


CoEPinKB: Evaluating Path Search Strategies in Knowledge Bases

Javier Guillot Jiménez   [Pontifical Catholic University of Rio de Janeiro | jguillot@inf.puc-rio.br]

Luiz André P. Paes Leme  [Fluminense Federal University | lapaesleme@ic.uff.br]

Marco A. Casanova  [Pontifical Catholic University of Rio de Janeiro | casanova@inf.puc-rio.br]

 Informatics Department, Pontifical Catholic University of Rio de Janeiro, R. Marquês de São Vicente, 225 RDC, Gávea, Rio de Janeiro, RJ, 22451-900, Brazil.

Received: 14 September 2021 • Accepted: 6 October 2022 • Published: 14 December 2022

Abstract A knowledge base, expressed using the Resource Description Framework (RDF), can be viewed as a graph whose nodes represent entities and whose edges denote relationships. The entity relatedness problem refers to the problem of discovering and understanding how two entities are related, directly or indirectly, that is, how they are connected by paths in a knowledge base. Strategies designed to solve the entity relatedness 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 paths returned. This article presents a framework, called CoEPinKB, that supports the empirical evaluation of such strategies. The proposed framework allows combining entity similarity and path ranking measures to generate different path search strategies. The main goals of this article are to describe the framework and present a performance evaluation of nine different path search strategies.

Keywords: Entity Relatedness, Similarity Measure, Relationship Path Ranking, Backward Search, Knowledge Base

1 Introduction

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. In this article, we present a framework, called CoEPinKB, that supports exploring a knowledge base to discover and understand how two entities are connected. This is known as the *entity relatedness problem*, formalized 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 them”.

Searching for relevant relationship paths between two entities has applications in several areas. For example, the academic community may be interested in finding interrelationships between researchers in co-authorship networks, or a historian may also want to identify the relationships between two politicians in History. Large knowledge bases describe entities and their relations and can be used to search for these kinds of relationships. However, entities may share too many direct relations and relationship paths, sometimes reaching hundreds of thousands in a graph of billion edges, making it challenging to compute and identify relevant relationship paths between pairs of entities.

Our proposal is based on Herrera [2017], which introduced a two-step strategy to address the entity relatedness problem: (1) search for relationship paths between pairs of entities; and (2) rank the paths and return the top- k , where k is specified by the user. This strategy must, however, be refined to avoid generating and ranking a very large number of paths. To address the first step, the author proposed a generic strategy based on the backward search heuristic [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. Herrera [2017] adapted the expansion process to use activation criteria that 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 [Herrera, 2017]. 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.

The first contribution of this article is the proposal and implementation of a framework that helps address the entity relatedness problem. The framework is called CoEPinKB, an acronym that stands for understanding the **Connectivity of Entity Pairs in Knowledge Bases**. The architecture of CoEPinKB, shown in **Figure 1**, features two main components: the *Backward Search* component 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. The two hot spots are the *activation function*, implementing the entity similarity measure, and the *path ranking measure*. The current version of CoEPinKB implements the three entity similarity measures and the three relationship path ranking measures defined in Section 2.

CoEPinKB differs from the implementation proposed in Herrera [2017] in three aspects. First, it was designed to make it easy for developers to add new entity similarity and relationship path ranking measures to generate new path

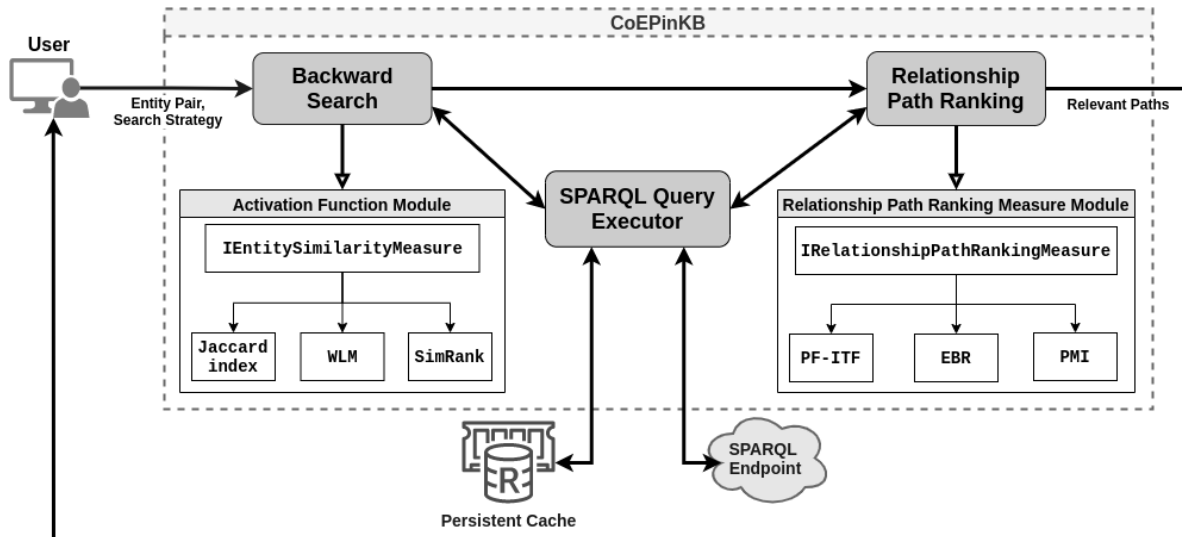


Figure 1. CoEPinKB architecture

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 SPARQL service over HTTP.

Existing analysis [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 article is then a more detailed analysis of the performance of these different strategies concerning execution time on two entertainment domains in DBpedia.

The remainder of this article is organized as follows. Section 2 presents an overview of background information necessary to understand our work. Section 3 briefly reviews related work. Section 4 describes the proposed solution and the process of finding relevant relationship paths between entity pairs through a backward search algorithm. Section 5 presents some technical aspects of the implementation of the proposed framework. Section 6 presents a performance evaluation of path search strategies, using CoEPinKB. Finally, Section 7 presents the conclusions and some directions for future work.

2 Background

In this section, we provide 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. The reader may skip this section on a first reading and go directly to Sections 4 and 5, which are the central contributions of the article.

2.1 RDF

The *Resource Description Framework* (RDF) is a flexible and extensible data model for representing information about resources [Schreiber and 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.

An RDF dataset R is a set of RDF triples. It can be modeled as a labeled graph $G_R = (V_R, E_R, L(E_R))$, where V_R is the set of subjects or objects of the triples in R and there is an edge $e_i(s, o) \in E_R$ (s, o) in E_R from s to o labeled $p = L(e_i(s, o))$ 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 $(w_0, p_1, w_1, p_2, w_2, \dots, p_{k-1}, w_{k-1}, p_k, w_k)$, 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.

2.2 Similarity Measures

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 and 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.

Feature-based similarity measures assume that the concepts can be represented as sets of features and assess the similarity of two concepts based on the commonalities among their feature sets. In this article, 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.

The *Jaccard index* [Jaccard, 1901] 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:

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

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

The *Wikipedia Link-based Measure* (WLM) [Milne and Witten, 2008] measures the semantic similarity of two Wikipedia pages by comparing their incoming and outgoing links. However, it can be also used to measure the similarity between two entities a and b in G and is defined as:

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

where V is the set of entities of G .

SimRank [Jeh and Widom, 2002] measures the similarity of the structural context in which objects occur based on their relationships with other 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:

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

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

The problem of reducing the computational cost of *SimRank* has been addressed, for example, in [Lizorkin and Velikhov, 2008; Li et al., 2010, 2020; Reyhani Hamedani and Kim, 2021].

2.3 Relationship Path Ranking Measures

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.

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 .

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 4 and Equation 5, respectively. The *inverse triple frequency* of p in G , $itf(p, G)$, and the *predicate frequency inverse triple frequency*, $pfif_x^w(p, G)$, are shown in Equation 6 and Equation 7, respectively.

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

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

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

$$pfif_x^w(p, G) = pf_x^w(p, G) \times itf(p, G) \quad (7)$$

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 7, $pfif_x^w(p, G)$ 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:

$$score(\pi(w_0, w_1), G) = \frac{pfif_o^{w_0}(p_1, G) + pfif_i^{w_1}(p_1, G)}{2} \quad (8)$$

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

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

The *Exclusivity-based Relatedness* (EBR), introduced by Hulpuş et al. [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. Using the notation introduced to define PF-ITF, the *exclusivity* of a relationship $a \xrightarrow{p} b$ is defined as:

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

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

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

The *Pointwise Mutual Information* (PMI) [Church and Hanks, 1990] measures the co-occurrence strength between two items. 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 12)
2. Co-occurrence of a property p and an entity w , when p is outgoing from w . (Equation 13)
3. Co-occurrence of a property p and an entity w , when p is incoming to w . (Equation 14)

These cases can be formalized as follows:

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

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

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

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

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

3 Related Work

Several strategies and tools have been proposed to discover the semantic associations between a pair of entities in a knowledge base. Some approaches [Heim et al., 2009; Pirrò, 2015; Herrera et al., 2016] first identify all possible relationships between two entities, using SPARQL queries to retrieve paths up to a certain length, 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 et al., 2012; De Vocht et al., 2013; Cheng et al., 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 et al., 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 that takes a pair of entities in a given knowledge base as input and identifies a ranked list of relationship paths, called by the authors as relationship explanations. 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 adaptation 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 et al. [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.

De 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. De 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 et al. [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. Cheng et al. [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 et al. [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 gener-

ate 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 this tool outperforms RECAP in terms of performance and usability.

As stated above, many approaches [Fang *et al.*, 2011; De Vocht *et al.*, 2013; Cheng *et al.*, 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.fm² 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.

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.

Table 1 compares CoEPinKB with the related systems mentioned above, in terms of the knowledge graphs supported, types of output, availability of filtering capabilities, and the requirement of local data.

Table 1. Comparison of CoEPinKB with related systems

System	Knowledge Graph	Output	Filtering Capabilities	Local Data
RelFinder	DBpedia	Graph	No	Yes
REX	Yahoo!	Graph	No	Yes
EXPLASS	DBpedia	Paths	Yes	Yes
RECAP	Any	Graph, Paths	Yes	No
DBpedia Profiler	DBpedia	Graph, Paths	No	Yes
CoEPinKB	Any	Paths	Yes	No*

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

CoEPinKB differs from most related systems in the following major aspects. As for the RDF knowledge base, only RECAP and our framework are knowledge base independent; CoEPinKB, as RECAP, only requires the availability of a remote SPARQL query endpoint. 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, we remark that none of these related works examined in detail the effects of using different entity similarity measures or path-ranking measures on the execution time of the path search strategies.

4 The CoEPinKB Approach

In this section, we describe 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 based on a single-machine configuration using the data parallel paradigm.

4.1 Overview

Recall that 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 .

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.

¹<https://www.imdb.com/>

²<https://www.last.fm/>

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 $(w_0, p_1, w_1, \dots, p_k, w_k)$ 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 article considers the three entity similarity measures, *Jaccard index*, *WLM*, and *SimRank*, described in **Section 2.2**.

This work 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 *et al.* [2012], 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 *et al.*, 2014], SCS Connector [Nunes *et al.*, 2014], RECAP [Pirró, 2015], and DBpedia Profiler [Herrera *et al.*, 2016], also considered bounded-length paths.

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 article considers the three path ranking measures, PF-ITF, EBR, and PMI, reviewed in **Section 2.3**.

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 2**, which we will evaluate in **Section 6**. The second column of the table contains the acronym used for each strategy hereafter in the document.

Table 2. Path Search Strategies

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

4.2 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. We recall that we assume that the edges of an RDF graph may be traversed in both directions.

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 the 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 τ , 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    $side \leftarrow length \bmod 2;$ 
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).

Algorithm 2 describes the *expansion* process. 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 through 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  $|N_w| \leq d$  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       Add 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       Add  $w_s$  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 returns a list S_w of neighbors similar to w . This list 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). A sub-path is extended by appending the edge from w_s to the activated entity w , regardless of the actual direction of the edge. 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.

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     Add paths resulting from
        $concat(P_{left}[w], P_{right}[w])$  to  $P$ ;
5 if  $b \in P_{left}$  then
6   Add paths in  $P_{left}[b]$  to  $P$ ;
7 if  $a \in P_{right}$  then
8   Add paths in  $P_{right}[a]$  to  $P$ ;
9 return  $P$ 

```

The approach for finding relationship paths in this article is adapted from the algorithm proposed by Herrera [2017]. In that proposal, the join of sub-paths coming from the left and the right is executed after each expansion iteration. That strategy 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.

4.3 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 τ , a real number $\lambda \in [0, 1]$ defining an expansion limit, a path ranking function γ , and a maximum number of paths *k*

Output: a list of the top-*k* relevant paths from *a* to *b*

- 1 $P \leftarrow \text{backwardSearch}(a, b, l, d, \tau, \lambda)$;
 - 2 $R \leftarrow \text{getPathsOrderedByScore}(P, \gamma)$;
 - 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.

5 The CoEPinKB Framework

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* relevant paths between the two entities provided.

During the first phase of the execution of CoEPinKB, the *Backward Search* component communicates with the *SPARQL 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 SPARQL Endpoint through SPARQL queries, 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 *SPARQL Query Executor* component requesting the required data to execute the path ranking algorithm. Finally, after the algorithm finishes, the *Relationship Path Ranking* component sends the list of ranked paths to the user through the user interface.

Frameworks are semi-complete, reusable applications that can be specialized to produce custom applications for a spe-

cific 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 and 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.

There are two key hot spots in CoEPinKB –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. As illustrated in **Figure 1**, the current version of CoEPinKB implements 3 entity similarity measures (i.e., *Jaccard index*, *WLM*, and *SimRank*) and 3 relationship path ranking measures (i.e., *PF-ITF*, *EBR*, and *PMI*).

To add a new activation function, a developer must create a class that implements the `IEntitySimilarityMeasure` interface. This interface has a `getSimilarity` method that receives an entity and a collection of RDF statements that connect the input entity with its neighbors and returns a list with the similarity values between the entity and each of its neighbors. Similarly, to add a new relationship path ranking measure, a developer must create a class that implements the `IRelationshipPathRankingMeasure` interface, which only has the `getPathsOrderedByRank` method declaration that receives a list of paths between a pair of entities and returns this list of paths ordered by the value of the current relationship path ranking measure.

To improve performance, 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. For example, if the task of computing the rank of each path in a list of relationship paths is simple enough (i.e., `paths.size()` is lower than a specified threshold), then the task is executed asynchronously. Otherwise, the list of paths is divided into sublists (the task is divided into subtasks) and the results of all subtasks are recursively joined into a single result.

At the data layer, the framework has the *SPARQL Query Executor* component that interacts with RDF datasets through their SPARQL endpoints. The framework also uses 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 relationship-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.

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

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.

Find Relationship Paths

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

P.F.I.FE

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 2. CoEPinKB UI

same time highly configurable interface in terms of the parameters that make up a search strategy. **Figure 2** shows the CoEPinKB User Interface and an excerpt of the result 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 of relevant paths between the two

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

The CoEPinKB framework is available online³ and was implemented in Java in conjunction with other technologies, such as Apache Jena⁴, a free and open-source Java framework for building Semantic Web and Linked Data applications, to interact with the RDF data sources; Redis⁵, a popular distributed in-memory key-value store, as our persistent cache; and the Jedis⁶ library, which allowed us to interact with a Redis instance from our Java application.

6 Evaluation

Herrera [2017] executed some experiments to evaluate a family of 9 path search strategies (shown in **Table 2**) against a ground truth [Herrera *et al.*, 2017] from the music and movies domains, and a baseline, RECAP [Pirró, 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. Those experiments suggested that the J&E strat-

³<http://semanticweb.inf.puc-rio.br:8080/CoEPinKB/>

⁴<https://jena.apache.org/>

⁵<https://redis.io/>

⁶<https://github.com/redis/jedis>

Table 3. Entity pairs from the music and movies domains

Music domain			Movies domain		
EP	Entity	Degree	EP	Entity	Degree
1	dbr:Michael_Jackson	9000	6	dbr:Elizabeth_Taylor	2319
	dbr:Whitney_Houston	4171		dbr:Richard_Burton	1967
2	dbr:The_Beatles	14105	7	dbr:Cary_Grant	1875
	dbr:The_Rolling_Stones	8602		dbr:Katharine_Hepburn	1916
3	dbr:Elton_John	7796	8	dbr:Laurence_Olivier	2644
	dbr:George_Michael	2703		dbr:Ralph_Richardson	1062
4	dbr:Led_Zeppelin	4810	9	dbr:Errol_Flynn	1423
	dbr:The_Who	4617		dbr:Olivia_de_Havilland	1208
5	dbr:Pink_Floyd	5199	10	dbr:William_Powell	874
	dbr:David_Gilmour	1654		dbr:Myrna_Loy	1039

egy, which adopts the *Jaccard index* and the EBR measure, is the best of the 9 strategies compared and obtained better results than the baselines.

Herrera [2017] did not include experiments to evaluate the performance of all these strategies concerning execution time. This article aims to fill this gap by presenting a performance evaluation of these different strategies using CoEPinKB.

6.1 Experimental Setup

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

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.

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 date⁷. 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 datasets⁸. When the experiments were performed, this dataset contained just over 400 million triples.

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 3** shows the selected entity pairs and the degree of each entity from both domains. 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 reported in Section 6.2.

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 2**.

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 *et al.*, 2014], RECAP [Pirró, 2015], DBpedia Profiler [Herrera *et al.*, 2016], and the experiments in Herrera [2017]. As argued by Nunes *et al.* [2014], paths longer than 4 would express unusual relationships, which users might misinterpret.

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 criterion is applied together with entity similarity during the entity expansion process because it can be assumed that nodes with a high degree often carry very unspecific information that negatively influences the path search process [Moore *et al.*, 2012].

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`

⁷<https://www.dbpedia.org/>

⁸<https://www.dbpedia.org/resources/sparql/>

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 *et al.* [2014]; Hulpuş *et al.* [2015]; 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].

6.2 Experiment Results

This experiment aims to evaluate the performance, in terms of average execution time, of the nine different path search strategies shown in Table 2. 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 3 shows the performance of the path search strategies in both domains.

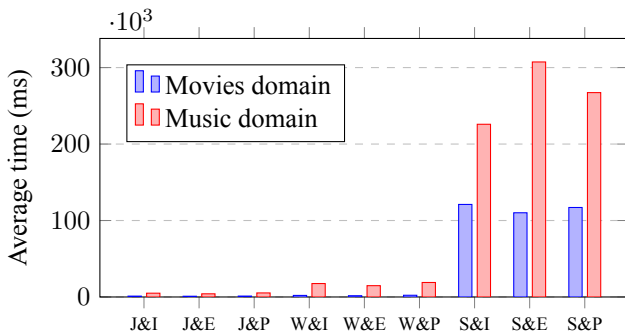


Figure 3. Average time of path search strategies using CoEPinKB

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 and Velikhov, 2008; Li *et al.*, 2010; Reyhani Hamedani and Kim, 2021]. However, as these strategies were not shown to be successful in finding relevant relationship paths in the experiments in Herrera [2017], we focused on the performance analysis of the rest of the strategies, leaving aside those that use *SimRank*.

Figure 4 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*.

We notice that for the entity pairs in the music domain the average execution time of all strategies is higher than the av-

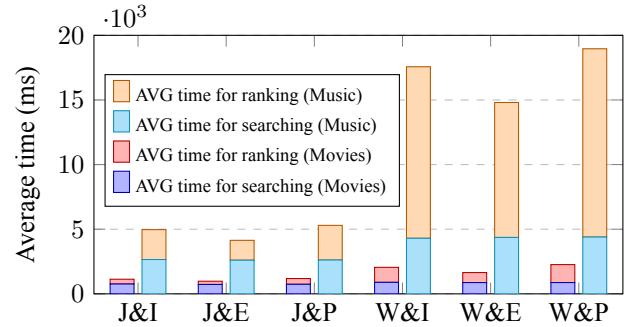


Figure 4. Average execution times of path search strategies in each domain (excluding S&I, S&E, and S&P strategies)

erage 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 5 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.

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

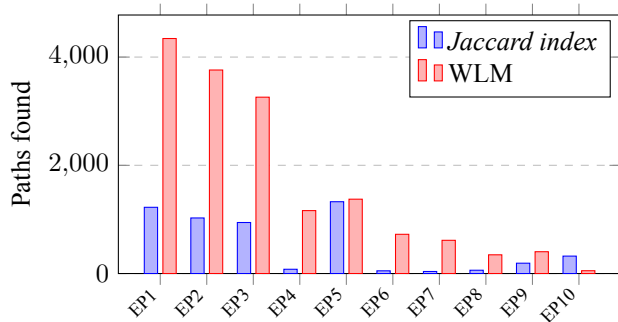


Figure 5. Number of paths found for each entity pair using *Jaccard index* and *WLM* as entity similarity measures

than the baselines. Concerning execution time, the results of this article 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.

7 Conclusions

In this article, we introduced CoEPinKB, a framework that allows empirically evaluating path search strategies that combine entity similarity and path ranking measures, to understand the connectivity of entity pairs in RDF datasets. CoEPinKB supports such evaluation by featuring two flexibility points: the entity similarity and the path ranking measures. Also, CoEPinKB was engineered to work with any knowledge base accessible using a SPARQL service over HTTP. Our performance evaluation of the path search strategies 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 the most effective strategies are also the fastest ones.

As future work, we plan to test the path search strategies in other knowledge bases and implement additional entity similarity and relationship path ranking measures.

Declarations

Funding

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and grants CAPES/88881.134081/2016-01, CNPq/302303/2017-0, CNPq/146411/2018-8, FAPERJ/E-26-202.818/2017, and E-26-200.834/2021.

Authors' Contributions

All authors contributed to the writing of this article, read and approved the final manuscript.

Competing interests

The author(s) declare(s) that they have no competing interests.

Availability of data and materials

Data can be made available upon request.

References

- Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., and Sudarshan, S. (2002). Keyword searching and browsing in databases using BANKS. In *Proceedings 18th International Conference on Data Engineering*, pages 431–440. DOI: 10.1109/ICDE.2002.994756.
- Cheng, G., Liu, D., and Qu, Y. (2021). Fast Algorithms for Semantic Association Search and Pattern Mining. *IEEE Transactions on Knowledge and Data Engineering*, 33(4):1490–1502. DOI: 10.1109/TKDE.2019.2942031.
- Cheng, G., Shao, F., and Qu, Y. (2017). An Empirical Evaluation of Techniques for Ranking Semantic Associations. *IEEE Transactions on Knowledge and Data Engineering*, 29(11):2388–2401. DOI: 10.1109/TKDE.2017.2735970.
- Cheng, G., Zhang, Y., and Qu, Y. (2014). Expluss: Exploring Associations between Entities via Top-K Ontological Patterns and Facets. In *The Semantic Web – ISWC 2014*, volume 8797, pages 422–437. Springer International Publishing, Cham. DOI: 10.1007/978-3-319-11915-1_27.
- Church, K. W. and Hanks, P. (1990). Word Association Norms, Mutual Information, and Lexicography. *Computational Linguistics*, 16(1):22–29.
- Cohen, W. W. (2010). *Graph walks and graphical models*, volume 5. Citeseer.
- De Vocht, L., Beecks, C., Verborgh, R., Mannens, E., Seidl, T., and Van de Walle, R. (2016). Effect of Heuristics on Serendipity in Path-Based Storytelling with Linked Data. In *Human Interface and the Management of Information: Information, Design and Interaction*, volume 9734, pages 238–251. Springer International Publishing, Cham. DOI: 10.1007/978-3-319-40349-6_23.
- De Vocht, L., Coppens, S., Verborgh, R., Sande, M. V., Mannens, E., and de Walle, R. V. (2013). Discovering Meaningful Connections between Resources in the Web of Data. In *Proceedings of the 6th Workshop on Linked Data on the Web (LDOW2013)*.
- Fang, L., Sarma, A. D., Yu, C., and Bohannon, P. (2011). REX: explaining relationships between entity pairs. *Proceedings of the VLDB Endowment*, 5(3):241–252. DOI: 10.14778/2078331.2078339.
- Heim, P., Hellmann, S., Lehmann, J., Lohmann, S., and Stegemann, T. (2009). RelFinder: Revealing Relationships in RDF Knowledge Bases. In *Semantic Multimedia*, volume 5887, pages 182–187. Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-642-10543-2_21.
- Herrera, J. E. T. (2017). *On the Connectivity of Entity Pairs in Knowledge Bases*. Doctoral Dissertation, Pontificia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil.
- Herrera, J. E. T., Casanova, M. A., Nunes, B. P., Leme, L. A. P. P., and Lopes, G. R. (2017). An Entity Relatedness Test Dataset. In *The Semantic Web – ISWC 2017*, volume 10588, pages 193–201. Springer International Publishing, Cham. DOI: 10.1007/978-3-319-68204-4_20.
- Herrera, J. E. T., Casanova, M. A., Nunes, B. P., Lopes, G. R., and Leme, L. (2016). 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).
- Hulpuş, I., Prangnawarat, N., and Hayes, C. (2015). Path-Based Semantic Relatedness on Linked Data and Its Use to Word and Entity Disambiguation. In *The Semantic Web - ISWC 2015*, volume 9366, pages 442–457. Springer International Publishing, Cham. DOI: 10.1007/978-3-319-25007-6_26.
- Jaccard, P. (1901). Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bull Soc Vaudoise Sci Nat*, 37:547–579.
- Jeh, G. and Widom, J. (2002). SimRank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 538–543. ACM.
- Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., and Karambelkar, H. (2005). Bidirectional expansion for keyword search on graph databases. In *Proceedings of the 31st International Conference on Very Large Data Bases, VLDB '05*, pages 505–516, Trondheim, Norway. VLDB Endowment.
- Le, W., Li, F., Kementsietsidis, A., and Duan, S. (2014). Scalable keyword search on large RDF data. *Knowledge and Data Engineering, IEEE Transactions on*, 26(11):2774–2788.
- Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., and Bizer, C. (2015). DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195. DOI: 10.3233/SW-140134.
- Lehmann, J., Schüppel, J., and Auer, S. (2007). Discovering Unknown Connections – the DBpedia Relationship Finder. In *The Social Semantic Web 2007–Proceedings of the 1st Conference on Social Semantic Web (CSSW)*, pages 99–109. Gesellschaft für Informatik e. V.
- Li, C., Han, J., He, G., Jin, X., Sun, Y., Yu, Y., and Wu, T. (2010). Fast computation of SimRank for static and dynamic information networks. In *Proceedings of the 13th International Conference on Extending Database Technology - EDBT '10*, page 465, Lausanne, Switzerland. ACM Press. DOI: 10.1145/1739041.1739098.
- Li, M., Choudhury, F. M., Borovica-Gajic, R., Wang, Z., Xin, J., and Li, J. (2020). CrashSim: An Efficient Algorithm for Computing SimRank over Static and Temporal Graphs. In *Proceedings of the IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1141–1152. IEEE. DOI: 10.1109/ICDE48307.2020.00103.
- Lizorkin, D. and Velikhov, P. (2008). Accuracy Estimate and Optimization Techniques for SimRank Computation. *Proceedings of the VLDB Endowment*, 1(1):12. DOI: 10.14778/1453856.1453904.
- Markiewicz, M. E. and Lucena, C. J. P. (2001). Object oriented framework development. *Crossroads*, pages 10–1145.
- Meymandpour, R. and Davis, J. G. (2016). A semantic similarity measure for linked data: An information content-based approach. *Knowledge-Based Systems*, 109:276–293.
- Milne, D. and Witten, I. H. (2008). An Effective, Low-Cost Measure of Semantic Relatedness Obtained from Wikipedia Links. In *Proceedings of the AAAI 2008 Workshop on Wikipedia and Artificial Intelligence*, pages 25–30, Chicago. AAAI Press.
- Moore, J. L., Steinke, F., and Tresp, V. (2012). A Novel Metric for Information Retrieval in Semantic Networks. In *The Semantic Web: ESWC 2011 Workshops*, volume 7117, pages 65–79. Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-642-25953-1_6.
- Nunes, B. P., Herrera, J., Taibi, D., Lopes, G. R., Casanova, M. A., and Dietze, S. (2014). SCS Connector - Quantifying and Visualising Semantic Paths Between Entity Pairs. In *Proceedings of the Satellite Events of the 11th European Semantic Web Conference (ESWC'14)*, pages 461–466. DOI: 10.1007/978-3-319-11955-7_67.
- Pirró, G. (2015). Explaining and Suggesting Relatedness in Knowledge Graphs. In *The Semantic Web - ISWC 2015*, volume 9366, pages 622–639. Springer International Publishing, Cham. DOI: 10.1007/978-3-319-25007-6_36.
- Reyhani Hamedani, M. and Kim, S.-W. (2021). On Investigating Both Effectiveness and Efficiency of Embedding Methods in Task of Similarity Computation of Nodes in Graphs. *Applied Sciences*, 11(1):162. DOI: 10.3390/app11010162.
- Schreiber, G. and Raimond, Y. (2014). RDF 1.1 Primer.