan whose cost, i.e., the number of Hadoop jobs that will be executed to solve the query, is bounded by the logarithm of the total number of variables in the given SPARQL query. Huang, Abadi & Ren (2011) presented a scale-out architecture for RDF data management using the

[3]Hadoop Distributed File System

Hadoop framework. The authors described data partitioning and placement techniques that reduce the amount of network communication at query time and provided an algorithm for automatically decomposing SPARQL queries into parallelizable Hadoop jobs. RDFPath is a declarative path query language for RDF proposed by Przyjaciel-Zablocki et al. (2012) that automatically transforms declarative path queries into MapReduce jobs and supports the exploration of graph properties such as shortest paths between two nodes in an RDF graph. Virgilio & Maccioni (2014) presented a distributed approach to keyword search query over large RDF datasets that exploits the MapReduce paradigm by switching the problem from graph-parallel to data-parallel processing. This paradigm shift is necessary because MapReduce is an effective data-parallel paradigm for computing algorithms that require reading the data only once and, for this reason, it is not efficient to perform join-intensive tasks typical of graph algorithms.

Schätzle et al. (2016) proposed ExtVP (**E**xtended **V**ertical **P**artitioning), a relational partitioning schema for RDF that extends the vertical partitioning (VP) schema and uses a semi-join based preprocessing. The authors also presented S2RDF (**S**PARQL on **S**park for **RDF**), a distributed SPARQL query processor for large-scale RDF data implemented on top of Spark. Schätzle et al. (2016) defined a property graph representation of RDF for GraphX, the Apache Spark's API for graphs and graph-parallel computation. They also introduced S2X (**S**PARQL on **S**park with Graph**X**), an RDF engine that combines graph-parallel abstraction of GraphX to implement the graph pattern matching part of SPARQL with data-parallel computation of Spark to build the results of other SPARQL operators. The results of the comparison of S2X with PigSPARQL (SCHäTZLE et al., 2013) show that the combination of both types of computation can be beneficial, when compared to a purely data-parallel execution.

Abdelaziz et al. (2017) presented a comparative survey of 22 state-of-the-art distributed RDF systems. They described the execution model and the graph partitioning strategy of each system, discussed the similarities and differences, explained the various trade-offs, and categorized the systems based on several characteristics. Then, they selected 12 representative systems and performed a comprehensive experimental evaluation concerning preprocessing cost, query performance, scalability, and workload adaptability, using a variety of synthetic and real large datasets. The results suggest that specialized in-memory systems provide the best performance, assuming the data can fit in the cumulative memory of the computing cluster.

The SPARKSQL RDF Processing Benchmarking[4] is a systematic benchmarking project on the performance of Spark SQL for processing vast RDF datasets. In the first phase of this project, Ragab, Tommasini & Sakr (2019) presented an analysis of the execution time of Spark SQL for answering SPARQL queries over RDF repositories on a centralized single-machine configuration. The authors conducted experiments on datasets with 100K, 1M, and 10M triples and evaluated the impact of using alternative relational schemas for RDF (i.e., ST, VT, and PT), various storage backends (i.e., PostgreSQL, Hive, and HDFS) and different data formats (e.g., CSV, Avro, Parquet and ORC). In the second phase of the project (RAGAB et al., 2020), the experiments include a larger dataset than before in a distributed environment. The authors evaluated the impact of using different RDF-based partitioning techniques (i.e., subject-based, predicate-based, and horizontal-based partitioning). Ragab et al. (2021) extended the previous experiments with new proposed RDF relational schema representations: Extended Vertically Partitioned Tables (ExtVP) and Wide Property Tables (WPT).

The strategies and frameworks described in this thesis take advantage of the ideas presented in the state-of-art works summarized in the previous sections to improve the process of identifying relevant paths between entity pairs using similarity measures and path-ranking approaches to prioritize some paths over others. The framework introduced by Jiménez, Leme & Casanova (2021) implements path search strategies using a multi-thread approach, while that proposed by Jiménez et al. (2021) is a distributed in-memory framework built on top of Apache Spark.

[4]https://datasystemsgrouput.github.io/SPARKSQLRDFBenchmarking/

4 Discovering Relevant Paths between Entity Pairs

This chapter describes the process of discovering relevant relationship paths that connect two entities in an RDF graph using different path search strategies that combine entity similarity and path ranking measures. We propose an approach for doing that on a single-machine configuration using the data parallel paradigm. Finally, we also propose an approach that extends that paradigm to the distributed case, to address the entity relatedness problem on a distributed scenario.

4.1 The Entity Relatedness Problem

4.1.1 Formalization of Problem

The *entity relatedness problem* refers to the question of exploring a knowledge base, represented as an RDF graph, to discover and understand how two entities are connected. An RDF knowledge base R is equivalent to an RDF graph G_R whose nodes represent the entities in R and whose edges denote the relationships expressed in R. This is a convenient representation to explore the connectivity in R of a pair of entities, a and b, which reduces to computing paths in G_R between a and b. Thus, the *entity relatedness problem* can be defined as: "Given an RDF knowledge base R and a pair of entities a and b, compute the paths in G_R from a to b that best describe the connectivity between a and b in R"

The main goal of this thesis is to facilitate the discovery and understanding of the relationship between entities in knowledge bases. Part of the research was motivated by the opportunities for future work identified by Herrera (2017), who presented various strategies to understand the connectivity between pairs of entities in a knowledge base. However, the author did not propose a flexible framework, which developers could extend with additional similarity and path ranking measures, and could be easily used by end-users to experiment with different path search strategies.

4.1.2 Overview of the Proposed Solution

Let G_R be the RDF graph that represents an RDF knowledge base R. We consider a family of path search strategies that receive as input a pair of target entities (w_0, w_k) in G_R and output a list of ranked paths in G_R from w_0 to w_k. Each path search strategy in the family has two basic steps:

1. Find a set of paths in G_R from w_0 to w_k such that each path satisfies a set of selection criteria.
2. Rank the paths found and select the top-k relevant ones.

The first step of a path search strategy uses the backward search heuristic (LE et al., 2014), which is a breadth-first search strategy that expands the paths starting from each target entity, in parallel, until a candidate relationship path is generated. The expansion process considers one or several of the following selection criteria to prioritize certain paths over others and to filter the entities less related to the target entities so that it can be easier to identify more meaningful paths:

Entity similarity: Select a path $(w_0, p_1, w_1, p_2, w_2, \ldots, p_{k-1}, w_{k-1}, p_k, w_k)$ iff there is $q \in [0, k]$ such that, for each $i \in [0, q)$, w_i and w_{i+1} are similar and, for each $j \in [q, k)$, w_j and w_{j+1} are similar.

Bounded entity degree: Select a path whose intermediate entities, or at least those intermediate entities that need to be expanded, have less than n neighbors in G_R.

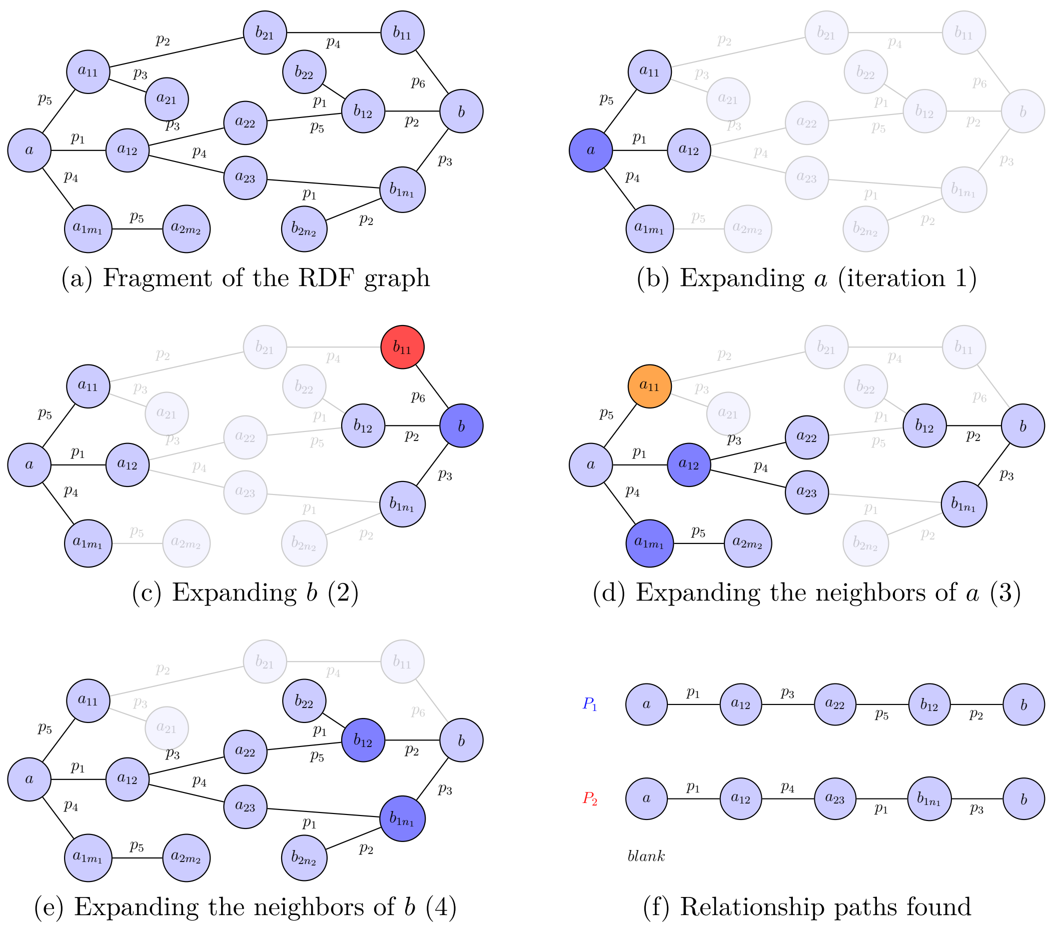
Bounded path length: Select a path of maximum length equal to k.

The first criterion says that a path can be broken into two parts, $left$ and *right*, such that the entities on the $left$ part are transitively similar to the first entity, w_0, and the entities on the *right* part are transitively similar to the second entity, w_k. This criterion maintains entities that are similar to the last entity reached in a partially constructed path, using some similarity measure, and can be implemented by a backward search strategy, as described in the next section. This thesis considers the three entity similarity measures, Jaccard index, WLM, and SimRank, described in Section 2.2.

This thesis also assumes that the bounded entity degree criterion is always applied together with the entity similarity criterion because, as stated Fang et al. (2011) and Moore, Steinke & Tresp (2012), it might be very expensive to expand nodes with a large degree and it can also be assumed that nodes with high degree influence the path search process with potentially very unspecific information.

Paths between target entities can have an arbitrary length. However, considering only paths of length at most k leads to relationship paths of manageable size that users can better interpret. Related approaches, such as REX (FANG et al., 2011), EXPLASS (CHENG; ZHANG; QU, 2014), RECAP (PIRRò, 2015), and DBPEDIA PROFILER (HERRERA et al., 2016), also considered bounded-length paths.

Figure 4.1 shows a simple example of the execution of the backward search to find the relationship paths of maximum size equal to 4 between two entities a and b in an RDF graph. It is important to note that the example only presents a fragment of the entire RDF graph (Figure 4.1a).



(a) Fragment of the RDF graph (b) Expanding a (iteration 1) (c) Expanding b (2) (d) Expanding the neighbors of a (3) (e) Expanding the neighbors of b (4) (f) Relationship paths found

Figure 4.1: Backward search execution example

In the first iteration (Figure 4.1b), the entity a is expanded and the neighbors most similar to a (i.e., a_{11}, a_{12}, a_{1m_1}) are selected to be expanded in a later expansion step. During the expansion of b, in the second iteration (Figure 4.1c), the entity b_{11} is discarded and will not be expanded in a later expansion step as it is not included among the entities most similar to b. Only entities b_{12} and b_{1n_1} are selected to be expanded in a later expansion step. In the third iteration (Figure 4.1d), the neighbors of the entity a selected during the first iteration are expanded, except for the entity a_{11} which does not satisfy the entity degree limit criterion, that is, it has a high number of neighbors. In the fourth and final iteration (Figure 4.1e), entities b_{12} and b_{1n_1} are expanded. Finally, it is verified that there are entities that were reached by the expansions from the left and from the right (i.e., a_{22} and $b_{a_{23}}$), so that the sub-paths that reach these entities can be joined to generate the paths from a to b (Figure 4.1f).

The second step of a path search strategy receives as input the set of paths found in the first step and uses a path ranking measure to sort the paths by relevance. Each of these paths is a possible explanation of how the two input entities are related. This thesis considers the three path ranking measures, PF-ITF, EBR, and PMI, reviewed in Section 2.3.

Just to exemplify, we present some of the paths that could be found in the DBpedia after using a strategy like that described in this section.

The entities `dbr:Elizabeth_Taylor` and `dbr:Richard_Burton` are directly related by the path `(dbr:Elizabeth_Taylor, dbo:spouse, dbr:Richard_Burton)`. This is a very simple path, but it is relevant because being married is a very exclusive relationship. Other relationship paths that connect these two entities are:

```
– (dbr:Elizabeth_Taylor,
       ^dbo:starring, dbr:Doctor_Faustus_(1967_film),
       dbo:producer, dbr:Richard_Burton)
– (dbr:Elizabeth_Taylor,
       ^dbo:starring, dbr:Love_Is_Better_Than_Ever,
       dbo:director, dbr:Stanley_Donen,
       ^dbo:director, dbr:Staircase_(film),
       dbo:starring, dbr:Richard_Burton)
```

In this case, the paths pass through entities representing movies or film directors, which are not discarded by the pathfinding step of the proposed strategy. Note that, in our approach, the similarity between two entities is not related to the classes or domains of the entities, but rather computed from the feature sets of each entity, which consists of the set of nearby entities.

This behaviour is also illustrated when trying to discover the connectivity between The Beatles and The Rolling Stones, as the following paths show:

```
– (dbr:The_Beatles,
       ^dbo:associatedBand, dbr:Brian_Jones,
       ^dbo:formerBandMember, dbr:The_Rolling_Stones)
– (dbr:The_Beatles,
       ^dbo:artist, dbr:Twist_and_Shout,
dbo:recordLabel, dbr:Atlantic_Records,
^dbo:recordLabel, dbr:The_Rolling_Stones)
```

To create different search strategies to discover relevant relationship paths, we combine entity similarity and path ranking measures to be used in the pathfinding and ranking processes, respectively. Therefore, we obtain a family of 9 path search strategies, presented in Table 4.1, which we will evaluate in Chapter 6. The second column of the table contains the acronym used for each strategy hereafter in the document.

Table 4.1: Path Search Strategies

#	**Acronym**	**Name**
1	J&I	Jaccard index & PF-ITF
2	J&E	Jaccard index & EBR
3	J&P	Jaccard index & PMI
4	W&I	WLM & PF-ITF
5	W&E	WLM & EBR
6	W&P	WLM & PMI
7	S&I	SimRank & PF-ITF
8	S&E	SimRank & EBR
9	S&P	SimRank & PMI

4.2 The CoEPinKB Approach to the Entity Relatedness Problem

In this section, we describe an approach to efficiently discover relevant paths up to a limited size between two entities in an RDF graph. The acronym CoEPinKB comes from the main goal of this approach: to facilitate understanding the **C**onnectivity **o**f **E**ntity **P**airs **in** **K**nowledge **B**ases. Specifically, in this first approach, we internally use SPARQL queries embedded in HTTP requests through a SPARQL endpoint. This proposal is adapted from the algorithm introduced by Herrera (2017). In this section, we also highlight the main differences between the two approaches.

4.2.1 Finding Relationship Paths between Entities in a Knowledge Graph

The backward search heuristic uses breadth-first search (BFS) to explore the neighbors of each target entity. Two BFS, which we call $left$ and $right$, are executed alternately to traverse the RDF graph. In each expansion step, the BFS ignores entities with a high degree (i.e., entities with a large number of incoming and outgoing links) and uses an entity similarity measure to prioritize the entities with a higher similarity score to generate relevant relationship paths. A path is generated if both BFS processes reach a common entity or a target entity and the length of the path does not exceed a set limit. We break backward search into two basic and independent steps: (1) expansion, and (2) join of the paths.

Algorithm 1 describes the implementation of the backward search. The input of the algorithm consists of a pair of entities, a and b, an integer l representing a path length limit, an integer d representing an entity degree limit, an activation function τ, and a real number $\lambda \in [0, 1]$ defining an expansion limit; and the output is a set P of paths between a and b.

```
Algorithm 1: backwardSearch
  Input: a pair of entities $a$ and $b$, a path length limit $l$, an entity
         degree limit $d$, an activation function $\tau$, and a real number
         $\lambda \in [0, 1]$ defining an expansion limit
  Output: a set $P$ of paths from $a$ to $b$
1  $side \leftarrow 0, left \leftarrow 0, right \leftarrow 1$;
2  $V_{left} \leftarrow \{a\}, V_{right} \leftarrow \{b\}$;
3  $P_{left} \leftarrow \emptyset, P_{right} \leftarrow \emptyset$;
4  $length \leftarrow 0$;
5  while $length < l$ do
6      $V_{side}, P_{side} \leftarrow expansion(V_{side}, P_{side}, d, \tau, \lambda)$;
7      $length \leftarrow length + 1$;
8      $side \leftarrow length$ mod 2;
9  $P \leftarrow join(P_{left}, P_{right}, a, b)$;
10 return $P$
```

The value of variable *side* alternates between 0, indicating that the expansion will be applied to the "left side" sub-paths, starting on a, and 1, indicating that the expansion will be applied to the "right side" sub-paths, starting on b. The values of variables $left$ and $right$ are therefore 0 and 1, respectively.

The sets V_{left} (i.e., V_0) and V_{right} (i.e., V_1) store the entities to expand in each iteration from "left" and "right", respectively. The undirected sub-paths generated during the expansion of the entities (Line 6) are stored in main memory in sets P_{left} (i.e., P_0, for the "left side" sub-paths) and P_{right} (i.e., P_1, for the "right side" sub-paths). The algorithm returns a set P of undirected paths between a and b created if there are sub-paths in P_{left} and P_{right} that reach a common entity, or if sub-paths in the "left" ("right") side reach b (a).

This approach for finding relationship paths is adapted from the algorithm proposed by Herrera (2017). In that proposal, the join of sub-paths coming from the left and the right (called *intersection* in that work) is executed after each expansion iteration, which has the disadvantage that paths might be generated repeatedly and must therefore be discarded. The author states that in this way the paths can be consumed without waiting for the completion of the backward search process, however, there is no guarantee that those first generated paths will have greater relevance than the paths that will emerge in later iterations. For this reason, our algorithm awaits the completion of the expansion process and then generates paths with all the sub-paths that start from a different side of the graph and reach a common node. Another difference with Herrera's algorithm is the criterion for stopping the loop where the expansion of the entities is performed. Since Herrera performs an unnecessary additional iteration in its repeat-until control flow statement, the generated sub-paths may result in paths of size greater than the user-defined limit and should be discarded.

Algorithm 2 describes the *expansion* process, which is almost identical to that proposed by Herrera (2017). For each entity w to expand, the algorithm retrieves the neighbors of w (Line 3). If this set of neighbors, represented as an adjacency list, is not already in memory, it retrieves it from the Web and stores it in memory. An important remark is that, if w is not a target entity (i.e., a or b) and w has more than the maximum number of links (*bounded entity degree* criterion), then w will not be expanded and the paths that pass beyond w will be discarded (Line 4).

Algorithm 2: `expansion`

Input: a set V_{side} of entities to expand, a set P_{side} of partial paths, a maximum entity degree d, an activation function τ, and a real number $\lambda \in [0, 1]$ defining an expansion limit

Output: a set V_{new} of activated neighbors of entities in V_{side}, and a set P_{side} of partial paths

```
1  $V_{new} \leftarrow \emptyset$;
2  for $w \in V_{side}$ do
3      $N_w \leftarrow neighborsOf(w)$;
4      if $|N_w| <= d$ or $w$ is a target entity then
5          $S_w \leftarrow \tau(w, N_w)$;
6          Truncate $S_w$ to retain only the first $\lambda$% elements;
7          for $w_s \in S_w$ do
8              Append new sub-paths into $P_{side}$ indexed by $w_s$ by
                 appending the edge from $w_s$ to $w$ to sub-paths of $P_{side}$
                 indexed by $w$;
9              Append $w_s$ to $V_{new}$;
10 return $V_{new}, P_{side}$
```

Algorithm 2 also calls the activation function τ (Line 5) with w and

N_w, the list of neighbors of w, as input. The activation function τ implements an entity similarity measure and it is performed in parallel on the list N_w of neighbors to return a list S_w of neighbors similar to w. That is, the list N_w is divided into smaller fragments that can be processed simultaneously by different threads. The list S_w is ordered by highest similarity and only the first λ percent of the elements of the list are considered (Line 6). For example, if $\lambda = 0.3$, only the first 30% of the elements in the list are considered to extend the sub-paths indexed by w in P_{side} (Lines 7-8). These new extended sub-paths are then indexed by the activated neighbor w_s of w. The activated entities are included in set V_{new} (Line 9) for later expansions. Finally, the sets V_{new} and P_{side} are returned by the algorithm (Line 10).

Algorithm 3 describes the *join* process, which generates undirected paths from a to b through the function *concat* when two undirected sub-paths, one coming from left and the other from the right, reach the same entity w (Lines 2-4). The Boolean expression "$w \in P_{left}$" is true when there is a set of paths, denoted $P_{left}[w]$, in P_{left} that end on w. Likewise, "$w \in P_{right}$" is true when there is a set of paths, denoted $P_{right}[w]$, in P_{right} that start on w. Also, if a path coming from left reaches the target entity (Lines 5-6), or a path coming from right reaches the source entity (Lines 7-8), it is included in the set P of the resulting paths.

Algorithm 3: `join`

Input: a set P_{left} of partial paths from left, a set P_{right} of partial paths from right, and a pair of entities a and b

Output: a set P of paths from a to b

```
1 $P \leftarrow \emptyset$, $left \leftarrow 0, right \leftarrow 1$;
2 for $w \in P_{left}$ do
3     if $w \in P_{right}$ then
4         Append paths resulting from $concat(P_{left}[w], P_{right}[w])$ to $P$;
5 if $b \in P_{left}$ then
6     Append paths in $P_{left}[b]$ to $P$;
7 if $a \in P_{right}$ then
8     Append paths in $P_{right}[a]$ to $P$;
9 return $P$
```

The idea of this algorithm is implemented by Herrera (2017) under the name of *intersection*. The main difference between the two proposals is that Herrera (2017) goes through the list of all the entities visited during the expansion process to later recover and join the sub-paths that reach the same entity from the left and the right, while in our algorithm we eliminate that additional loop. We simply verify that the entities that were reached through the partial paths from the left side were also reached by partial paths from the right, and then concatenate those paths.

4.2.2 Ranking Relationship Paths in a Knowledge Graph

The number of paths connecting two entities a and b in a knowledge base can be very large. To help users understand the connectivity between that pair of entities it is necessary to reduce the size of the resulting path set. In this section, we discuss how to effectively select the most relevant relationship paths between a pair of entities.

Specifically, given a relationship path ranking measure and a parameter k, the relationship path ranking algorithm returns a ranked list of top-k most relevant relationship paths based on the relationship path ranking measure. Algorithm 4 shows the pseudocode of *getRelevantPaths*, which includes three steps: (1) executing the backward search to find the relationship paths (using the Algorithm 1), (2) executing the path ranking function to get an ordered list of ranked paths, and (3) selecting the top-k most relevant paths.

Algorithm 4: `getRelevantPaths`

```
Input: a pair of entities $a$ and $b$, a path length limit $l$, an entity
       degree limit $d$, an activation function $\tau$, a real number
       $\lambda \in [0,1]$ defining an expansion limit, a path ranking function
       $\gamma$, and a maximum number of paths $k$
Output: a list of the top-$k$ relevant paths from $a$ to $b$
1 $P \leftarrow backwardSearch(a, b, l, d, \tau, \lambda)$;
2 $R \leftarrow getPathsOrderedByScore(P, \gamma)$;
3 Truncate $R$ to retain only the first $k$ elements;
4 return R
```

The *getPathsOrderedByScore* function uses the path ranking function γ, which implements some path ranking measure, to calculate the score of each of the paths in the list P. This function runs in parallel to speed up the ranking process. After calculating the score of each discovered path and ordering the list of paths in descending order according to this score, we take the first k elements and this result is the output of the algorithm.

4.3 The DCoEPinKB Approach to the Entity Relatedness Problem

In addition to the approach for discovering relevant paths between two entities in an RDF graph on a single-machine configuration using the data parallel paradigm, we also propose an approach that extends that paradigm to the distributed case, to address the entity relatedness problem on a distributed scenario. The acronym DCoEPinKB stands for a **D**istributed way of understanding the **C**onnectivity **of** **E**ntity **P**airs **in** **K**nowledge **B**ases. Specifically, in this second approach, we use Spark in a local infrastructure that could be replicated on a cluster.

Distributed data processing frameworks require data partitioning to harness the full power of a distributed solution. Spark is not designed to perform native RDF processing, but relational schemas can be used to represent RDF data in an efficient and scalable way. Our approach uses the Statement Table schema that directly maps RDF triples onto a table with three columns (subject, predicate, object), in which each tuple corresponds to an RDF statement. In this way, it is possible to use all the power of Spark SQL and its data structures for distributed collections and take advantage of the efficient execution of distributed and parallel operations such as *map* and *reduce*.

The strategy we propose to discover relevant paths between pairs of entities in distributed RDF graphs is very similar to what we presented previously for a single-machine configuration in the CoEPinKB approach. Similarly, we use the backward search heuristic with activation criteria to find the paths and path ranking measures to select the most relevant ones, however, in this approach we execute some tasks over distributed collections of data in a distributed way.

Algorithm 5 describes in pseudocode the implementation of the expansion process for the DCoEPinKB approach. Note that this algorithm is very similar to Algorithm 2, but, in this case, the request for the list of neighbors of the entities that are waiting to be expanded is made using the *map* and *reduce* operators, which allow distributed processing and can be performed in parallel. Internally, the activation function τ also implements an entity similarity measure using a combination of the *map*, *filter*, and *sortBy* operators to perform similarity calculations in parallel over a distributed data collection.

First, we call the function $neighborsOf$ within the *map* operation for each of the entities in the set of entities to be expanded (Line 2) and then we collect all these neighbors in the distributed data collection N by executing the *reduce* operation (Line 3). We then use a parallel transformation like $filter$ to select exactly the neighbors of a given entity within this distributed collection that contains all neighbors (Line 5). Unlike the expansion process in CoEPinKB, in this new proposal we define an expansion limit that allows selecting a percentage of the most similar entities in the list S_w of neighbors similar to w or an absolute value of those most similar entities. That is, for

Algorithm 5: `expansion (2)`

Input: a set V_{side} of entities to expand, a set P_{side} of partial paths, a maximum entity degree d, an activation function τ, and a real number λ defining an expansion limit

Output: a set V_{new} of activated neighbors of entities in V_{side}, and a set P_{side} of partial paths

```
1  $V_{new} \leftarrow \emptyset$;
2  $N \leftarrow map(V_{side}, w => neighborsOf(w))$;
3  $N \leftarrow reduce(N, union)$;
4  for $w \in V_{side}$ do
5      $N_w \leftarrow filter(N, w)$;
6      if $|N_w| <= d$ or $w$ is a target entity then
7          $S_w \leftarrow \tau(w, N_w)$;
8          Truncate $S_w$ to satisfy the expansion limit criterion;
9          for $w_s \in S_w$ do
10             Append new sub-paths into $P_{side}$ indexed by $w_s$ by
                 appending the edge from $w_s$ to $w$ to sub-paths of $P_{side}$
                 indexed by $w$;
11             Append $w_s$ to $V_{new}$;
12 return $V_{new}, P_{side}$
```

example, 50% of the most similar entities or the top-10 most similar entities can be expanded.

Similarly, the computation of the score of each of the paths found can be carried out in a distributed manner using the *map* operation, as shown in Algorithm 6. The path ranking function γ, which implements a specific path ranking measure, is executed in parallel within the *map* operation (Line 1).

Algorithm 6: `getPathsOrderedByScore`

Input: a set P of paths from a to b, a path ranking function γ

Output: a list R of paths from a to b ordered descending by the score

```
1  $R \leftarrow map(P, path => (path, \gamma(path)))$;
2  $R \leftarrow sort(R, reverse)$;
3  return R
```

4.4 Chapter Conclusions

In this chapter, we presented a solution to the entity relatedness problem. We combined entity similarity and path ranking measures to generate a family of 9 path search strategies that receive as input a pair of target entities a and b and output a ranked list of paths in an RDF graph G from a to b. Each path search strategy in the family has two basic steps: (1) find a set of paths in G from a to b such that each path satisfies a set of selection criteria; (2) rank the paths found and select the top-k relevant ones. We proposed an approach to address this problem in a single-machine configuration based on a data-parallel paradigm, called CoEPinKB, and another approach that extends that strategy for distributed data-parallel execution, called DCoEPinKB.

5
Implementation

This chapter presents the technical aspects of CoEPinKB and DCoEPinKB, two frameworks that implement the approaches proposed in Sections 4.2 and 4.3, respectively. First, we present in Section 5.1 an overview of the general architecture of the frameworks. Then, we describe in Sections 5.2.1 and 5.3.1 the specific architecture and workflow process of each framework. We also report some details about the technologies and programming paradigms used. Finally, in Sections 5.2.2 and 5.3.2 we present the user interface of these frameworks that allow a user to submit an entity pair and a search strategy for searching the relevant paths that link both entities and return a list of ranked paths between this pair of entities.

5.1
Overview

Frameworks are semi-complete, reusable applications that can be specialized to produce custom applications for a specific domain. The flexibility points of a framework, called hot spots, are the interfaces, abstract classes, or methods that must be implemented to add the functionality specific to a problem (MARKIEWICZ; LUCENA, 2001). Some features of the framework are not mutable and are known as frozen spots. These points of immutability compose the kernel of the framework and are pieces of code already implemented within the framework that call one or more hot spots.

Figure 5.1 shows the general architecture of the two proposed frameworks. The architecture is divided into three main layers (i.e., presentation, business, and data layers) and the cross-cutting layer that allows communication between them.

The main layers can be summarized as follows:

Presentation Layer: contains all of the classes responsible for presenting the user interface to the end-user or sending the response to some external system through a service interface.

Business Layer: contains all the logic that is required by the framework to meet its functional requirements. The main components of the workflow process, algorithms, and interfaces reside in this layer.

Data Layer: contains all the classes responsible for accessing, transforming, and persisting data.

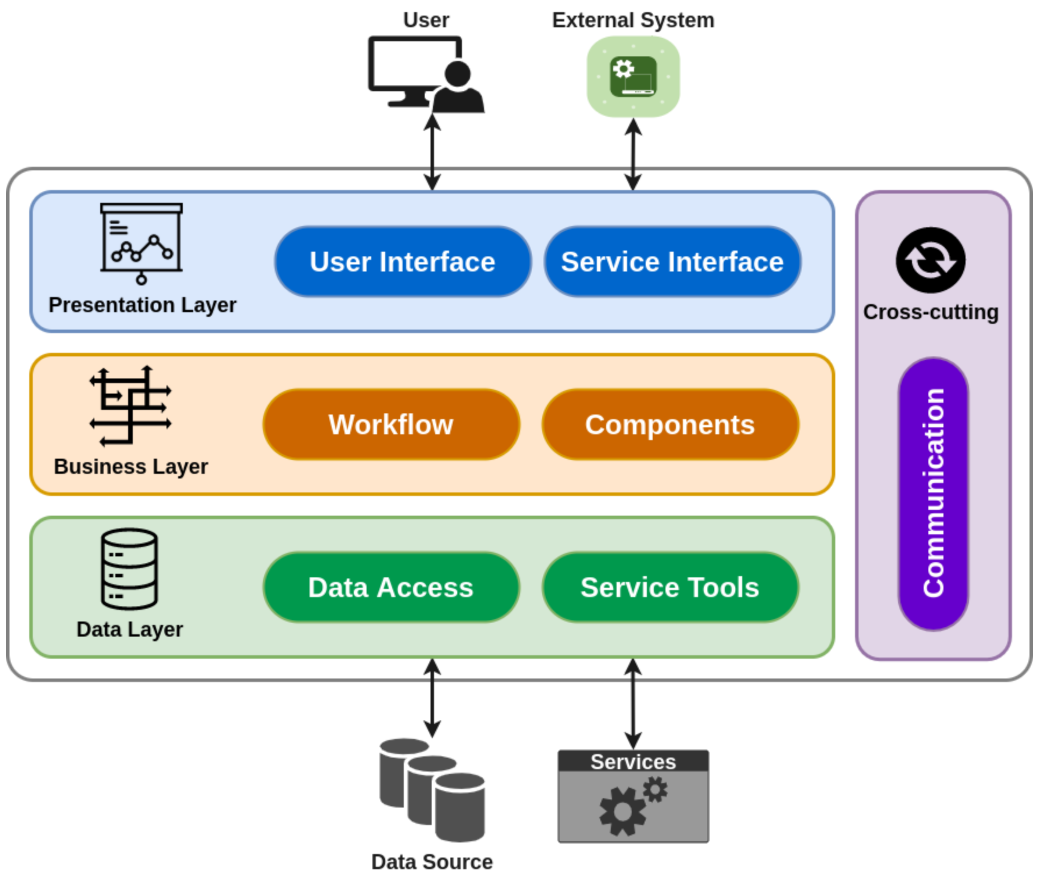


Figure 5.1: General Architecture

The main benefit of developing a layered and modular architecture is the ease of reusability and extensibility of our frameworks to produce new tools that address the entity relatedness problem. The proposed frameworks have some hot spots in the business and data layers for developers to easily add new entity similarity and relationship path ranking measures to generate new path search strategies, as well as to work with different knowledge bases.

5.2 The CoEPinKB Framework

The CoEPinKB[1] framework was implemented in Java in conjunction with other technologies, such as Apache Jena[2], a free and open-source Java framework for building Semantic Web and Linked Data applications, to interact with the RDF data sources; Redis[3], a popular distributed in-memory key-value store (solid IT, 2020), as our persistent cache; and the Jedis[4] library, which allowed us to interact with a Redis instance from our Java application.

[1]The source code of CoEPinKB is available at https://bitbucket.org/guillot/coepinkb/
[2]https://jena.apache.org/
[3]https://redis.io/
[4]https://github.com/redis/jedis

In this section, we present the architecture, workflow, and user interface of CoEPinKB, as well as some details of the implementation that allowed us to parallelize certain computations.

5.2.1 Architecture

As we described earlier in Section 4.2, our approach to address the entity relatedness problem is to apply a two-step strategy, and each of these two sequential phases corresponds to two of the main components of CoEPinKB: the Backward Search component, which executes a breadth-first search starting from each input entity and expanding similar entities to find the most relevant relationship paths; and the Relationship Path Ranking component, which ranks the resulting paths of the previous step. A third component, called SPARQL Query Executor, implements the execution of SPARQL queries to communicate with a SPARQL endpoint and to save the result of those queries in a persistent cache. The first two components belong to the business layer, while the last one belongs to the data layer. Figure 5.2 shows an overview of the architecture of the CoEPinKB framework and its workflow process.

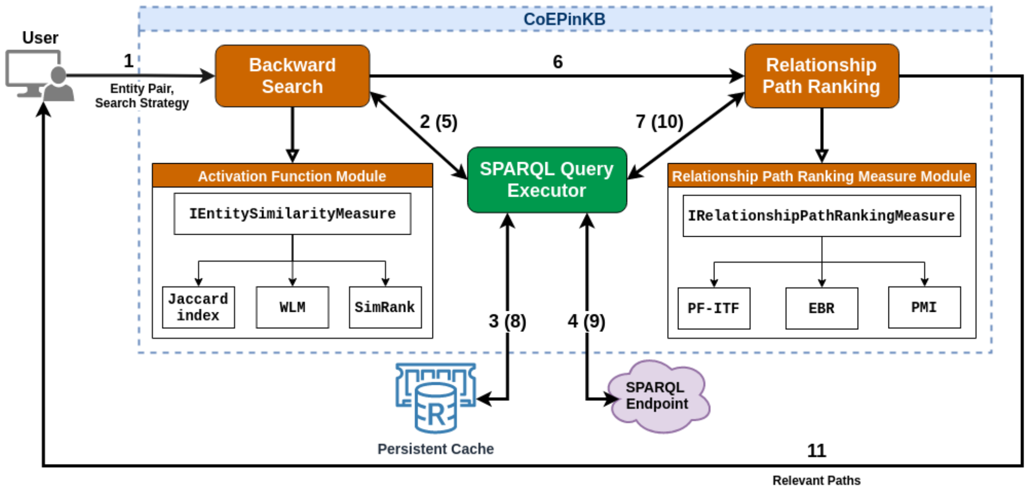


Figure 5.2: CoEPinKB Architecture & Workflow

The workflow process in CoEPinKB goes as follows:

1. The CoEPinKB framework takes as input a pair of entities and a search strategy. A search strategy consists of an entity similarity measure that will be used by the backward search algorithm as the activation function to decide when to expand some neighbor of an entity or not, and a relationship path ranking measure to select the top-k most relevant paths between the two entities provided.

2. The Backward Search component communicates with the SPARQL Query Executor component requesting the required data to execute the backward search algorithm.
3. The SPARQL Query Executor component tries to get the requested data from the persistent cache.
4. If the requested data is not available in the persistent cache (Step 3), then the SPARQL Query Executor component gets the data directly from the SPARQL Endpoint through SPARQL queries, and stores it in the persistent cache to speed up future searches.
5. The SPARQL Query Executor component sends the requested data to the Backward Search component. (Steps 2-5 will repeat until the backward search algorithm completes.)
6. The Backward Search component sends a list of relationship paths between the pair of entities to the Relationship Path Ranking component.
7. Similarly to the previous phase, the Relationship Path Ranking component communicates with the SPARQL Query Executor requesting the required data to execute the path ranking algorithm.
8. The SPARQL Query Executor component tries to get the requested data from the persistent cache.
9. If the requested data is not available in the persistent cache, then the SPARQL Query Executor component gets the data directly from the SPARQL Endpoint through SPARQL queries, and stores it in the persistent cache to speed up future queries.
10. The SPARQL Query Executor component sends the requested data to the Backward Search component. (Steps 7-10 will repeat until the path ranking algorithm completes.)
11. Finally, the Relationship Path Ranking component sends the list of ranked paths to the user through the user interface.

There are two key hot spots in the CoEPinKB framework –the activation function, implementing the entity similarity measure, and the path ranking measure– which are the core of the Backward Search and Relationship Path Ranking components. These components were designed using an architectural pattern based on interfaces, which increases the extensibility of the framework by making it easier to add new entity similarity measures and relationship path ranking measures.

The backward search algorithm calls the method `getSimilarity` from a class that implements the interface `IEntitySimilarityMeasure`. Likewise, the path ranking process executes the method `getPathsOrderedByScore` from a class that implements the interface `IRelationshipPathRankingMeasure`. As illustrated in Figure 5.2, the current version of CoEPinKB implements three entity similarity measures (i.e., Jaccard index, WLM, and SimRank) and three relationship path ranking measures (i.e., PF-ITF, EBR, and PMI).

At the data layer, the framework has the SPARQL Query Executor component that uses Apache Jena's ARQ API to interact with RDF datasets through their SPARQL endpoints. In this way, local data availability and complex data processing infrastructure are not required. Data processing is reduced to evaluating a set of queries through the SPARQL endpoint, as well as locally executing some computations on the results of those queries.

Local data are optionally required to be used as a cache to speed up queries. For this, the framework includes a persistent cache to store the result of the SPARQL queries executed during the expansion of the entities in the RDF graph. The main reason for this decision is that the backward search and the path ranking algorithms require executing a large number of queries (quite possibly over the network), which can negatively affect the overall performance of the framework. We use Redis to implement the persistent cache and, since the data in the RDF knowledge base can evolve, we define an expiration time equivalent to one week for data stored in the cache. The source code below shows an example of the use of key expiration times in Redis when storing the adjacency list of an entity (i.e., the neighbors of the entity).

Code 2: Using key expiration time for data in the cache

```
private final int REDIS_KEY_EXPIRATION_TIME = 604800; // 1 week
...
public ResultSetMem getNeighborsOfResource(String iri) {
  String key = String.format("%s:neighborsOf:%s", paramHash, iri);
  if (jedis.get(key) == null) {
    ResultSetMem neighbors = new ResultSetMem(
        getOutgoingLinksFromResource(iri),
        getIncomingLinksToResource(iri));
    jedis.set(key, ResultSetFormatter.asXMLString(neighbors));
    jedis.expire(key, REDIS_KEY_EXPIRATION_TIME);
    return neighbors;
  }
  return new ResultSetMem(ResultSetFactory.fromXML(jedis.get(key)));
}
```

To improve performance, in addition to the persistent cache, we use concurrent programming in the implementation of the entity similarity and path ranking measures through the fork/join framework in Java. Following a divide and conquer approach, we split intensive tasks, such as computing the similarity between an entity and its neighbors, into smaller independent subtasks that can be performed in parallel to maximize the use of multi-core processors, as shown in Figure 5.3.

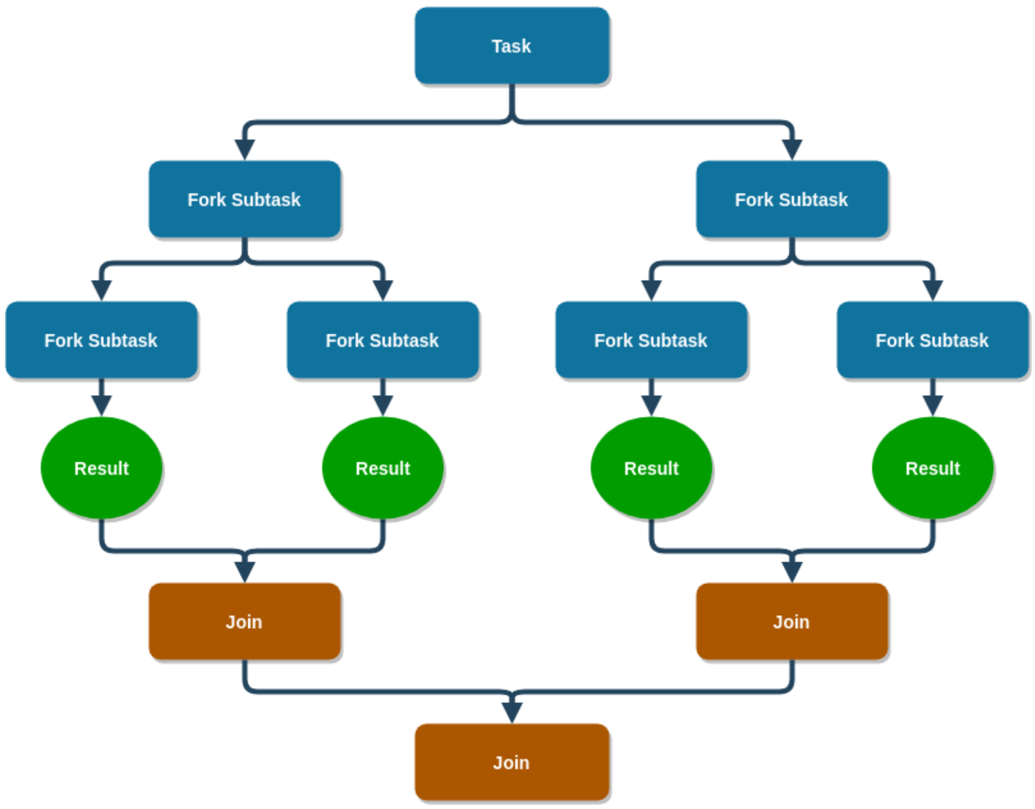


Figure 5.3: Fork/Join Example

The source code below shows an example of the use of the fork/join framework to compute the JACCARD INDEX. The same logic applies to computing the entity similarity using the other measures.

Code 3: Using the fork/join framework to compute the JACCARD INDEX

```
protected List<Similarity> compute() {
  List<Similarity> result = new ArrayList<>();
  if (neighbors.size() < THRESHOLD)
    for (QuerySolution neighbor : neighbors)
      result.add(getSimilarity(entity, neighbor));
  else {
    int middle = neighbors.size() / 2;
    List<QuerySolution> l1 = neighbors.subList(0, middle);
    List<QuerySolution> l2 = neighbors.subList(middle, neighbors.
        size());
    JaccardIndexTask task1 = new JaccardIndexTask(entity, l1);
    JaccardIndexTask task2 = new JaccardIndexTask(entity, l2);
    task1.fork();
    result.addAll(task2.compute());
    result.addAll(task1.join());
  }
  return result;
}
```

In this example, if the task of computing the similarity between an entity and its neighbors is simple enough (i.e., `neighbors.size()` is lower than a specified threshold), then the task is executed asynchronously. Otherwise, the list of neighbors is divided into sublists (the task is divided into subtasks) and the results of all subtasks are recursively joined into a single result.

5.2.2 User Interface

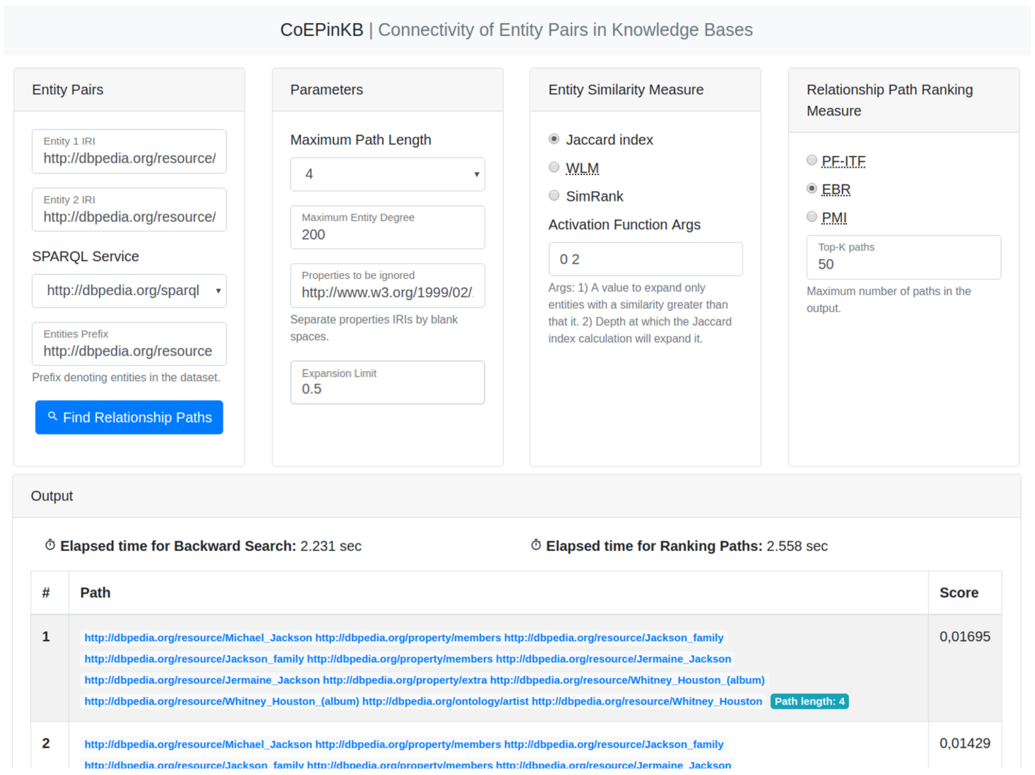
A framework that facilitates the understanding of the connectivity between pairs of entities in knowledge bases using different search strategies requires a simple and at the same time highly configurable interface in terms of the parameters that make up a search strategy. The CoEPinKB interface[5] was implemented using Java servlets and JSP technologies. Figure 5.4 shows an example of the presentation of the results when the input entities are `dbr:Michael_Jackson` and `dbr:Whitney_Houston`, and the entity similarity and relationship path ranking measures are Jaccard index and EBR, respectively.

The user also specifies other parameters through the interface such as the maximum path length between the entities (set to 4 by default, but the user can set this parameter to search for shorter/longer paths); the maximum entity degree, to discard entities with a high number of neighbors during the expansion; a list of properties irrelevant when building the relationship paths; an entity prefix, to expand only to resources that are considered entities; an expansion limit $\lambda \in [0, 1]$, understood as a percentage, that limits the expansion process; and the maximum number of paths that the user wants.

The results are presented using a table layout and the relevant paths are ordered in descending order by the score. The interface also allows the user to navigate to the page of the resource –subject, predicate, or object– by clicking on the corresponding URI.

CoEPinKB also provides a RESTful API, so the user can submit a GET request that returns a JSON document containing the corresponding list

[5]http://semanticweb.inf.puc-rio.br:8080/CoEPinKB/



CoEPinKB | Connectivity of Entity Pairs in Knowledge Bases

Entity Pairs

Entity 1 IRI: http://dbpedia.org/resource/

Entity 2 IRI: http://dbpedia.org/resource/

SPARQL Service: http://dbpedia.org/sparql

Entities Prefix: http://dbpedia.org/resource

Prefix denoting entities in the dataset.

Parameters

Maximum Path Length: 4

Maximum Entity Degree: 200

Properties to be ignored: http://www.w3.org/1999/02/

Separate properties IRIs by blank spaces.

Expansion Limit: 0.5

Entity Similarity Measure

[✓ Jaccard index] WLM SimRank

Activation Function Args: 0 2

Args: 1) A value to expand only entities with a similarity greater than that it. 2) Depth at which the Jaccard index calculation will expand it.

Relationship Path Ranking Measure

PF-ITF [✓ EBR] PMI

Top-K paths: 50

Maximum number of paths in the output.

Output

Elapsed time for Backward Search: 2.231 sec

Elapsed time for Ranking Paths: 2.558 sec

#	Path	Score
1	http://dbpedia.org/resource/Michael_Jackson http://dbpedia.org/property/members http://dbpedia.org/resource/Jackson_family http://dbpedia.org/resource/Jackson_family http://dbpedia.org/property/members http://dbpedia.org/resource/Jermaine_Jackson http://dbpedia.org/resource/Jermaine_Jackson http://dbpedia.org/property/extra http://dbpedia.org/resource/Whitney_Houston_(album) http://dbpedia.org/resource/Whitney_Houston_(album) http://dbpedia.org/ontology/artist http://dbpedia.org/resource/Whitney_Houston Path length: 4	0,01695
2	http://dbpedia.org/resource/Michael_Jackson http://dbpedia.org/property/members http://dbpedia.org/resource/Jackson_family http://dbpedia.org/resource/Jackson_family http://dbpedia.org/property/members http://dbpedia.org/resource/Jermaine_Jackson	0,01429

Figure 5.4: CoEPinKB User Interface

of relevant paths between the two entities. This form of interaction with the framework makes it easy to execute batch searches and perform experiments.

5.3 The DCoEPinKB Framework

The DCoEPinKB[6] framework was implemented in Scala with the help of some other technologies, such as Apache Spark; Redis, as our persistent cache; and the `scala-redis`[7] library, for connecting to a Redis server. In this section, we describe the architecture and user interface of DCoEPinKB and mainly discuss the characteristics that differ between this framework and CoEPinKB. We do not delve so deeply into those characteristics that are similar in the two frameworks and that we already discussed in Section 5.2.1.

5.3.1 Architecture

The acronym DCoEPinKB stands for a **D**istributed way of understanding the **C**onnectivity **of** **E**ntity **P**airs **in** **K**nowledge **B**ases. As we mentioned earlier in Section 4.3, the DCoEPinKB approach is very similar to the

[6]The source code of DCoEPinKB is available at https://bitbucket.org/guillot/dcoepin-kb/

[7]https://github.com/debasishg/scala-redis

CoEPinKB approach, but this new framework addresses the entity relatedness problem in a distributed manner using Apache Spark (ZAHARIA et al., 2010) for large-scale data processing.

Spark has a programming model similar to MapReduce but extends it with a data-sharing abstraction called *Resilient Distributed Datasets*, or RDDs. Under the hood, these RDDs are stored in partitions on different cluster nodes. A partition is the main unit of parallelism in Spark and basically, it is a logical chunk of a large distributed dataset. It provides the possibility to distribute the work across the cluster, divide the task into smaller parts and reduce memory requirements for each node.

The DCoEPinKB framework, unlike CoEPinKB, has a distributed local infrastructure to store the knowledge base on which the user intends to address the entity relatedness problem. Figure 5.5 shows an overview of the architecture of DCoEPinKB.

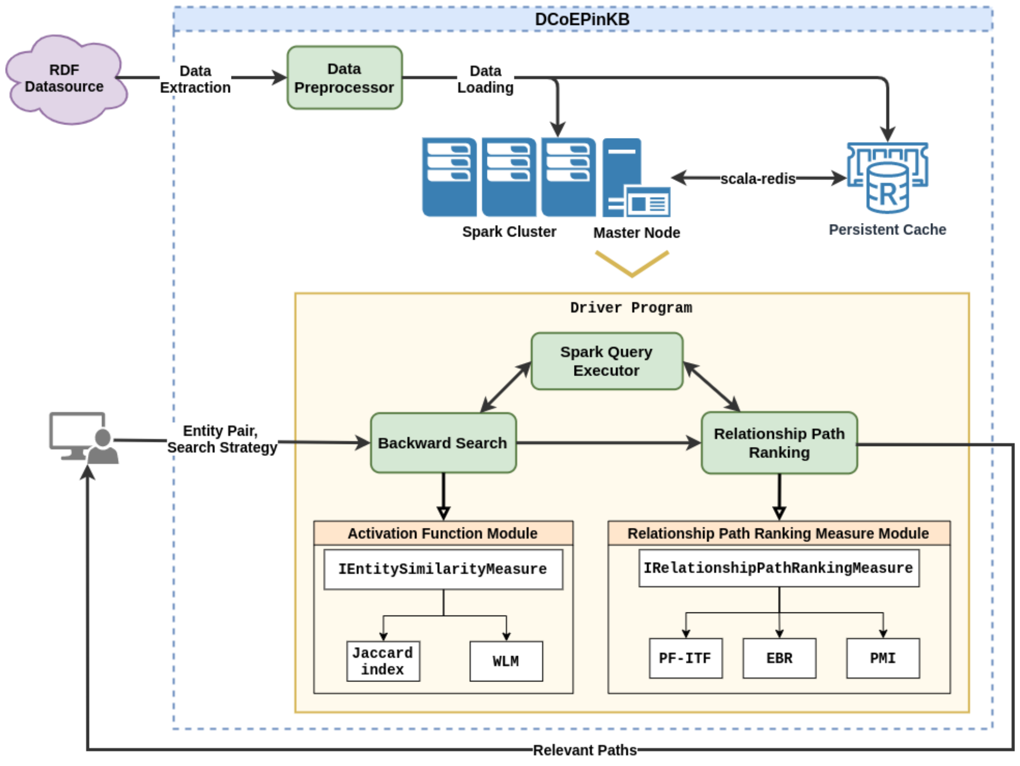


Figure 5.5: DCoEPinKB Architecture

Spark is not designed to perform native RDF processing. However, relational schemas such as the Statement Table, the Property Table, and the Vertical Partitioning schemas can be used to represent RDF data. The Data Preprocessor component transforms the source files of an RDF knowledge base into files in the Parquet format using some RDF relational schema, partitions these new files into fragments, and distributes the fragments over a cluster. Apache Parquet[8] is a columnar storage format that provides optimizations to speed up queries and is a much more efficient file format than CSV or JSON, supported by many data processing frameworks. It provides flexible and efficient data compression and encoding schemes with enhanced performance. As the data type for each column is quite similar –we store strings that represent the resources identifiers– the compression of each column is straightforward.

At the data layer, the framework also has the Spark Query Executor component that interacts with DataFrames that represent views of RDF datasets stored in the distributed file system as Parquet files. Based on what was developed for the SPARQL Query Executor component in CoEPinKB, we made translations of implemented SPARQL queries to Spark SQL queries. Spark SQL is one of the most popular modules of Spark, targeted for processing structured data, using the Datasets and DataFrames data abstractions, and provides support for reading and writing Parquet files.

DCoEPinKB also uses Redis to include a persistent cache to store the result of the queries executed during the expansion of the entities in the RDF graph. We designed the structure of the keys in the cache in a way that facilitates partitioning data on a cluster of Redis nodes using a concept called *hash tags*[9]. So, we can use these *hash tags* to force certain keys to be stored in the same hash slot.

After the data preprocessing stage and with the RDF graph ready to be queried, the two-step strategy to search for the most relevant paths between a pair of entities can start. First, the user enters a pair of entities and specifies a path search strategy by selecting an entity similarity measure, together with an expansion limit, and a path ranking measure. The user also specifies other parameters such as the maximum path length between the entities; the maximum entity degree, to discard entities with a high number of neighbors during the expansion; a list of properties irrelevant to the analysis when building the relationship paths; and an entity prefix, to expand only to resources that are considered entities.

During the first phase of the execution of DCoEPinKB, the Backward Search component communicates with the Spark Query Executor component requesting the required data to execute the backward search algorithm. This last component gets the requested data using two different approaches: (i) first, it tries to get the data from the persistent cache; (ii) if the requested data is not available then it gets the data directly from the Dataframe object

[8]https://parquet.apache.org/
[9]https://redis.io/topics/cluster-spec

in Spark that represents a view of the data available in the Parquet file, and stores it in the persistent cache to speed up future searches.

After the backward search algorithm finishes, the Backward Search component sends a list of relationship paths between the pair of entities to the Relationship Path Ranking component. Then, the second phase begins. Similar to the previous phase, the Relationship Path Ranking component communicates with the Spark Query Executor component requesting the required data to execute the path ranking algorithm. After the algorithm finishes, the Relationship Path Ranking component sends the list of ranked paths to the user through the user interface.

Similar to CoEPinKB, there are two main hot spots in the DCoEPinKB framework: the activation function, implementing the entity similarity measure, and the path ranking measure. The Backward Search and the Relationship Path Ranking components were designed using an architectural pattern based on interfaces, specifically using *traits* in Scala. As illustrated in Figure 5.5, the current version of DCoEPinKB implements two entity similarity measures (i.e., Jaccard index and WLM) and three relationship path ranking measures (i.e., PF-ITF, EBR, and PMI).

5.3.2 User Interface

Unlike the CoEPinKB framework, this first version of DCoEPinKB runs in batch mode and the user interacts with the framework using the console. A goal for future work is to improve the framework to run in interactive mode with a graphical user interface, probably using Apache Livy[10]. Apache Livy is a service that enables easy interaction with a Spark cluster over a REST interface and simplifies the interaction between Spark and application servers, thus enabling the use of Spark for interactive web/mobile applications. Currently, the user creates a simple text file that contains the input parameters of the algorithm and that the framework uses to execute the search strategy. The results are then returned in CSV files.

5.4 Chapter Conclusions

In this chapter, we presented the architecture, the interface, and the most important implementation details of the CoEPinKB and DCoEPinKB frameworks. The architectures of these frameworks correspond to the approaches proposed in Sections 4.2 and 4.3 to address the entity relatedness

[10]https://livy.apache.org/

problem. The corresponding user interfaces allow the user to combine entity similarity and path ranking measures to execute different path search strategies between a pair of entities in a knowledge base.

Table 5.1 compares the CoEPinKB and DCoEPinKB frameworks with related systems introduced in Chapter 3 in terms of knowledge graphs supported, types of output, availability of filtering capabilities, the requirement of local data, and architecture.

Table 5.1: Comparison of the proposed frameworks with related systems

System	**Knowledge Graph**	**Output**	**Filtering Capabilities**	**Local Data**	**Architecture**
RelFinder	DBpedia	Graph	No	Yes	Centralized
REX	Yahoo!	Graph	No	Yes	Centralized
Explass	DBpedia	Paths	Yes	Yes	Centralized
RECAP	Any	Graph, Paths	Yes	No	Centralized
DBpedia Profiler	DBpedia	Graph, Paths	No	Yes	Centralized
CoEPinKB	Any	Paths	Yes	No[*]	Centralized
DCoEPinKB	Any	Paths	Yes	Yes	Cluster

[*] Local data is only necessary to be used as a cache to speed up queries, but it is not mandatory.

CoEPinKB and DCoEPinKB differ from most related systems in the following major aspects. As for the RDF knowledge base, only RECAP and our frameworks are knowledge base independent, that is, these approaches are flexible enough to be used with different knowledge bases; CoEPinKB, as RECAP, only requires the availability of a remote SPARQL query endpoint, while DCoEPinKB pre-processes any RDF dataset and transforms it into Parquet files that can be later consumed by the framework. Regarding the local data requirement, CoEPinKB and RECAP do not assume local data availability or any data pre-processing. However, by using a local cache, CoEPinKB can speed up the execution of the queries. Finally, DCoEPinKB is the only approach that addresses the entity relatedness problem in a distributed manner.

6
Evaluation

This chapter presents an evaluation of the path search strategies proposed in Chapter 4 using the CoEPinKB and DCoEPinKB frameworks over the DBpedia. First, we describe the experimental setup adopted in the experiments: the hardware and software configurations, the datasets, and parameter settings. Then, we execute the experiments over two entertainment domains (i.e., music and movies) over real data available in the DBpedia. We evaluate the performance, in terms of average execution time, of the two frameworks, and the ranking accuracy of the path search strategies for different expansion limit values in DCoEPinKB. The evaluation of the DCoEPinKB framework required the use of novel ground truth, In this chapter, we also describe how we created this ground truth. Finally, we present the results of the experiments, which provide insights into the performance of the path search strategies.

6.1
CoEPinKB Evaluation

Herrera (2017) executed some experiments to evaluate the family of nine path search strategies shown in Table 4.1 against a ground truth (HERRERA et al., 2017) from the music and movies domains, and a baseline, RECAP (PIRRò, 2015). The pairwise comparison method was used to identify the path search strategy that returns the best ranking compared with the ground truth, and to compare the best strategy with the baseline. Herrera (2017) did not include experiments to evaluate the performance of all these strategies concerning execution time. In this section, we present a performance evaluation of these different strategies using CoEPinKB.

6.1.1
Experimental Setup

This section describes the experimental environment, hardware and software components, dataset, and parameter settings for experimenting with the CoEPinKB framework.

6.1.1.1
Hardware and Software Configurations

All the experiments were performed on a Linux server with Ubuntu 16.04.7 LTS system, an Intel® Core™ i7-5820K CPU @ 3.30GHz, and 6GB of memory dedicated to Java applications. We used Java v11.0.10, Tomcat v.9.0.36, and Redis v3.0.6.

6.1.1.2 Dataset

The experiments were carried out over DBpedia (LEHMANN et al., 2015), a well-known large public knowledge base which data is extracted from Wikipedia infoboxes. DBpedia constitutes the main resource of Linked Open Data on the Web containing more than 228 million entities to date[1]. The CoEPinKB framework queries the DBpedia dataset online via the public OpenLink Virtuoso SPARQL protocol endpoint at http://dbpedia.org/sparql. OpenLink Virtuoso serves as both the back-end database SPARQL query engine and the front-end HTTP/SPARQL server with an Nginx overlay primarily to cache results for each submitted query string. This public endpoint does not include all available DBpedia datasets[2]. When the experiments were performed, this dataset contained just over 400 million triples.

6.1.1.3 Target Entity Pairs

We selected 10 entity pairs from the Entity Relatedness Test Dataset (HERRERA et al., 2017), 5 entity pairs from the music domain, and 5 from the movies domain. Table 6.1 shows the selected entity pairs from both domains.

Table 6.1: Entity pairs selected for experimenting with CoEPinKB

#	Entity Pair (music domain)	#	Entity Pair (movies domain)
1	dbr:Michael_Jackson	6	dbr:Elizabeth_Taylor
	dbr:Whitney_Houston		dbr:Richard_Burton
2	dbr:The_Beatles	7	dbr:Cary_Grant
	dbr:The_Rolling_Stones		dbr:Katharine_Hepburn
3	dbr:Elton_John	8	dbr:Laurence_Olivier
	dbr:George_Michael		dbr:Ralph_Richardson
4	dbr:Led_Zeppelin	9	dbr:Errol_Flynn
	dbr:The_Who		dbr:Olivia_de_Havilland
5	dbr:Pink_Floyd	10	dbr:William_Powell
	dbr:David_Gilmour		dbr:Myrna_Loy

[1]https://www.dbpedia.org/
[2]https://www.dbpedia.org/resources/sparql/

6.1.1.4
Path Search Strategies

Using CoEPinKB, we proceeded to evaluate the family of nine path search strategies obtained by combining three entity similarity measures (Jaccard index, WLM, and SimRank) and three path ranking measures (PF-ITF, EBR, and PMI), as shown in Table 4.1.

6.1.1.5
Configuration parameters

We configured the experiments using the following parameters:

Maximum path length between the entities: set to 4, since this was the limit adopted by previous works, as REX (FANG et al., 2011), EXPLASS (CHENG; ZHANG; QU, 2014), RECAP (PIRRò, 2015), DBpedia Profiler (HERRERA et al., 2016), and the experiments in Herrera (2017).

Maximum entity degree: set to 200. This degree limit was deduced from DBpedia statistics (HERRERA, 2017), which indicate that 90% of the entities have less than 200 links. This kind of criterion is applied together with entity similarity during the entity expansion process because, as in Moore, Steinke & Tresp (2012), it can be assumed that nodes with a high degree often carry very unspecific information that negatively influences the path search process.

Expansion limit: set to $\lambda = 0.5$. So, 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 similarity score in the graph (COHEN, 2010).

Set of ignored properties: a total of 10 properties were ignored during the exploration of the knowledge base. These properties are: `purl:subject`, `rdfs:seeAlso`, `rdf:type`, `dbo:type`, `dbo:wikiPageRedirects`, `dbo:wikiPageDisambiguates`, `dbp:aux`, `dbp:name`, `dbp:title`, `dbp:wordnet_type`, and `dbp:governmentType`. This is justified by the fact that these properties describe relationships between entities that are irrelevant for our analysis.

For instance, if we considered properties like `purl:subject` and `rdf:type`, we would have to deal with too many paths that are of little interest to us. There are more than 225 statements in which the subject is the entity `dbr:Michael_Jackson` and the predicate is one of these properties. The property `dbo:wikiPageRedirects` is also present in many statements (almost 70 times in the case that the entity `dbr:Michael_Jackson` is the object) that mainly link entities with typographical errors or other types of minor errors with the corresponding correct entity.

Entity prefix: set to `http://dbpedia.org/resource`. This prefix was used to expand only to resources that are considered entities of our interest.

Maximum number of paths: set to 50, because this value suffices to explore the connectivity between the entities, as reported in Fang et al. (2011), Cheng, Zhang & Qu (2014), Hulpuş, Prangnawarat & Hayes (2015), and Pirrò (2015). Also, this value was used in the experiments performed by Herrera (2017) and this is the exact number of paths for each entity pair in the ground truth proposed by Herrera et al. (2017).

6.1.2
Experiment 1 – Performance Evaluation

This experiment aims to evaluate the performance, in terms of average execution time, of the nine different path search strategies shown in Table 4.1. For each pair of entities in each domain, we searched the top-k relationship paths between them six times (we excluded the first cold start run time, to avoid the warm-up bias) and calculated the average time of the last five executions of the program. Figure 6.1 shows the performance of the path search strategies in both domains.

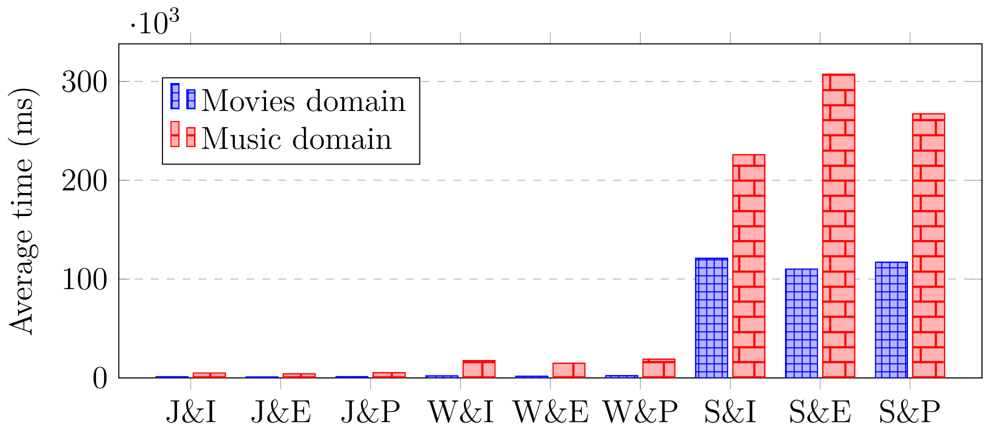


Figure 6.1: Average time of path search strategies using CoEPinKB

The results showed that strategies that use SimRank have a poor performance. They took, on average, 116,070 ms and 266,838 ms to execute the pathfinding process in the movies and music domains, respectively, that is, almost 2 minutes for the movies domain, and almost 4 minutes and a half for the music domain. This is due to the recursive definition of SimRank. As we mentioned in Section 2.2, there are many studies to speed up its computations (LIZORKIN; VELIKHOV, 2008; LI et al., 2010; LI et al., 2020; HAMEDANI; KIM, 2021). However, as these strategies were not shown to be successful for finding relevant relationship paths in the experiments executed by Herrera (2017), we focused on the performance analysis of the rest of the strategies, leaving aside those that use SimRank.

Figure 6.2 shows the performance of the path search strategies in the music and movies domains, splitting the average execution times of each strategy into the average time spent searching and ranking the relationship paths, and excluding the strategies that use SimRank.

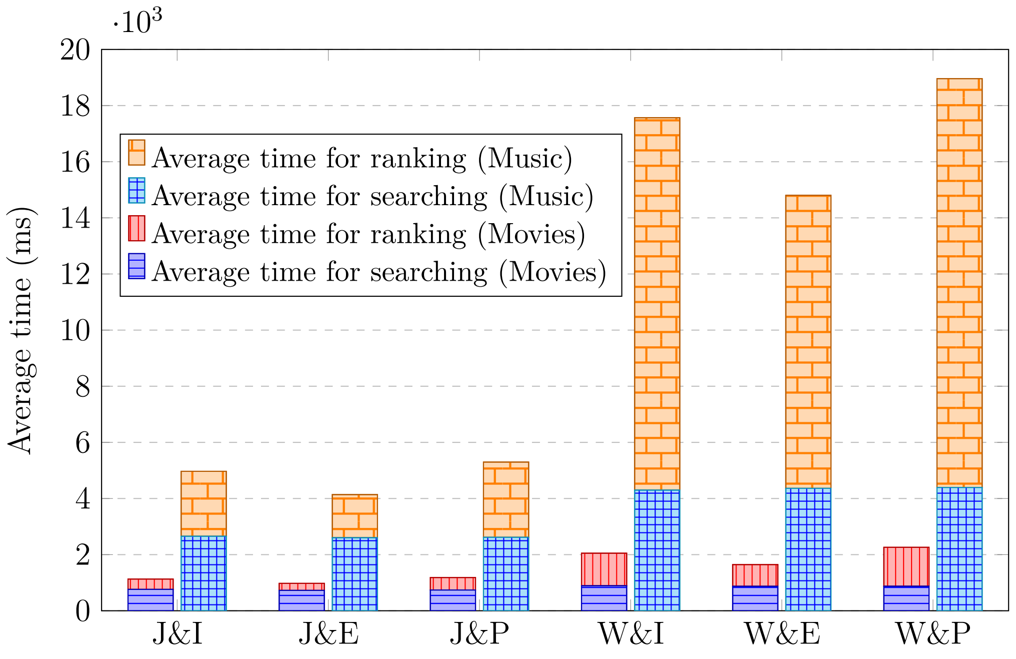


Figure 6.2: Average execution times of path search strategies using CoEPinKB in each domain (excluding S&I, S&E and S&P strategies)

We notice that for the entity pairs in the music domain the average execution time of all strategies is higher than the average execution time for the entity pairs in the movies domain. This is because the entity pairs in the music domain represent "more popular" subjects within DBpedia than those in the movies domain, which means a larger number of links with other entities, which in turn dramatically impacts the performance of graph traversal algorithms such as backward search and the implemented entity similarity and path ranking measures. Overall, in both domains, the best path search strategies in terms of performance are J&E (2,558 ms), J&I (3,049 ms), and J&P (3,241 ms). In the movies domain, these strategies achieved the following average execution times: J&E (976 ms), J&I (1,130 ms), and J&P (1,184 ms). While in the music domain the behavior of the average execution times was: J&E (4,139 ms), J&I (4,968 ms), and J&P (5,298 ms).

The experiments reflect the particularity of how each of the entity similarity and path ranking measures is calculated. The average times for the strategies using the JACCARD INDEX or WLM were quite good and very similar when compared to those using SIMRANK because both entity similarity measures use the feature sets A_d and B_d, which are stored and quickly accessed in our persistent cache. In our experiments, the depth d at which the graph is traversed to acquire features of an entity is set to 2. However, the strategies that use WLM take longer than those that use the JACCARD INDEX because during the process of finding paths they expand the graph through connections that end up generating a larger number of paths in most cases, which affects the performance of the searching and ranking algorithms. On average, the strategies that use the JACCARD INDEX found 132 paths in the movies domain and 920 paths in the music domain, while the strategies that use WLM found 428 and 2,780 paths in the movies and music domain, respectively.

Figure 6.3 shows the number of paths found for each entity pair using the JACCARD INDEX and WLM. Recall that the first 5 entity pairs (EP1-EP5) belong to the music domain, while the rest (EP6-EP10) belong to the movies domain.

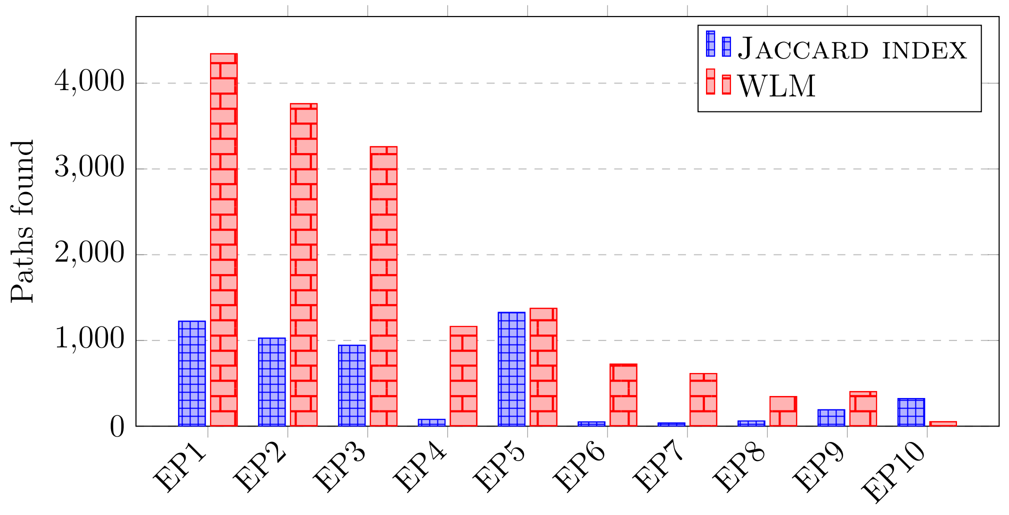


Figure 6.3: Number of paths found for each entity pair using JACCARD INDEX and WLM as entity similarity measures

As for the path ranking measures, EBR executes fewer calculations than PF-ITF and PMI (see Section 2.3). For this reason, the average time for ranking paths using EBR is better than the average time using PF-ITF and PMI, as confirmed in the evaluation of the different strategies.

The experiments in Herrera (2017) indicated that J&E and W&E perform better than the other strategies as far as finding the relevant paths between a pair of entities in the music and movies domains and that the J&E strategy performs better than the baselines. Concerning execution time, the results of this thesis indicated that the most effective strategy is also the fastest one. Therefore, we may conclude that J&E is the fastest strategy and performs better than the other strategies.

6.2 DCoEPinKB Evaluation

We implemented DCoEPinKB in Scala in conjunction with Apache Spark, as mentioned in Section 5.3, and conducted experiments using real-world datasets to evaluate its efficiency and effectiveness. In this section, we discuss the setup and results of our experiments.

6.2.1 Experimental Setup

This section describes the experimental environment, hardware and software components, datasets, and parameter settings for experimenting with the DCoEPinKB framework.

6.2.1.1 Hardware and Software Configurations

All the experiments were performed on a Linux server with Ubuntu 16.04.7 LTS system, an Intel® Core™ i7-5820K CPU @ 3.30GHz, and 16GB of memory dedicated to Spark applications. We used Spark v2.4.3 in the Spark Standalone Mode and Redis v3.0.6. We experimented with a proof-of-concept standalone setup, leaving to future work testing the framework in a fully distributed environment. However, although the experiments were carried out on a single-machine configuration, the methods used for transforming and partitioning the source datasets in multiple Parquet files, which we describe in the next sections, as well as the data structures used for representing data and the subsequent execution of our algorithms for finding relevant paths between entity pairs, are the same regardless of the architecture used.

6.2.1.2
Datasets

We extracted and used two publicly available subsets of the English DBpedia corpus to create the knowledge bases used in the experiments. The first source dataset consists of the cleaned version of high-quality statements with IRI object values extracted by the mappings extraction from Wikipedia Infoboxes[3], and the second dataset consists of data from Wikipedia Infoboxes, as it is, with some smart automatic parsing; this second dataset[4] has better fact coverage than the first one but has less consistency.

Using the Data Preprocessor component available in DCoEPinKB, we transformed these source datasets from the Turtle format to two new datasets in the Parquet format that we called DBpedia21M and DBpedia45M, respectively. DBpedia21M contains the statements in the first source dataset, and DBpedia45M contains the union of the triples in both source datasets. In both cases, we excluded statements involving literals or blank nodes. For each dataset, Table 6.2 shows: the total number of triples, the number of different subjects, properties, and objects; the average out and in node degrees; the size of the datasets source file in Turtle format; and the size of the datasets files after preprocessing and transforming the source files to files in Parquet format.

Table 6.2: Datasets for experimenting with DCoEPinKB

Dataset	**DBpedia21M**	**DBpedia45M**
Total number of triples	21,447,757	45,458,494
Number of subjects	5,422,448	6,143,627
Number of properties	632	13,691
Number of objects	4,605,723	6,035,047
Average outdegree	3.96	7.40
Average indegree	4.66	7.53
Turtle Size	3.1 GB	16.2 GB
Parquet Size	673 MB	1.5 GB

6.2.1.3
Target Entity Pairs

We selected 20 entity pairs from the Entity Relatedness Test Dataset (HERRERA et al., 2017), 10 entity pairs from the music domain, and 10 from the movies domain. We selected the same 10 entity pairs that

[3]https://downloads.dbpedia.org/repo/dbpedia/mappings/mappingbased-objects/2021.03.01/mappingbased-objects_lang=en.ttl.bz2

[4]https://downloads.dbpedia.org/repo/dbpedia/generic/infobox-properties/2021.03.01/infobox-properties_lang=en.ttl.bz2

were used in the experimentation with CoEPinKB, and 10 more entity pairs. Table 6.3 shows the selected entity pairs and the degree of each entity in the datasets used for experimentation. Observe that the entities from the music domain have a higher degree than the entities from the movies domain, which affects the performance of the path search strategies, as discussed in Experiment 2 reported in Section 6.2.2.

Table 6.3: Entity pairs selected for experimenting with DCoEPinKB

Music domain				Movies domain			
EP	Entity	Degree in DBpedia21M	Degree in DBpedia45M	EP	Entity	Degree in DBpedia21M	Degree in DBpedia45M
1	dbr:Michael_Jackson	442	857	11	dbr:Elizabeth_Taylor	83	150
	dbr:Whitney_Houston	189	362		dbr:Richard_Burton	79	139
2	dbr:The_Beatles	441	980	12	dbr:Cary_Grant	83	153
	dbr:The_Rolling_Stones	353	769		dbr:Katharine_Hepburn	70	126
3	dbr:Elton_John	415	945	13	dbr:Laurence_Olivier	96	170
	dbr:George_Michael	192	402		dbr:Ralph_Richardson	55	107
4	dbr:Led_Zeppelin	135	316	14	dbr:Errol_Flynn	83	149
	dbr:The_Who	277	550		dbr:Olivia_de_Havilland	69	109
5	dbr:Pink_Floyd	303	560	15	dbr:William_Powell	96	174
	dbr:David_Gilmour	187	303		dbr:Myrna_Loy	105	189
6	dbr:U2	314	595	16	dbr:James_Stewart	103	190
	dbr:R.E.M.	250	450		dbr:Henry_Fonda	122	220
7	dbr:Metallica	188	353	17	dbr:Paul_Newman	99	175
	dbr:Anthrax	129	219		dbr:Joanne_Woodward	48	89
8	dbr:Rihanna	224	446	18	dbr:Bette_Davis	110	207
	dbr:Nicki_Minaj	261	519		dbr:Joan_Crawford	103	197
9	dbr:Velvet_Revolver	84	117	19	dbr:John_Wayne	181	295
	dbr:Guns_N'_Roses	259	392		dbr:Kirk_Douglas	104	190
10	dbr:Bob_Dylan	649	1663	20	dbr:Charlie_Chaplin	184	395
	dbr:The_Band	124	245		dbr:Frank_D._Williams	57	109
	Average	**271**	**552**		**Average**	**97**	**177**
	Max	**649**	**1663**		**Max**	**184**	**395**
	Min	**84**	**117**		**Min**	**48**	**89**
	Standard Deviation	**136,97**	**353,37**		**Standard Deviation**	**35,39**	**70,02**

6.2.1.4 Data Storage and Partitioning

We used the RDF relational schema known as Statement Table to logically represent the datasets. This schema directly maps RDF data onto a table with three columns (subject, predicate, object), in which each tuple corresponds to an RDF statement. Other well-known schemes, such as Property Table and Vertical Partitioning, are not suitable due to the type of queries that our framework frequently performs, mainly one-step queries at a time (from subject to object, or vice versa), and the particularities of the graphs used, which have a large number of properties, for example. Regardless of this, one of the suggested future directions is to experiment with other partitioning strategies. Another aspect to evaluate in the future is the possibility of additionally representing the inverse triples, given how the graph is expanded and that the paths are considered as undirected.

For data partitioning, we used the horizontal-based partitioning technique, which evenly partitions the data horizontally over the number of machines in the cluster. For our proof-of-concept, we partitioned the two statement tables representing the data in the two datasets according to the number of CPU cores on the machine. Both datasets were partitioned into 200 Parquet files each, each file representing a partition. The files corresponding to DBpedia21M have an average size of 3.5 MB, while the files corresponding to DBpedia45M have an average size of 7.5 MB. As future work, we plan to test the framework on a real distributed environment, with multiple machines in a cluster, and to conduct experiments to evaluate the performance using different partitioning techniques and a different number of partitions.

6.2.1.5 Path Search Strategies

Using DCoEPinKB, we proceeded to evaluate a family of six path search strategies obtained by combining two entity similarity measures (Jaccard index and WLM) and three path ranking measures (PF-ITF, EBR, and PMI), which correspond to the first six strategies presented in Table 4.1. Table 6.4 shows the path search strategies evaluated using DCoEPinKB.

Table 6.4: Path Search Strategies evaluated using DCoEPinKB

#	**Acronym**	**Name**
1	J&I	Jaccard index & PF-ITF
2	J&E	Jaccard index & EBR
3	J&P	Jaccard index & PMI
4	W&I	WLM & PF-ITF
5	W&E	WLM & EBR
6	W&P	WLM & PMI

6.2.1.6 Expansion limits

We evaluate the six path search strategies in Table 6.4 for different expansion limits during the pathfinding process. So, for the experiments, we successively set the expansion limit to $\lambda = 5, 10, 15, 20, 25$, that is, to the top 5,..., 25 adjacent nodes, ranked by the entity similarity measure, and also to the top 50% of the adjacent nodes, ranked by the entity similarity measure.

6.2.1.7 Configuration parameters

The rest of the configuration parameters (i.e., the maximum path length between the entities, the maximum entity degree, the set of ignored properties, the entity prefix, and the maximum number of paths) were set as in the experiments with CoEPinKB (Section 6.1.1.5).

6.2.1.8
Ground Truth

We needed a ground truth to evaluate the ranking accuracy of the path search strategies for different expansion limit values using $nDCG_k$, for $k = 1$ to 50. We did not adopt the ranked lists of paths in Herrera et al. (2017) as our ground truth because the subsets of DBpedia we used were different from those in Herrera et al. (2017) –DBpedia indeed constantly changes. This section describes how we constructed the ground truth.

Let π_i, for $i = 1...6$, be one of the six path search strategies listed in Table 6.4. For each entity pair in each DBpedia dataset (i.e., DBpedia21M and DBpedia45M), we created a separate ground truth path ranking for π_i by:

1. Executing π_i with different expansion limits ($\lambda = 5, 10, 15, 20, 25, 50\%$). It resulted in six sets of paths $P_{\pi_i}^{\lambda_j}$, one for each expansion limit λ_j.
2. Combining all sets of paths obtained in one set of paths $P_{\pi_i} = \bigcup_{j=1}^{6} P_{\pi_i}^{\lambda_j}$.
3. Ranking the paths in P_{π_i} using the path ranking measure adopted in π_i, and retaining the top-50 ranked paths.

This resulted in a dataset[5] that contains a total of 240 ranked lists, with 50 relationship paths each, between entity pairs in the music and movies domains. For each pair of entities there are 12 ranked lists, 6 for each of the 2 DBpedia datasets (i.e., DBpedia21M and DBpedia45M) and each one specific to one of the 6 path search strategies (i.e., J&I, J&E, J&P, W&I, W&E, W&P). This dataset supports the evaluation of approaches that address the entity relatedness problem and specifically permits evaluating the impact of the expansion limit, for a given entity similarity measure and a path ranking measure.

6.2.1.9
Ranking Accuracy

We adopted the Normalized Discounted Cumulative Gain (nDCG) to measure the accuracy of the rankings obtained. This measure was previously introduced in Section 2.4.

[5]The dataset containing the ground truth files is available at https://figshare.com/articles/dataset/Ground_Truth_for_Entity_Relatedness_Problem_over_DBpedia_datasets/15181086.

6.2.2
Experiment 2 – Performance Evaluation

Using DCoEPinKB, the first set of experiments evaluated the performance, in terms of average execution time, of different path search strategies for increasing values of the expansion limit. First, we analyze the results using the J&E strategy, as this strategy achieved the best performance for finding relevant relationship paths in Herrera (2017) and the best average execution times as demonstrated in Experiment 1 (Section 6.1.2). Figure 6.4 shows the average execution times for this strategy. For each pair of entities, we searched 6 times the top-50 paths between them, excluded the first run time to avoid the warm-up bias, and calculated the average time of the last 5 executions.

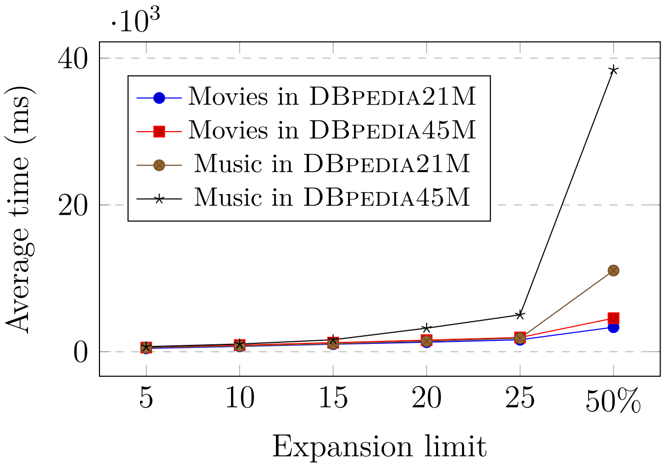


Figure 6.4: Average execution time over all entity pairs in each domain and dataset for the J&E strategy varying the expansion limit

Clearly, the execution time increases with higher expansion limits. For the entity pairs from the movies domain, DCoEPinKB kept the time for finding relevant paths, on average, below 2.0 secs, when the expansion limit was set to 25, or below. When the expansion limit was the top 50% of most similar adjacent nodes, the algorithm took, on average, around 3.3 secs for DBpedia21M and 4.6 secs for DBpedia45M. For the entity pairs from the music domain, the time for finding relevant paths remained, on average, below 2.0 secs for DBpedia21M and 5.0 secs for DBpedia45M when the expansion limit was set to 25, or below. When the expansion limit was the top 50% of most similar adjacent nodes, the algorithm took, on average, around 11.0 secs for DBpedia21M and 38.4 secs for DBpedia45M, which means quite a noticeable increase. The average execution times of the other five strategies behaved similarly to the average execution times of the J&E strategy when increasing the expansion limits, as shown in Figure A.1 in Appendix A.

As the execution time is highly dependent on the number of paths found, we also show in Figure 6.5 the average number of paths found for different expansion limits using the J&E strategy. The number of paths found is closely related to the degree of the entities involved. Hence, by expanding the 50% most similar adjacent nodes, in the case of entities with a high degree, the framework will carry out a broader exploration of the graph and increase the probability of finding many more paths to be ranked, which implies that the running time also increases.

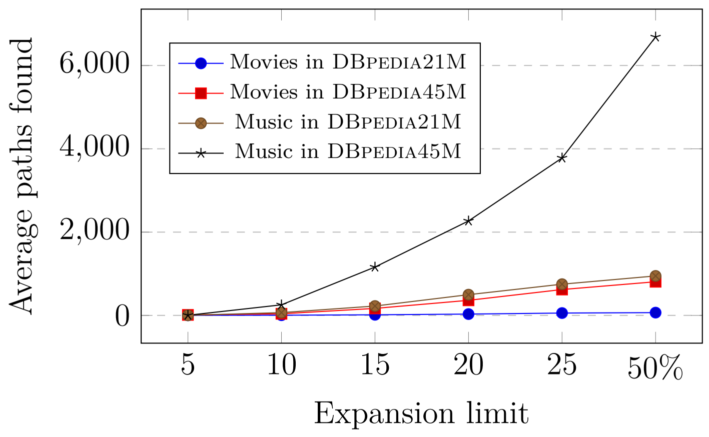


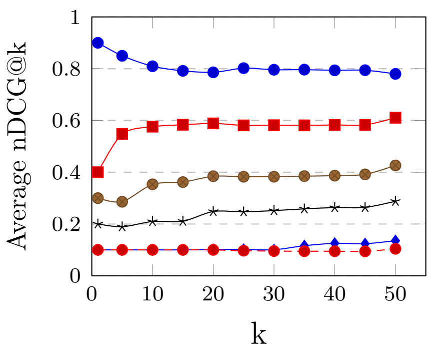
Figure 6.5: Number of paths found over all entity pairs in each domain and dataset for the J&E strategy varying the expansion limit

6.2.3 Experiment 3 – Ranking Accuracy

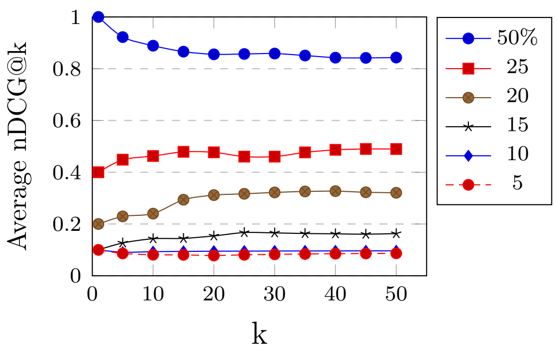
The second set of experiments with DCoEPinKB evaluated the ranking accuracy of the path search strategies, for different expansion limit values, as compared to the proposed ground truth, using $nDCG_k$, for $k = 1$ to 50. In what follows, let "S with top-n" indicates the strategy S expanding the top n most similar adjacent nodes, where n may also be a percentage.

Figure 6.6 shows the average $nDCG_k$ for the J&E strategy for the movies and music domains in DBpedia21M and DBpedia45M. For the movies domain, J&E with top-50% obtained a good performance in both datasets: the average $nDCG_k$ was above 0.80 using DBpedia21M (Figure 6.6a), and above 0.86 using DBpedia45M (Figure 6.6b), without a significant loss for higher values of k. J&E with top-50% also had a good performance for the music domain. In this case, the average $nDCG_k$ was above 0.73 using DBpedia21M (Figure 6.6c), and above 0.84 using DBpedia45M (Figure 6.6d). Finally, note that, although J&E with top-50% had a high average execution time over DBpedia45M (Figure 6.4), the difference between the average $nDCG_k$

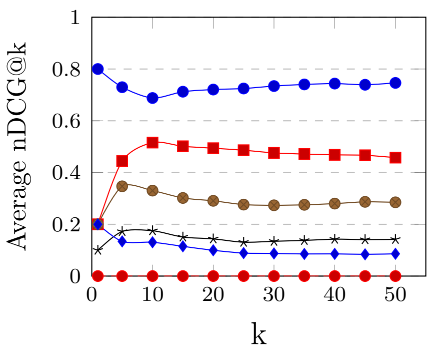
for J&E with top-50% and J&E with top-25 does not justify saving time in detriment of finding the most relevant paths. The smallest difference in the ranking accuracy between both expansion strategies occurs between positions 2 and 8 of the ranking, where the top-50% strategy reaches an average $nDCG_k$ equal to 0.79, while the top-25 strategy reaches a low 0.44.



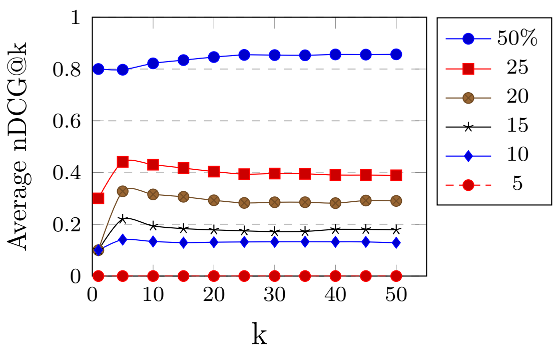
(a) Movies domain in DBpedia21M



(b) Movies domain in DBpedia45M



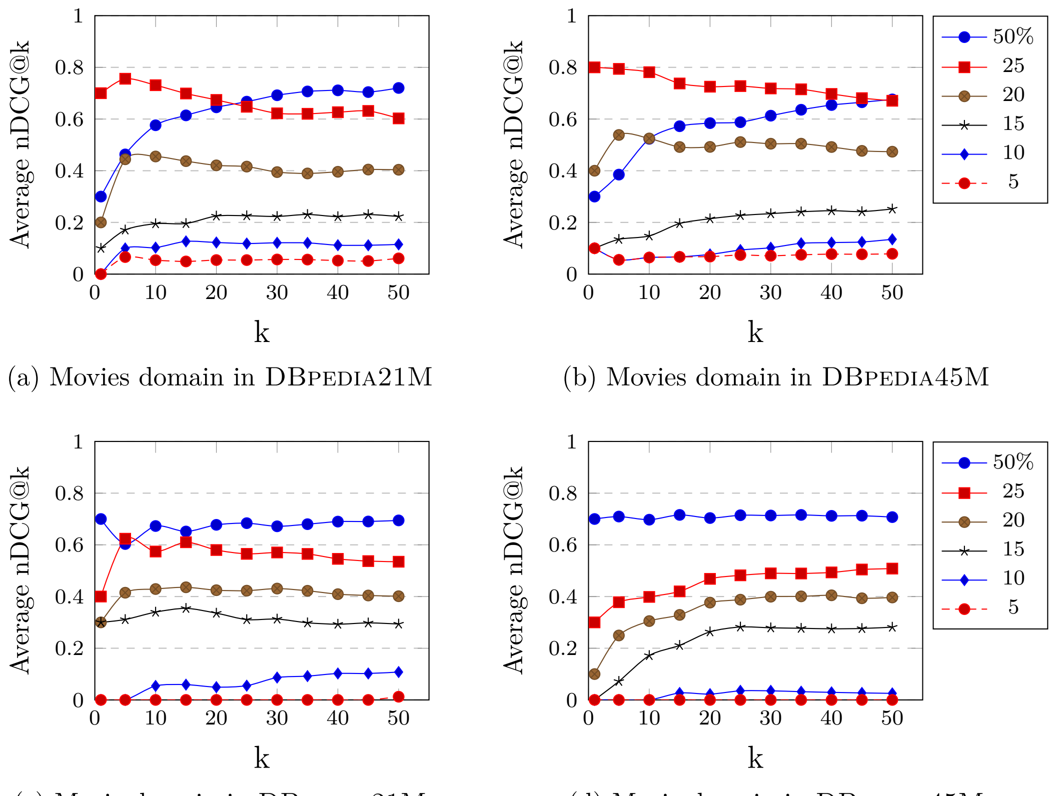
(c) Music domain in DBpedia21M



(d) Music domain in DBpedia45M

Figure 6.6: Average nDCG@k over the movies and music domains for the J&E strategy varying the expansion limit

Figure 6.7 shows the average $nDCG_k$ for the J&I strategy for the movies and music domains in DBpedia21M and DBpedia45M. We argue that, by using the PF-ITF measure for ranking the relationship paths, it is possible to achieve acceptable performance in the movies domain using J&I with the top-25. Indeed, the average $nDCG_k$ for J&I with top-25 was above 0.67 using DBpedia21M (Figure 6.7a), and above 0.73 using DBpedia45M (Figure 6.7b). Figure 6.7a shows that the average $nDCG_k$ for J&I with top-25 decreases for higher values of k. But, for $k \leq 20$, it had an average $nDCG_k$ of 0.72, which is better than the average $nDCG_k$ of 0.52 for J&I with top-50%. Figure 6.7b also shows that the average $nDCG_k$ for J&I with top-25 is better than the average $nDCG_k$ for J&I with top-50%. For the music domain and DBpedia21M, and for $3 \leq k \leq 15$, J&I with top-25 and J&I with top-50% both had an average $nDCG_k$ close to 0.62 (Figure 6.7c), which justifies using, in this case, a slightly less expensive expansion strategy. Figure A.1a shows the average execution times of the J&I strategy. Given the high average execution time of J&I with top-50%, it is worth opting for J&I with top-25, especially if one wants the first few most relevant paths.



(a) Movies domain in DBpedia21M (b) Movies domain in DBpedia45M

(c) Music domain in DBpedia21M (d) Music domain in DBpedia45M

Figure 6.7: Average nDCG@k over the movies and music domains for the J&I strategy varying the expansion limit

Similar to the J&I with the top-25, in the movies domain, W&I with the top-25 achieves a very good performance that outperforms the rest of the expansion strategies. Indeed, the average $nDCG_k$ for W&I with top-25 was above 0.78 using DBpedia21M (Figure A.3a), and above 0.77 using DBpedia45M (Figure A.3a). Another interesting result is that by using the PMI measure to rank the relationship paths, it is possible to achieve acceptable performance in the movies domain in DBpedia21M using J&P or W&P with the top-25. Indeed, using DBpedia21M, the average $nDCG_k$ for J&P with top-25 was above 0.67 (Figure A.2a), and above 0.69 for W&P with top-25 (Figure A.5a).

To summarize, in most cases, the top-50% most similar adjacent nodes strategy had the highest average execution times and achieved the best accuracy rankings. However, in some cases, it is feasible to use a strategy

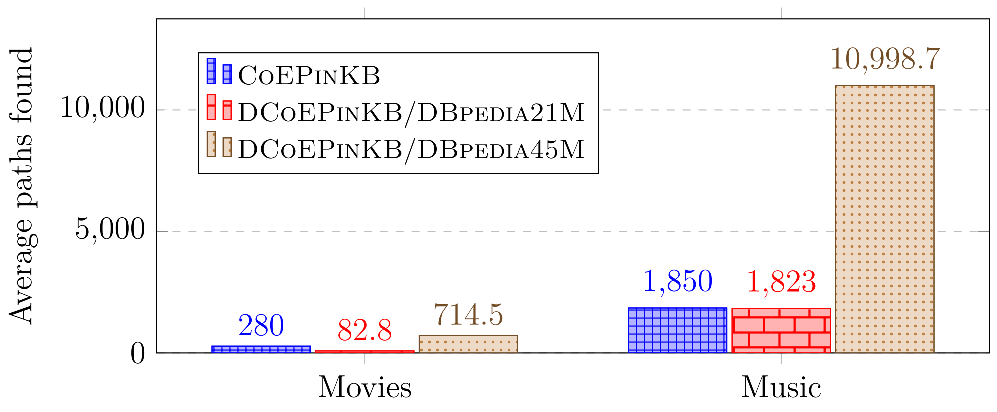


Figure 6.8: Average paths found for each domain using CoEPinKB and DCoEPinKB

such as the top-25 most similar adjacent nodes, which is faster and achieves acceptable results.

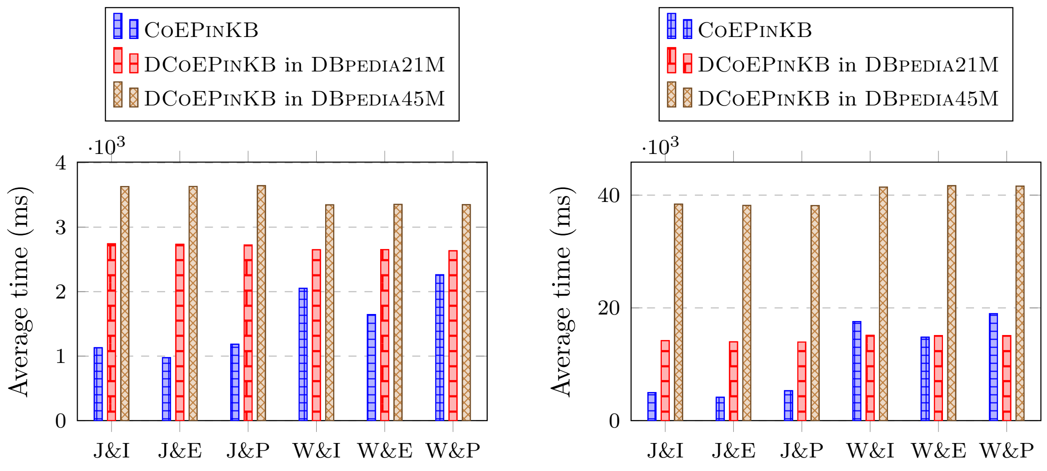
6.3 Comparison of CoEPinKB and DCoEPinKB

In this section, we compare the performance of the CoEPinKB and DCoEPinKB frameworks. Although the experiments with both frameworks were performed on different DBpedia datasets, it is possible to draw some conclusions from the results of these experiments.

Figure 6.8 shows the average paths found for the movies and music domains using CoEPinKB and DCoEPinKB using the DBpedia21M and DBpedia45M datasets. DCoEPinKB was tested with different expansion limits. However, for the comparison to make sense, we calculated the average number of paths found using the expansion limit equal to 50%. Similarly, for this calculation, we only considered those 10 entity pairs that were also used in the experiments with CoEPinKB. The experiments revealed that the number of paths found using CoEPinKB, as well as DCoEPinKB with the DBpedia21M dataset, are quite similar. However, in the DBpedia45M dataset, the number of paths found using DCoEPinKB is much higher, especially in the music domain, where the number of paths found is almost 6 times higher than the number of paths found using CoEPinKB or DCoEPinKB in the DBpedia21M dataset.

This significant difference between the number of paths found affects the execution times of each of the strategies. Figure 6.9 shows the average execution times of the six path search strategies using CoEPinKB and DCoEPinKB in the movies and music domains.

As expected, the average execution times of DCoEPinKB were the



(a) Movies domain (b) Music domain

Figure 6.9: Average time of path search strategies using CoEPinKB and DCoEPinKB in the movies and music domains

highest in both domains in DBpedia45M. Furthermore, the average execution times of DCoEPinKB in DBpedia21M were higher than those of CoEPinKB in 10 of the 12 combinations of path search strategies and domains. DCoEPinKB only beats CoEPinKB using the W&I and W&P strategies in the music domain, over DBpedia21M. But one has to remember that our proof-of-concept implementation of DCoEPinKB was tested on a single-machine configuration and, therefore, had to allocate its resources both for data processing as well as for data storage in the HDD and main memory. This limitation partly justifies the results reported in Figure 6.9, but calls for further experiments to assess the performance of DCoEPinKB in a truly distributed experiment.

6.4 Chapter Conclusions

In this chapter, we presented a set of experiments to evaluate a family of path search strategies using the CoEPinKB and DCoEPinKB frameworks over real-world datasets extracted from DBpedia. We evaluated the performance, in terms of average execution time, of the two frameworks, and the ranking accuracy of the path search strategies for different expansion limit values in DCoEPinKB. We also described how we created a ground truth to experiment with DCoEPinKB.

Our performance evaluation of the path search strategies using CoEPinKB indicated that any strategy that uses SimRank as the activation function has a poor performance when compared with the other strategies. We also verified that J&E, which is the most effective strategy, is also the fastest one. The experiments with DCoEPinKB showed that reducing the expansion limit, when finding the paths between entities with a high degree, can improve the execution time, as expected, but without a significant loss in the accuracy of the ranking when only the first few (i.e., 10 or less) top paths are requested.

In a broader sense, this evaluation contributed to understanding the interplay between entity similarity and path ranking measures. Indeed, the literature provides an extensive list of such measures, which are often addressed separately or, at best, a specific pair is investigated. However, in the context of the entity relatedness problem, they have a combined effect to select the paths that best explain the relationships between two entities.

7 Conclusions and Future Work

7.1 Conclusions

The entity relatedness problem refers to the question of exploring a knowledge base, represented as an RDF graph, to discover and understand how two entities are connected. Strategies designed to address this problem typically adopt an entity similarity measure to reduce the path search space and a path ranking measure to order and filter the list of relevant paths returned.

The main contribution of this thesis, presented in Chapters 4 and 5, is the proposal and implementation of approaches that help address the entity relatedness problem. We introduced two frameworks, called CoEPinKB and DCoEPinKB, to empirically evaluate path search strategies that combine entity similarity and path ranking measures. Both frameworks have some hot spots for developers to easily add new entity similarity and relationship path ranking measures to generate new path search strategies, as well as to work with different knowledge bases.

The second contribution, presented in Section 6.2.1.8, is the proposal of a ground truth that supports the evaluation of approaches that address the entity relatedness problem and specifically permits evaluating the impact of the expansion limit, for a given entity similarity measure and a path ranking measure. This dataset contains a total of 240 ranked lists, with 50 relationship paths each, between entity pairs in the music and movies domains.

Finally, another important contribution, presented in Chapter 6, is an extensive experimental evaluation on the music and movies domains over real-data, collected from DBpedia, to assess the correctness and the performance of a family of path search strategies using the CoEPinKB and DCoEPinKB frameworks. The performance evaluation indicated that strategies that use SimRank as activation function have a poor performance when compared with the other strategies. We also verified that J&E, which is the most effective strategy, is also the fastest one. The experiments with DCoEPinKB showed that reducing the expansion limit, when finding the paths between entities with a high degree, can improve the execution time, as expected, but without a significant loss in the accuracy of the ranking when only the first few (i.e., 10 or less) top paths are requested.

The work on the CoEPinKB framework was published in Jiménez, Leme & Casanova (2021). The work on the DCoEPinKB framework was presented at the Brazilian Symposium on Databases (SBBD 2021) (JIMéNEZ et al., 2021) and will be published soon.

7.2 Future Work

As future work, we plan to deploy the DCoEPinKB framework in a fully distributed environment and repeat the experiments in this new configuration setting. We also want to improve DCoEPinKB to run in interactive mode with a graphical user interface that facilitates the use by the end-users. A significant improvement could be in the visualization of the relevant relationship paths, not as a sequence of identifiers of entities and properties, but as a graph that the user can visually explore.

Additionally, we plan to evaluate the performance of DCoEPinKB using different relational schemas for RDF other than the Statement Table schema, such as the Property Table and Vertical Partitioning schemas, various RDF-based partitioning techniques (i.e., subject-based, predicate-based, and horizontal-based partitioning), and data formats other than Parquet. We also plan to evaluate the complexity of the algorithms using different configurations.

We plan to implement additional entity similarity metrics (including SimRank in the distributed framework) and relationship path ranking measures and test the path search strategies over other domains and knowledge bases. We also intend to evaluate the benefits of incorporating the InfoRank measure (MENENDEZ, 2019) into the framework. Using InfoRank, we can assign an importance value to each resource and, in this way, we will be able to select the most important similar entities during the expansion process of the backward search algorithm to find the most relevant paths. This will make it possible to have new search strategies since, for each similarity measure, it will now be possible to take into account the importance of the neighbor of a node to decide whether that neighbor should be expanded or not.

The proposed frameworks work with RDF datasets, but, in the future, a solution that allows exploring the connectivity of entities in relational databases could be implemented. In that case, the relational data could be transformed in the following way: the tuples within a table would be considered as the entities, identified by the primary key, and the foreign keys would represent the properties or relationships with other entities.

8
Bibliography

ABADI, D. J. et al. SW-Store: a vertically partitioned DBMS for Semantic Web data management. **The VLDB Journal**, v. 18, n. 2, p. 385–406, abr. 2009. ISSN 0949-877X. Available from Internet: <https://doi.org/10.1007/s00778-008-0125-y>. Cited in page 24.

ABDELAZIZ, I. et al. A survey and experimental comparison of distributed SPARQL engines for very large RDF data. **Proceedings of the VLDB Endowment**, v. 10, n. 13, p. 2049–2060, set. 2017. ISSN 2150-8097. Available from Internet: <https://dl.acm.org/doi/10.14778/3151106.3151109>. Cited 5 times in pages 18, 23, 24, 36, and 37.

BERNERS-LEE, T. **Linked Data - Design Issues**. 2006. Available from Internet: <https://www.w3.org/DesignIssues/LinkedData.html>. Cited in page 22.

BHALOTIA, G. et al. Keyword searching and browsing in databases using BANKS. In: **Proceedings 18th International Conference on Data Engineering**. [S.l.: s.n.], 2002. p. 431–440. ISSN: 1063-6382. Cited in page 32.

CHENG, G.; LIU, D.; QU, Y. Fast Algorithms for Semantic Association Search and Pattern Mining. **IEEE Transactions on Knowledge and Data Engineering**, v. 33, n. 4, p. 1490–1502, abr. 2021. ISSN 1558-2191. Conference Name: IEEE Transactions on Knowledge and Data Engineering. Cited in page 33.

CHENG, G.; SHAO, F.; QU, Y. An Empirical Evaluation of Techniques for Ranking Semantic Associations. **IEEE Transactions on Knowledge and Data Engineering**, v. 29, n. 11, p. 2388–2401, nov. 2017. ISSN 1558-2191. Conference Name: IEEE Transactions on Knowledge and Data Engineering. Cited in page 33.

CHENG, G.; ZHANG, Y.; QU, Y. Explass: Exploring Associations between Entities via Top-K Ontological Patterns and Facets. In: MIKA, P. et al. (Ed.). **The Semantic Web – ISWC 2014**. Cham: Springer International Publishing, 2014. v. 8797, p. 422–437. ISBN 978-3-319-11914-4 978-3-319-11915-1. Series Title: Lecture Notes in Computer Science. Available from Internet: <http://link.springer.com/10.1007/978-3-319-11915-1_27>. Cited 8 times in pages 17, 18, 32, 33, 34, 41, 65, and 66.

CHURCH, K. W.; HANKS, P. Word Association Norms, Mutual Information, and Lexicography. **Computational Linguistics**, v. 16, n. 1, p. 22–29, 1990. Available from Internet: <https://www.aclweb.org/anthology/J90-1003>. Cited 2 times in pages 28 and 29.

COHEN, W. W. **Graph walks and graphical models**. [S.l.]: Citeseer, 2010. v. 5. Cited in page 65.

CYGANIAK, R.; WOOD, D.; LANTHALER, M. **RDF 1.1 Concepts and Abstract Syntax**. 2014. W3C Recommendation 25 February 2014. Available from Internet: <https://www.w3.org/TR/rdf11-concepts/>. Cited in page 21.

DEAN, J.; GHEMAWAT, S. MapReduce: simplified data processing on large clusters. **Communications of the ACM**, v. 51, n. 1, p. 107–113, jan. 2008. ISSN 0001-0782, 1557-7317. Available from Internet: <https://dl.acm.org/doi/10.1145/1327452.1327492>. Cited in page 36.

FANG, L. et al. REX: explaining relationships between entity pairs. **Proceedings of the VLDB Endowment**, v. 5, n. 3, p. 241–252, nov. 2011. ISSN 2150-8097. Available from Internet: <http://dl.acm.org/doi/10.14778/2078331.2078339>. Cited 8 times in pages 17, 18, 32, 34, 40, 41, 65, and 66.

FAYE, D. C.; CURé, O.; BLIN, G. A survey of RDF storage approaches. **Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées**, Volume 15, p. 11–35, set. 2012. ISSN 1638-5713. Available from Internet: <https://arima.episciences.org/1956>. Cited 2 times in pages 23 and 24.

HAMEDANI, M. R.; KIM, S.-W. On Investigating Both Effectiveness and Efficiency of Embedding Methods in Task of Similarity Computation of Nodes in Graphs. **Applied Sciences**, v. 11, n. 1, p. 162, jan. 2021. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute. Available from Internet: <https://www.mdpi.com/2076-3417/11/1/162>. Cited 2 times in pages 27 and 67.

HAYES, P. J.; PATEL-SCHNEIDER, P. F. **RDF 1.1 Semantics**. 2014. Available from Internet: <https://www.w3.org/TR/2014/REC-rdf11-mt-20140225/>. Cited in page 21.

HEIM, P. et al. RelFinder: Revealing Relationships in RDF Knowledge Bases. In: HUTCHISON, D. et al. (Ed.). **Semantic Multimedia**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. v. 5887, p. 182–187. ISBN 978-3-642-10542-5 978-3-642-10543-2. Series Title: Lecture Notes in Computer Science. Available from Internet: <http://link.springer.com/10.1007/978-3-642-10543-2_21>. Cited 2 times in pages 17 and 32.

HERRERA, J. E. T. **On the Connectivity of Entity Pairs in Knowledge Bases**. Tese (Doctoral Dissertation) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil, maio 2017. Available from Internet: <http://www.maxwell.vrac.puc-rio.br/Busca_etds.php?strSecao=resultado&nrSeq=30742@2>. Cited 15 times in pages 17, 19, 32, 35, 39, 43, 44, 45, 46, 63, 65, 66, 67, 69, and 74.

HERRERA, J. E. T. et al. DBpedia Profiler Tool: Profiling the Connectivity of Entity Pairs in DBpedia. In: **Proceedings of the 5th International Workshop on Intelligent Exploration of Semantic Data (IESD 2016)**. [S.l.: s.n.], 2016. Cited 6 times in pages 17, 18, 32, 34, 41, and 65.

HERRERA, J. E. T. et al. An Entity Relatedness Test Dataset. In: D'AMATO, C. et al. (Ed.). **The Semantic Web – ISWC 2017**. Cham: Springer International Publishing, 2017. v. 10588, p. 193–201. ISBN 978-3-319-68203-7 978-3-319-68204-4. Series Title: Lecture Notes in Computer Science. Available from Internet: <http://link.springer.com/10.1007/978-3-319-68204-4_20>. Cited 8 times in pages 18, 34, 35, 63, 64, 66, 70, and 73.

HUANG, J.; ABADI, D. J.; REN, K. Scalable SPARQL querying of large RDF graphs. **Proceedings of the VLDB Endowment**, v. 4, n. 11, p. 1123–1134, ago. 2011. ISSN 2150-8097. Available from Internet: <https://dl.acm.org/doi/10.14778/3402707.3402747>. Cited 2 times in pages 18 and 36.

HULPUș, I.; PRANGNAWARAT, N.; HAYES, C. Path-Based Semantic Relatedness on Linked Data and Its Use to Word and Entity Disambiguation. In: ARENAS, M. et al. (Ed.). **The Semantic Web - ISWC 2015**. Cham: Springer International Publishing, 2015. v. 9366, p. 442–457. ISBN 978-3-319-25006-9 978-3-319-25007-6. Series Title: Lecture Notes in Computer Science. Available from Internet: <http://link.springer.com/10.1007/978-3-319-25007-6_26>. Cited 3 times in pages 28, 29, and 66.

HUSAIN, M. et al. Heuristics-Based Query Processing for Large RDF Graphs Using Cloud Computing. **IEEE Transactions on Knowledge and Data Engineering**, v. 23, n. 9, p. 1312–1327, set. 2011. ISSN 1041-4347. Available from Internet: <http://ieeexplore.ieee.org/document/5765957/>. Cited 2 times in pages 18 and 36.

HUSAIN, M. F. et al. Storage and Retrieval of Large RDF Graph Using Hadoop and MapReduce. In: HUTCHISON, D. et al. (Ed.). **Cloud Computing**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. v. 5931, p. 680–686. ISBN 978-3-642-10664-4 978-3-642-10665-1. Series Title: Lecture Notes in Computer Science. Available from Internet: <http://link.springer.com/10.1007/978-3-642-10665-1_72>. Cited in page 36.

JACCARD, P. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. **Bull Soc Vaudoise Sci Nat**, v. 37, p. 547–579, 1901. Cited 2 times in pages 25 and 26.

JEH, G.; WIDOM, J. SimRank: a measure of structural-context similarity. In: **Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining**. ACM, 2002. p. 538–543. Available from Internet: <http://dl.acm.org/citation.cfm?id=775126>. Cited 2 times in pages 25 and 27.

JIMéNEZ, J. G.; LEME, L. A. P. P.; CASANOVA, M. A. CoEPinKB: A Framework to Understand the Connectivity of Entity Pairs in Knowledge Bases. In: **Proceedings of the Integrated Software and Hardware Seminar (SEMISH)**. SBC, 2021. p. 97–105. ISSN: 2595-6205. Available from Internet: <https://sol.sbc.org.br/index.php/semish/article/view/15811>. Cited 3 times in pages 19, 38, and 82.

JIMéNEZ, J. G. et al. A distributed framework to investigate the entity relatedness problem in large RDF knowledge bases. Forthcoming. 2021. Cited 3 times in pages 19, 38, and 82.

JäRVELIN, K.; KEKäLäINEN, J. Cumulated gain-based evaluation of IR techniques. **ACM Transactions on Information Systems (TOIS)**, v. 20, n. 4, p. 422–446, out. 2002. ISSN 1046-8188, 1558-2868. Available from Internet: <http://dl.acm.org/doi/10.1145/582415.582418>. Cited in page 30.

KACHOLIA, V. et al. Bidirectional expansion for keyword search on graph databases. In: **Proceedings of the 31st international conference on Very**

large data bases. Trondheim, Norway: VLDB Endowment, 2005. (VLDB '05), p. 505–516. ISBN 978-1-59593-154-2. Cited in page 33.

KIM, T. et al. Similarity query support in big data management systems. **Information Systems**, v. 88, p. 101455, fev. 2020. ISSN 03064379. Available from Internet: <https://linkinghub.elsevier.com/retrieve/pii/S0306437919305071>. Cited in page 36.

LE, W. et al. Scalable keyword search on large RDF data. **Knowledge and Data Engineering, IEEE Transactions on**, v. 26, n. 11, p. 2774–2788, 2014. Available from Internet: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6720109>. Cited 3 times in pages 18, 35, and 40.

LEHMANN, J. et al. DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia. **Semantic Web**, v. 6, n. 2, p. 167–195, 2015. ISSN 15700844. Available from Internet: <https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/SW-140134>. Cited 2 times in pages 17 and 64.

LEHMANN, J.; SCHüPPEL, J.; AUER, S. Discovering Unknown Connections – the DBpedia Relationship Finder. In: **The Social Semantic Web 2007–Proceedings of the 1st Conference on Social Semantic Web (CSSW)**. Gesellschaft für Informatik e. V., 2007. p. 99–109. ISBN 978-3-88579-207-9. Accepted: 2019-05-15T08:44:12Z ISSN: 1617-5468. Available from Internet: <http://dl.gi.de/handle/20.500.12116/22392>. Cited in page 32.

LESKOVEC, J.; RAJARAMAN, A.; ULLMAN, J. D. **Mining of Massive Datasets**. [S.l.]: Cambridge University Press, 2014. Cited in page 26.

LI, C. et al. Fast computation of SimRank for static and dynamic information networks. In: **Proceedings of the 13th International Conference on Extending Database Technology - EDBT '10**. Lausanne, Switzerland: ACM Press, 2010. p. 465–476. ISBN 978-1-60558-945-9. Available from Internet: <http://portal.acm.org/citation.cfm?doid=1739041.1739098>. Cited 2 times in pages 27 and 67.

LI, M. et al. CrashSim: An Efficient Algorithm for Computing SimRank over Static and Temporal Graphs. In: **Proceedings of the IEEE 36th International Conference on Data Engineering (ICDE)**. [S.l.]: IEEE, 2020. p. 1141–1152. Cited 2 times in pages 27 and 67.

LIZORKIN, D.; VELIKHOV, P. Accuracy Estimate and Optimization Techniques for SimRank Computation. **Proceedings of the VLDB Endowment**, v. 1, n. 1, p. 422–433, ago. 2008. Cited 2 times in pages 27 and 67.

MAHRIA, B. B.; CHAKER, I.; ZAHI, A. An empirical study on the evaluation of the RDF storage systems. **Journal of Big Data**, v. 8, n. 100, p. 1–20, jul. 2021. ISSN 2196-1115. Available from Internet: <https://doi.org/10.1186/s40537-021-00486-y>. Cited in page 23.

MARKIEWICZ, M. E.; LUCENA, C. J. P. Object oriented framework development. **Crossroads**, p. 10–1145, 2001. Cited in page 51.

MENENDEZ, E. S. **Novel Node Importance Measures to Improve Keyword Search over RDF Graphs**. Tese (Doctoral Dissertation) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil, 2019. Cited in page 82.

MEYMANDPOUR, R.; DAVIS, J. G. A semantic similarity measure for linked data: An information content-based approach. **Knowledge-Based Systems**, v. 109, p. 276–293, out. 2016. Publisher: Elsevier. Cited in page 25.

MILNE, D.; WITTEN, I. H. An Effective, Low-Cost Measure of Semantic Relatedness Obtained from Wikipedia Links. In: **Proceedings of the AAAI 2008 Workshop on Wikipedia and Artificial Intelligence**. Chicago: AAAI Press, 2008. p. 25–30. Cited 2 times in pages 25 and 26.

MOORE, J. L.; STEINKE, F.; TRESP, V. A Novel Metric for Information Retrieval in Semantic Networks. In: GARCíA-CASTRO, R.; FENSEL, D.; ANTONIOU, G. (Ed.). **The Semantic Web: ESWC 2011 Workshops**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. v. 7117, p. 65–79. ISBN 978-3-642-25952-4 978-3-642-25953-1. Series Title: Lecture Notes in Computer Science. Available from Internet: <http://link.springer.com/10.1007/978-3-642-25953-1_6>. Cited 4 times in pages 32, 33, 40, and 65.

PIRRò, G. Explaining and Suggesting Relatedness in Knowledge Graphs. In: ARENAS, M. et al. (Ed.). **The Semantic Web - ISWC 2015**. Cham: Springer International Publishing, 2015. v. 9366, p. 622–639. ISBN 978-3-319-25006-9 978-3-319-25007-6. Series Title: Lecture Notes in Computer Science. Available from Internet: <http://link.springer.com/10.1007/978-3-319-25007-6_36>. Cited 10 times in pages 17, 18, 28, 32, 33, 34, 41, 63, 65, and 66.

PRUD'HOMMEAUX, E.; SEABORNE, A. **SPARQL Query Language for RDF**. 2008. Available from Internet: <https://www.w3.org/TR/rdf-sparql-query>. Cited in page 22.

PRZYJACIEL-ZABLOCKI, M. et al. RDFPath: Path Query Processing on Large RDF Graphs with MapReduce. In: GARCíA-CASTRO, R.; FENSEL, D.; ANTONIOU, G. (Ed.). **The Semantic Web: ESWC 2011 Workshops**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. v. 7117, p. 50–64. ISBN 978-3-642-25952-4 978-3-642-25953-1. Series Title: Lecture Notes in Computer Science. Available from Internet: <http://link.springer.com/10.1007/978-3-642-25953-1_5>. Cited 3 times in pages 18, 36, and 37.

RAGAB, M. et al. An In-depth Investigation of Large-scale RDF Relational Schema Optimizations Using Spark-SQL. In: **Proceedings of the 23rd International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP 2021)**. Nicosia, Cyprus: [s.n.], 2021. p. 71–80. Cited 3 times in pages 18, 36, and 38.

RAGAB, M. et al. Towards making sense of Spark-SQL performance for processing vast distributed RDF datasets. In: **Proceedings of The International Workshop on Semantic Big Data**. Portland, Oregon: ACM, 2020. p. 1–6. ISBN 978-1-4503-7974-8. Available from Internet: <https://dl.acm.org/doi/10.1145/3391274.3393632>. Cited 2 times in pages 36 and 38.

RAGAB, M.; TOMMASINI, R.; SAKR, S. Benchmarking Spark-SQL under Alliterative RDF Relational Storage Backends. In: **Proceedings of the QuWeDa 2019: 3rd Workshop on Querying and Benchmarking the Web of Data**. Auckland, New Zealand: [s.n.], 2019. p. 67–82. Cited 2 times in pages 36 and 38.

ROHLOFF, K.; SCHANTZ, R. E. High-Performance, Massively Scalable Distributed Systems using the MapReduce Software Framework: The SHARD Triple-Store. **Programming support innovations for emerging distributed applications**, p. 1–5, 2010. Cited 2 times in pages 18 and 36.

SCHREIBER, G.; RAIMOND, Y. **RDF 1.1 Primer**. 2014. Available from Internet: <https://www.w3.org/TR/rdf11-primer/>. Cited in page 21.

SCHäTZLE, A. et al. S2X: Graph-Parallel Querying of RDF with GraphX. In: WANG, F. et al. (Ed.). **Biomedical Data Management and Graph Online Querying**. Cham: Springer International Publishing, 2016. v. 9579, p. 155–168. ISBN 978-3-319-41575-8 978-3-319-41576-5. Series Title: Lecture Notes in Computer Science. Available from Internet: <http://link.springer.com/10.1007/978-3-319-41576-5_12>. Cited 2 times in pages 36 and 37.

SCHäTZLE, A. et al. PigSPARQL: A SPARQL Query Processing Baseline for Big Data. In: **Proceedings of the ISWC 2013 Posters & Demonstrations Track**. [S.l.: s.n.], 2013. v. 1035, p. 241–244. Cited in page 37.

SCHäTZLE, A. et al. S2RDF: RDF Querying with SPARQL on Spark. **Proceedings of the VLDB Endowment**, v. 9, n. 10, p. 804–815, jun. 2016. ISSN 21508097. Available from Internet: <http://dl.acm.org/citation.cfm?doid=2977797.2977806>. Cited 3 times in pages 18, 36, and 37.

SIDIROURGOS, L. et al. Column-store support for RDF data management: not all swans are white. **Proceedings of the VLDB Endowment**, v. 1, n. 2, p. 1553–1563, ago. 2008. ISSN 2150-8097. Available from Internet: <https://doi.org/10.14778/1454159.1454227>. Cited in page 24.

solid IT. **DB-Engines Ranking - Popularity ranking of key-value stores**. 2020. Library Catalog: db-engines.com. Available from Internet: <https://db-engines.com/en/ranking/key-value+store>. Cited in page 52.

SUN, J. et al. Dima: a distributed in-memory similarity-based query processing system. **Proceedings of the VLDB Endowment**, v. 10, n. 12, p. 1925–1928, ago. 2017. ISSN 2150-8097. Available from Internet: <https://dl.acm.org/doi/10.14778/3137765.3137810>. Cited in page 35.

SUN, J. et al. Balance-aware distributed string similarity-based query processing system. **Proceedings of the VLDB Endowment**, v. 12, n. 9, p. 961–974, maio 2019. ISSN 2150-8097. Available from Internet: <https://dl.acm.org/doi/10.14778/3329772.3329774>. Cited 2 times in pages 18 and 35.

VIRGILIO, R. D.; MACCIONI, A. Distributed Keyword Search over RDF via MapReduce. In: HUTCHISON, D. et al. (Ed.). **The Semantic Web: Trends and Challenges**. Cham: Springer International Publishing, 2014. v. 8465, p. 208–223. ISBN 978-3-319-07442-9 978-3-319-07443-6. Series Title: Lecture Notes in

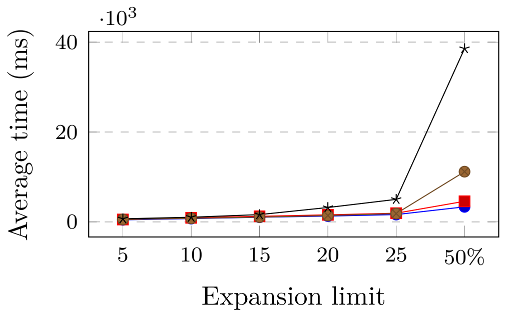
Computer Science. Available from Internet: <http://link.springer.com/10.1007/978-3-319-07443-6_15>. Cited 3 times in pages 18, 36, and 37.

VOCHT, L. D. et al. Effect of Heuristics on Serendipity in Path-Based Storytelling with Linked Data. In: YAMAMOTO, S. (Ed.). **Human Interface and the Management of Information: Information, Design and Interaction**. Cham: Springer International Publishing, 2016. v. 9734, p. 238–251. ISBN 978-3-319-40348-9 978-3-319-40349-6. Series Title: Lecture Notes in Computer Science. Available from Internet: <http://link.springer.com/10.1007/978-3-319-40349-6_23>. Cited in page 33.

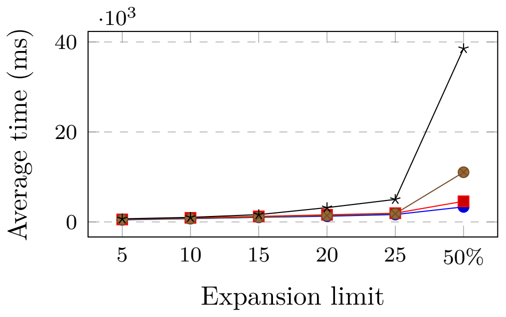
VOCHT, L. D. et al. Discovering Meaningful Connections between Resources in the Web of Data. In: **Proceedings of the 6th Workshop on Linked Data on the Web (LDOW)**. [S.l.: s.n.], 2013. Cited 5 times in pages 17, 18, 32, 33, and 34.

ZAHARIA, M. et al. Spark: Cluster Computing with Working Sets. **HotCloud**, v. 10, n. 10-10, p. 7, 2010. Cited 4 times in pages 23, 35, 36, and 59.

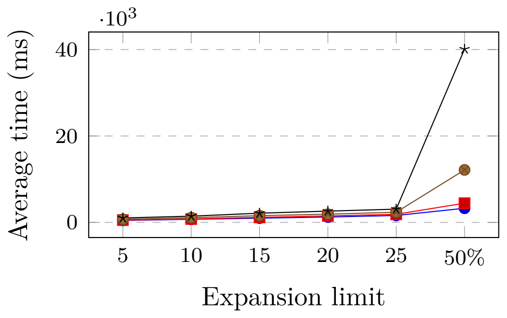
A
Additional Results of the Experiments with DCoEPinKB



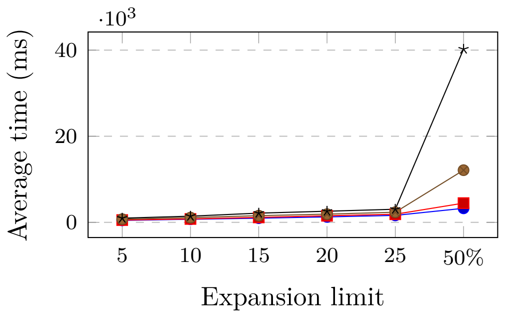
(a) J&I strategy



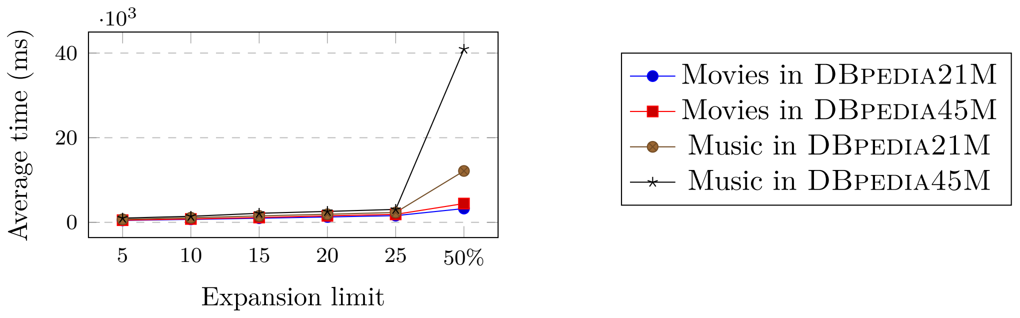
(b) J&P strategy



(c) W&I strategy

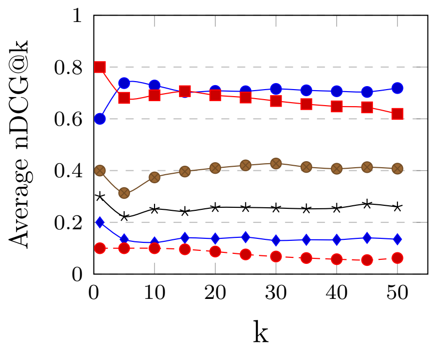


(d) W&E strategy

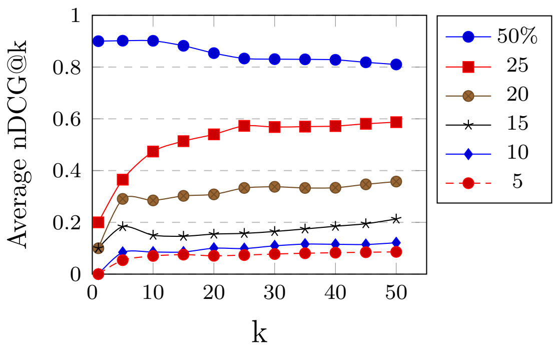


(e) W&P strategy

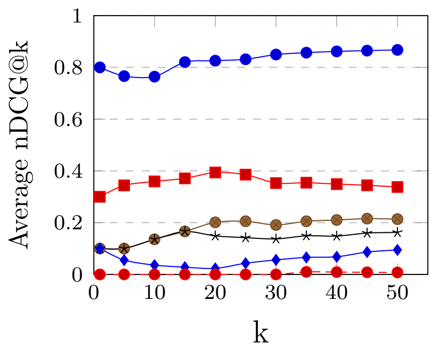
Figure A.1: Average execution time over all entity pairs in each domain and dataset for different strategies varying the expansion limit



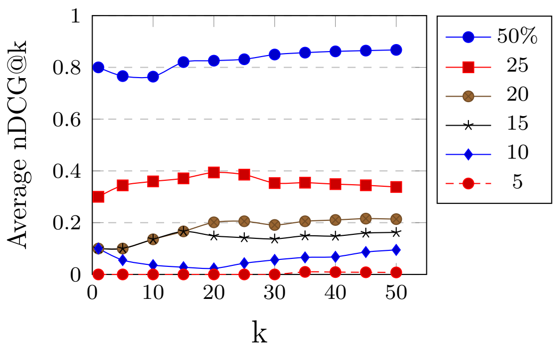
(a) Movies domain in DBpedia21M



(b) Movies domain in DBpedia45M

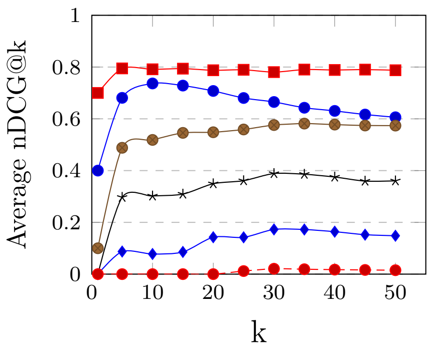


(c) Music domain in DBpedia21M

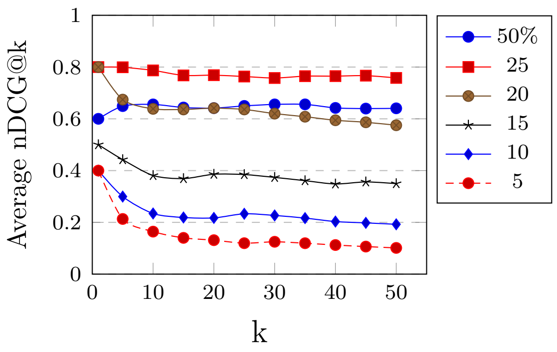


(d) Music domain in DBpedia45M

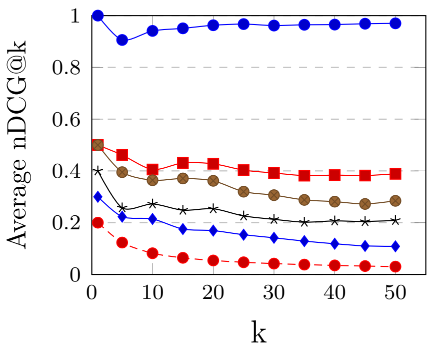
Figure A.2: Average nDCG@k over the movies and music domains for the J&P strategy varying the expansion limit



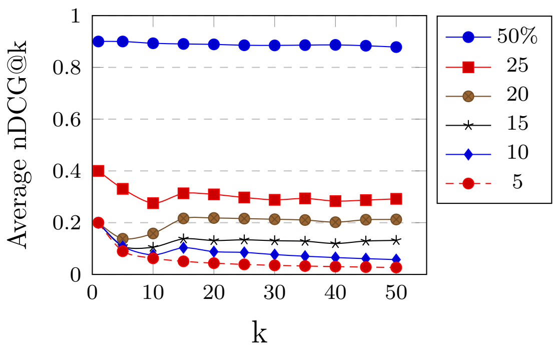
(a) Movies domain in DBpedia21M



(b) Movies domain in DBpedia45M

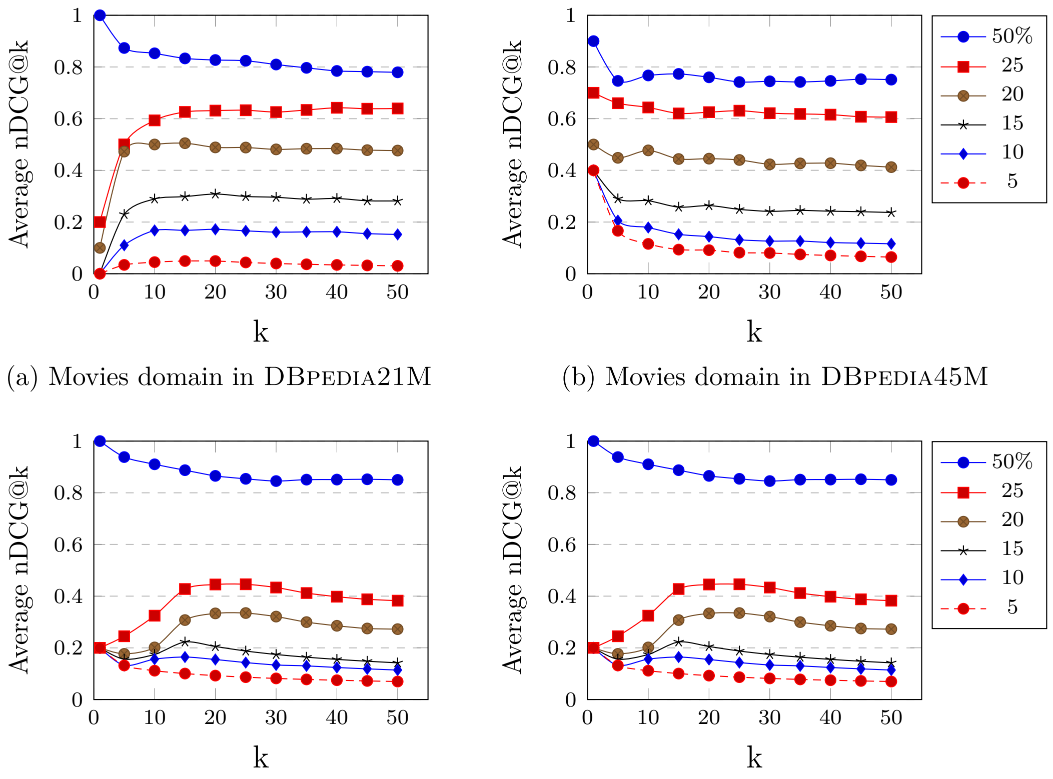


(c) Music domain in DBpedia21M



(d) Music domain in DBpedia45M

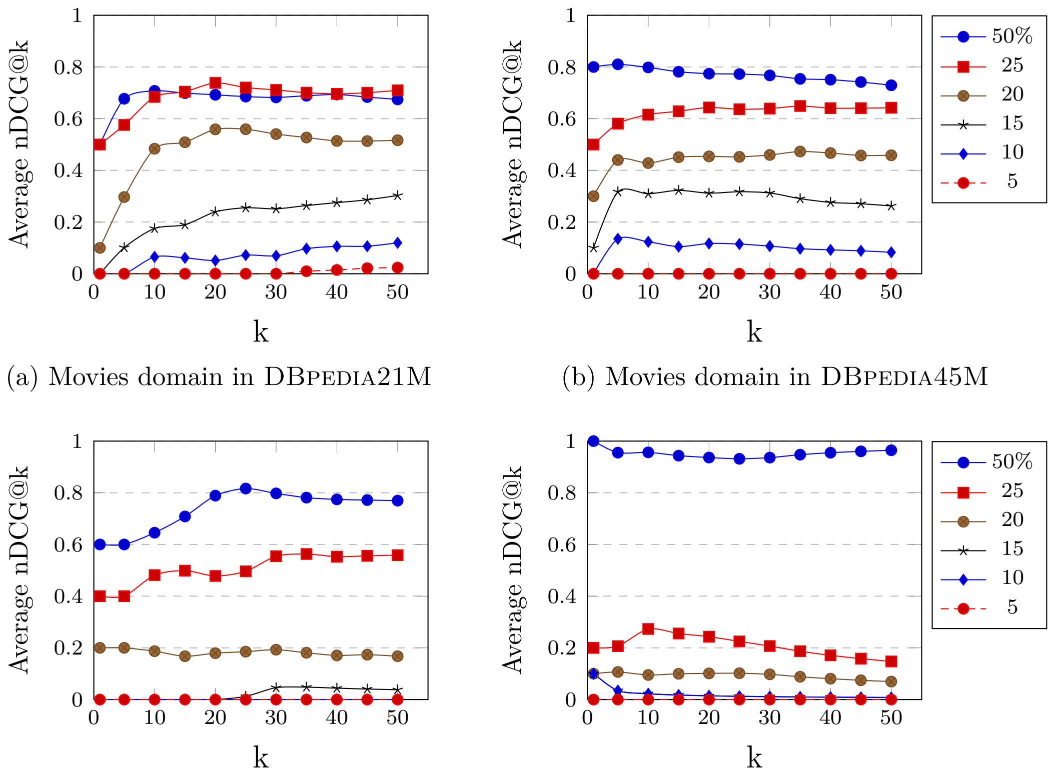
Figure A.3: Average nDCG@k over the movies and music domains for the W&I strategy varying the expansion limit



(a) Movies domain in DBpedia21M (b) Movies domain in DBpedia45M

(c) Music domain in DBpedia21M (d) Music domain in DBpedia45M

Figure A.4: Average nDCG@k over the movies and music domains for the W&E strategy varying the expansion limit



(a) Movies domain in DBpedia21M

(b) Movies domain in DBpedia45M

(c) Music domain in DBpedia21M

(d) Music domain in DBpedia45M

Figure A.5: Average nDCG@k over the movies and music domains for the W&P strategy varying the expansion limit