A distributed framework to investigate the entity relatedness problem in large RDF knowledge bases

Javier Guillot Jiménez[1], **Luiz André P. Paes Leme**[2],
Yenier Torres Izquierdo[1], **Angelo Batista Neves**[1], **Marco A. Casanova**[1]

[1]Department of Informatics, PUC-Rio – Rio de Janeiro – RJ – Brazil

[2]Federal Fluminense University – Niteroi, RJ – Brazil

{jguillot,yizquierdo,ajunior,casanova}@inf.puc-rio.br,lapaesleme@ic.uff.br

Abstract. *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, which combines an entity similarity measure, with 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 paper first introduces* DCoEPinKB*, an in-memory distributed framework that addresses the entity relatedness problem. Then, it presents an evaluation of path search strategies using* DCoEPinKB *over real data collected from DBpedia. The results provide insights about the performance of the path search strategies.*

1. Introduction

An RDF knowledge base K is equivalent to an RDF graph G_K whose nodes represent the entities in K and whose edges denote the relationships expressed in K. This is a convenient representation to explore the connectivity in K of a pair of entities, a and b, which reduces to computing paths in G_K between a and b. This paper focuses on the *entity relatedness problem for large RDF knowledge bases*, defined as: "Given a large RDF knowledge base K and a pair of entities a and b, compute the paths in G_K from a to b that best describe the connectivity between a and b in K".

Following [Fang et al. 2011, Herrera 2017, Jiménez et al. 2021], the paper investigates the entity relatedness problem for large RDF knowledge bases by exploring *path search strategies*, which have two major steps. The first step uses 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. The expansion process prioritizes certain paths over others and filters out entities that are less related to the target entities. The process maintains entities similar to the last entity reached in a partially constructed path, using an entity similarity measure and a threshold, or expansion limit. The second step ranks relationship paths, using a path ranking measure, and returns the top-*k* paths as a description of the connectivity of the entity pair.

However, implementing a path search strategy over a large RDF graph, such as that of DBpedia [Lehmann et al. 2015], is challenging. The paper then introduces a novel framework, called DCoEPinKB, which allows experimenting with path search strategies over large RDF graphs, using different entity similarity measures, expansion limits, and path ranking measures. Unlike the approaches described in [Herrera 2017,

Jiménez et al. 2021], the implementation of the DCoEPinKB framework is distributed and built on top of Apache Spark [Zaharia et al. 2010].

Using DCoEPinKB, the paper presents an evaluation of a family of path search strategies over two large RDF knowledge bases extracted from DBpedia data, DBpedia21M and DBpedia45M, in two entertainment domains. The results provide insights about the impact of the entity similarity measures, the expansion limits, and path ranking measures on the performance of the path search strategies, measured by their execution time and the Normalized Discounted Cumulative Gain (nDCG) [Järvelin and Kekäläinen 2002] of the path rankings obtained.

The main contributions of the paper, therefore, are: (1) a flexible, distributed framework that helps investigate the entity relatedness problem for large RDF knowledge bases; (2) a performance analysis of a family of path search strategies over two entertainment domains over real data available in the DBpedia.

The remainder of this paper is organized as follows. Section 2 discusses path search strategies. Section 3 describes the architecture and some technical aspects of the implementation of the proposed framework. Section 4 describes the evaluation setup of our experiments. Section 5 presents a performance evaluation of two path search strategies, using the proposed framework. Section 6 briefly reviews related work. Finally, Section 7 contains the conclusions and directions for future work.

2. Finding Relevant Relationship Paths between Entity Pairs

Let G be an RDF graph. We consider a family of *path search strategies* that receive as input a pair of target entities (w_0, w_k) and output a ranked list of paths in G 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 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 considers one or both of the following selection criteria: (1) select a path whose entities have less than n neighbors in G; and (2) 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.

The last criterion says that a path can be broken into two parts, $left$ and $right$, such that the entities in the $left$ part are transitively similar to the first entity, w_0, and the entities in the $right$ part are transitively similar to the second entity, w_k. This criterion can be implemented by a backward search strategy that executes two breadth-first searches (BFS) alternately to traverse the RDF graph starting from each input entity. In each expansion step, the BFS uses an entity similarity measure σ and an expansion limit λ to move from a node p_{i-1} to a node p_i iff $\sigma(p_{i-1}, p_i)$ falls in the top λ values. A path is generated if both BFS processes reach a common entity or a target entity. In this paper, we consider two entity similarity measures (i.e., the *Jaccard index* [Jaccard 1901] and the *Wikipedia Link-based Measure* (WLM) [Milne and Witten 2008]) and experiment with various expansion limits.

The second step of each 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 en-

tities are related. In this paper, we consider three path ranking measures: the *Predicate Frequency Inverse Triple Frequency* (PF-ITF) [Pirrò 2015], the *Exclusivity-based Relatedness* (EBR) [Hulpuş et al. 2015], and the *Pointwise Mutual Information* (PMI) [Church and Hanks 1990]. Table 1 shows the six path search strategies obtained, to be evaluated in Section 5.

Table 1. Path Search Strategies

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

3. The DCoEPinKB Framework

To investigate path search strategies, we introduce a distributed framework, called DCoEPinKB[1], which stands for a **D**istributed way of understanding the **C**onnectivity **o**f **E**ntity **P**airs **in K**nowledge **B**ases. DCoEPinKB uses Scala in conjunction with other technologies, such as Apache Spark, for large-scale data processing through Spark SQL and Dataframes; Redis, as a persistent cache; and the `scala-redis` library, for connecting Scala applications to a Redis server. Spark has a programming model similar to MapReduce [Dean and Ghemawat 2008], extended with a data-sharing abstraction called *Resilient Distributed Datasets*, or RDDs, which are distributed collections of data, partitioned across cluster nodes that operate in parallel.

Figure 1 shows an overview of the architecture of DCoEPinKB. The Data Preprocessor component transforms the source files of an RDF knowledge base into files in the Parquet format, partitions these new files into fragments, and distributes the fragments over a cluster. Apache Parquet 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.

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 the path search strategy, 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

[1]The source code of DCoEPinKB is available at https://bitbucket.org/guillot/dcoepinkb/

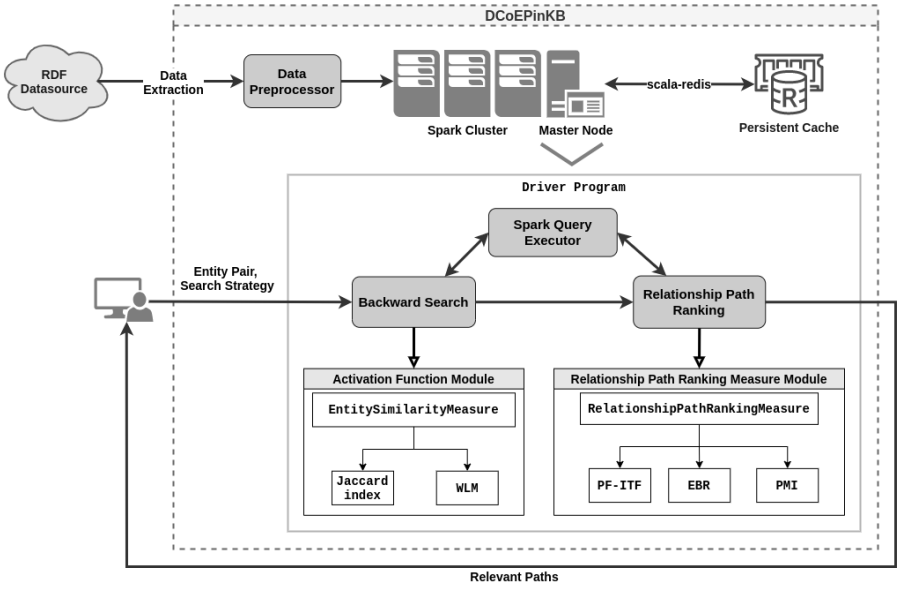


Figure 1. DCoEPinKB architecture

from the Dataframe object in Spark, 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. 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 back to the user.

There are two key points of flexibility in the framework –the activation function, implementing the entity similarity measure and the expansion limit, 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 (specifically using *traits* in Scala), 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 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).

At the data layer, the framework has the SPARK QUERY EXECUTOR component that interacts with the Dataframes that represent different views of RDF datasets stored in the distributed file system as Parquet files. The framework also uses a persistent cache to store the result of the queries executed during the expansion of the path. The main reason for this decision is that both the backward search and the path ranking algorithm require executing a large number of queries.

4. Experimental Setup

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. In fact, we experimented with a proof-of-concept standalone setup, leaving to future work testing the framework in a fully dis-

tributed 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, 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.

Knowledge Bases: We extracted and used two publicly available subsets of the English DBpedia corpus to form our two experiment knowledge bases. 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[2], and the second dataset consists of data from Wikipedia Infoboxes, as it is, with some smart automatic parsing; this dataset[3] 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 – 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 exclude statements involving literals or blank nodes. For each dataset, Table 2 shows the total number of triples (#T), the count of different subjects (#S), properties (#P), and objects (#O); the average out and in node degrees; the size of the source file in Turtle format; and the size of the file after preprocessing and transforming it to Parquet format.

Table 2. Datasets

Dataset	#T	#S	#P	#O	AVG Outdegree	AVG Indegree	Turtle Size	Parquet Size
DBPEDIA21M	21.5 M	5.4 M	632	4.6 M	3.96	4.66	3.1 GB	673 MB
DBPEDIA45M	45.5 M	6.1 M	13691	6.0 M	7.40	7.53	16.2 GB	1.5 GB

Data Storage and Partitioning. We logically represent the datasets using the Statement Table schema, which maps RDF data onto a table with three columns (subject, predicate, object), in which each tuple corresponds to an RDF statement. 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.

Selected Entity Pairs: We selected 20 entity pairs from the Entity Relatedness Test Dataset [Herrera et al. 2017], which contains entities belonging to the movie and music domains (10 entity pairs from each domain). Table 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 1 reported in Section 5.

Configuration parameters: The following parameters were used:

Entity similarity and path ranking measures: as in Table 1.

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

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

Table 3. Entity pairs from music and movies domains

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**

Expansion limit: successively set 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.

Maximum path length between the entities: set to 4, since this was the limit adopted by previous works, as REX [Fang et al. 2011], RECAP [Pirrò 2015], and EXPLASS [Cheng et al. 2014].

Maximum entity degree: set to 200. This degree limit was deduced from DBpedia statistics, which indicate that 90% of the entities have less than 200 links. This kind of criterion is applied together with entity similarity because, as in [Moore et al. 2012], it can be assumed that nodes with a high degree influence the path search process with potentially very unspecific information.

Set of ignored properties: about 10 properties were ignored during the exploration of the knowledge base because many of these properties describe relationships between entities that are irrelevant for our analysis.

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 [Cheng et al. 2014, Fang et al. 2011, Hulpuş et al. 2015, Pirrò 2015].

Ground Truth: 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. Experiment 2 reported in Section 5 describes how we constructed the ground truths.

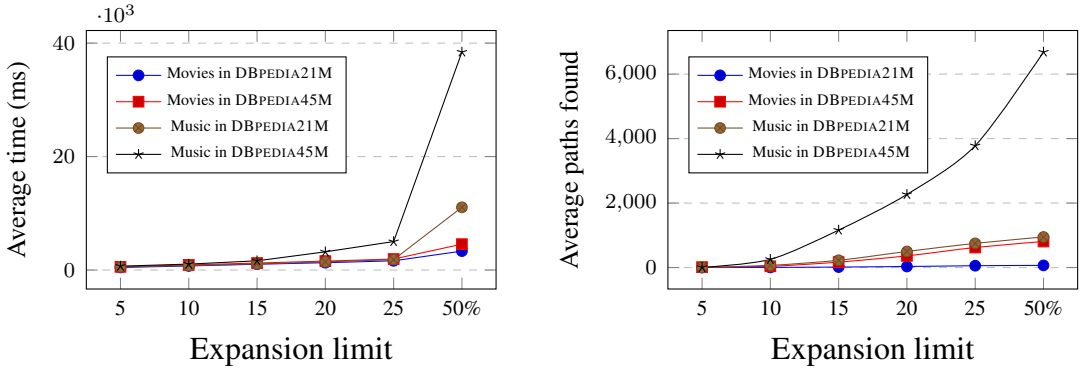
Ranking Quality: We adopted the Normalized Discounted Cumulative Gain (nDCG) to measure the quality of the rankings obtained. The Discounted Cumulative Gain (DCG) is a well-known measure used in Information Retrieval to assess ranking quality. This measure accumulates the gain from the top of a ranked list to the bottom, penalizing

lower ranks, and can be parameterized to consider only the top-k elements of the ranked list. Consider a list with n documents with ratings $rel_1, \ldots, rel_n$, and let the discounted cumulative gain of the top-k results, with $1 \leq k \leq n$, denoted DCG_k, be defined as $DCG_k = rel_1 + \sum_{i=2}^{k} \frac{rel_i}{\log_2(i+1)}$. DCG_k is normalized by $IDCG_k$, the discounted cumulative gain for an ideal ranking of the top-k results. Then, $nDCG_k = \frac{DCG_k}{IDCG_k}$.

5. Evaluation

The experiments in [Herrera 2017] indicated that J&E and W&E perform better than the others 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. The experiments in [Jiménez et al. 2021] showed that the J&E strategy is also the fastest one. Taking into account these results, we conducted the following experiments.

Experiment 1 - 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. Due to space limitations, we only report 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 time in [Jiménez et al. 2021]). Figure 2a shows the average execution times for this strategy. For each pair of entities in each dataset, 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.



(a) Average execution time **(b) Average paths found**

Figure 2. Average execution time and average number of paths found 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, the implementation of 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 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.

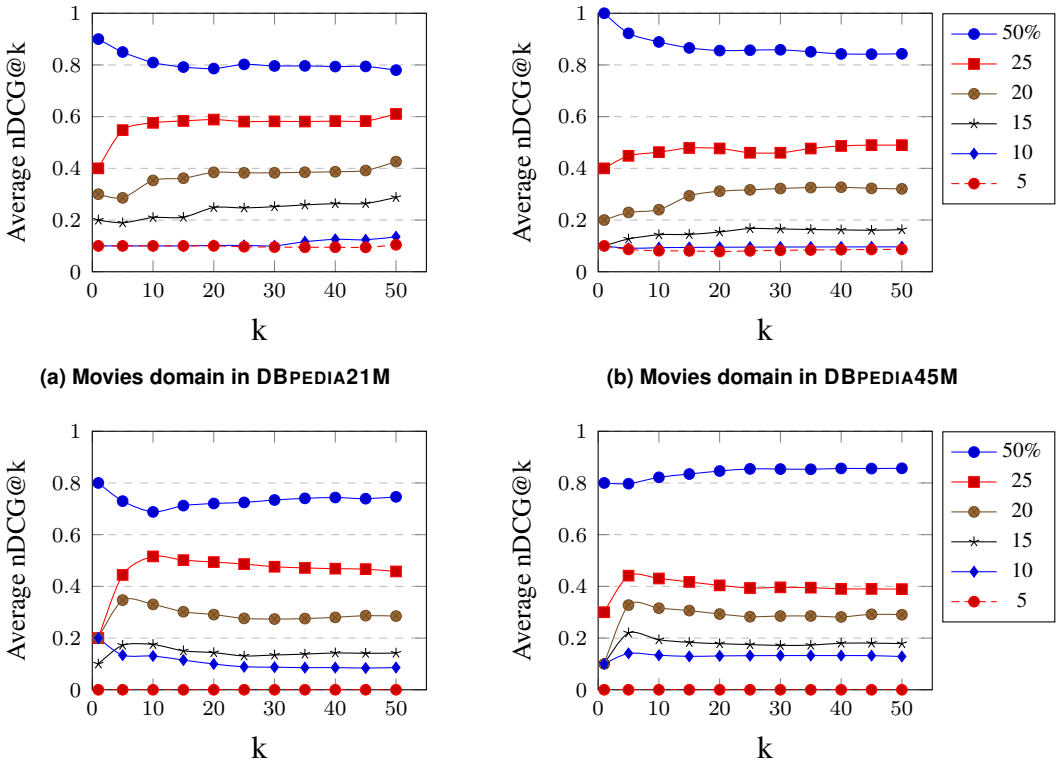
As the execution time depends on the number of paths found, we also show in Figure 2b the average number of paths found for different expansion limits using the

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, as shown in Figure 2b, which implies that the running time also increases.

Experiment 2 - Ranking Accuracy. The second set of experiments evaluated the ranking accuracy of the path search strategies, for different expansion limit values, as compared to a ground truth, using $nDCG_k$, for $k = 1$ to 50. For space limitations, we only report the results for J&E and J&I. The J&I strategy is the second fastest strategy, as stated in [Jiménez et al. 2021], and it also presents advantages in terms of the ranking accuracy in the movies domain using a less expensive expansion limit strategy.

Let π be one of the six path search strategies listed in Table 1. For each entity pair in each DBpedia dataset, we created a separate ground truth path ranking for π by: (1) executing π with different expansion limits; (2) combining all sets of paths thus obtained; (3) ranking the combined set using π, and retaining the top-50 ranked paths. This permits evaluating the impact of the expansion limit, for a given entity similarity measure and a path ranking measure.[4]

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.



(a) Movies domain in DBPEDIA21M

(b) Movies domain in DBPEDIA45M

(c) Music domain in DBPEDIA21M

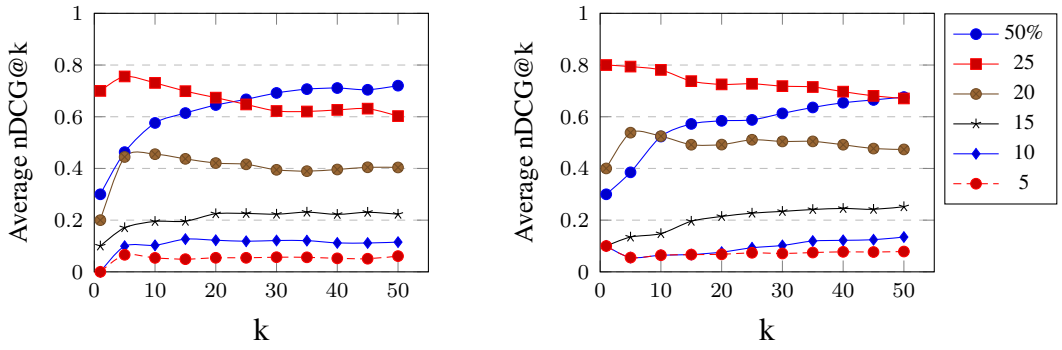
(d) Music domain in DBPEDIA45M

Figure 3. Average nDCG@k for the J&E strategy varying the expansion limit

[4]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

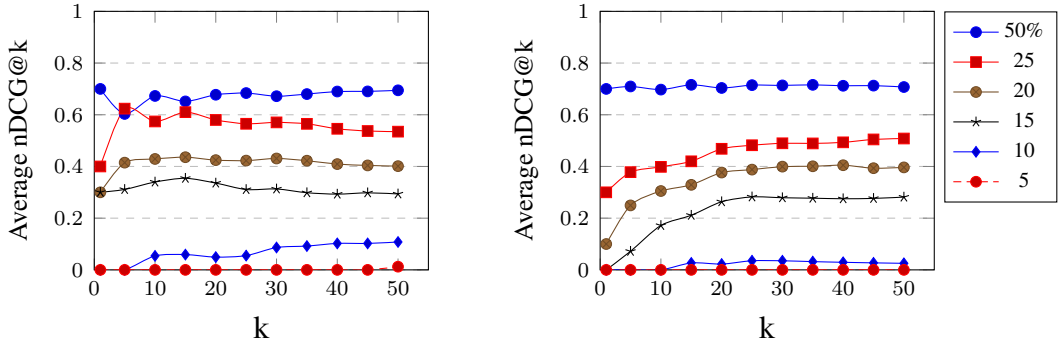
Figure 3 shows the average $nDCG_k$ for the J&E strategy for the movies and music domains in DBPEDIA21M and DBPEDIA45M. For the movies domain, J&E with top-50% obtained a good performance in both datasets: the average $nDCG_k$ was above 0.80 using DBPEDIA21M (Figure 3a), and above 0.86 using DBPEDIA45M (Figure 3b), 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 3c), and above 0.84 using DBPEDIA45M (Figure 3d).

Finally, note that, although J&E with top-50% had a high average execution time over DBPEDIA45M, 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



(a) Movies domain in DBPEDIA21M

(b) Movies domain in DBPEDIA45M



(c) Music domain in DBPEDIA21M

(d) Music domain in DBPEDIA45M

Figure 4. Average nDCG@k for the J&I strategy varying the expansion limit

Figure 4 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 most similar adjacent nodes. Indeed, the average $nDCG_k$ for J&I with top-25 was above 0.67 using DBPEDIA21M (Figure 4a), and above 0.73 using DBPEDIA45M (Figure 4b). Figure 4a 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 4b 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%. Furthermore, 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 4c).

Lastly, due to space limitations, we did not include the average execution times of the J&I strategy, but we observe that they were almost identical to those shown in Figure 2a for the J&E strategy. Hence, given the high average execution time of J&I with top-50%, it is worth opting for J&I with top-25, mainly if one wants the first few most relevant paths.

6. Related Work

Processing Large RDF Datasets in Distributed Environments. Distributed SPARQL query engines are generally built on top of distributed data processing frameworks, such as MapReduce [Dean and Ghemawat 2008] or Spark [Zaharia et al. 2010]. Husain et al. [Husain et al. 2011] designed a storage scheme to store RDF data in HDFS. They proposed a greedy algorithm that produces a query plan with a minimal number of Hadoop jobs and built a framework that supports data-intensive query processing. However, MapReduce is not efficient to perform join-intensive tasks typical of graph algorithms [De Virgilio and Maccioni 2014].

Schätzle et al. [Schätzle et al. 2016] described a relational partitioning schema for RDF data, and built a prototype system on top of Spark that achieved sub-second runtimes for the majority of queries on a billion triples RDF graph. The results of a comparative survey of 22 state-of-the-art distributed RDF systems [Abdelaziz et al. 2017] suggests that specialized in-memory systems provide the best performance, assuming the data can fit in the cumulative memory of the computing cluster. Ragab et al. [Ragab et al. 2021] presented a systematic analysis of the performance of the Spark-SQL query engine for answering SPARQL queries over large RDF datasets in a distributed environment. The experiments show interesting insights about the impact of the relational encoding scheme, storage backends, and storage formats on the performance of the query execution process.

Entity Relationship Discovery and Ranking in Knowledge Bases. Several strategies have been proposed to discover the semantic associations between entities in a knowledge base [Fang et al. 2011, Moore et al. 2012, De Vocht et al. 2013, Cheng et al. 2014, Pirrò 2015, Herrera et al. 2016, Herrera 2017]. While path-ranking measures were proposed, for example, in [Cheng et al. 2014, Hulpuş et al. 2015, Pirrò 2015].

REX [Fang et al. 2011] is a system that uses two BFS on the RDF graph to enumerate relationship paths between two entities and considers the degree of a node as an activation criterion to prioritize nodes to produce a ranked list of relationship paths. EXPLASS [Cheng et al. 2014], RECAP [Pirrò 2015], and DBpedia Profiler [Herrera et al. 2016] implement pathfinding processes in an RDF knowledge graph with the help of SPARQL queries. A family of path search strategies, based on the backward search heuristic, that combines entity similarity and path-ranking measures was introduced in [Herrera 2017]. The COEPINKB framework [Jiménez et al. 2021] extends the previous work and implements path search strategies using a multi-thread approach.

Table 4 compares DCOEPINKB with related systems. As for the RDF knowledge base, only RECAP, COEPINKB and DCOEPINKB are knowledge base independent; 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. Regarding the architecture, DCoEPinKB is the only approach that addresses the entity relatedness problem in a distributed manner.

Table 4. Comparison of DCoEPinKB with related systems

System	KB	Output	Filtering Capabilities	Local Data	Architecture
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.

Benchmarks. A benchmark to evaluate the accuracy of path search strategies was introduced in [Herrera et al. 2017]. However, the benchmark was based on specific versions of certain knowledge bases. Section 5 of this paper circumvents this problem by introducing a strategy to construct ground truths for any given version of a knowledge base.

7. Conclusions

This paper introduced a distributed framework, called DCoEPinKB, that addresses the entity relatedness problem in large RDF knowledge bases. DCoEPinKB features two points of flexibility: (1) the entity similarity measure, with an expansion limit; and (2) the path ranking measure. It works with any knowledge base stored using the Parquet data format in a distributed file system, and even in a traditional file system on a single node. In particular, the experiments showed that by 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 quality of the ranking, when only the first few (i.e., 10 or less) top paths are requested.

As future work, we plan to deploy the DCoEPinKB framework in a fully distributed environment and to implement additional entity similarity and relationship path ranking measures. We also plan to investigate effective optimization techniques in Spark to reduce the execution time for path search strategies.

References

Abdelaziz, I. et al. (2017). A survey and experimental comparison of distributed SPARQL engines for very large RDF data. *Proc. VLDB Endowment*, 10(13):2049–2060.

Cheng, G. et al. (2014). Explass: Exploring Associations between Entities via Top-K Ontological Patterns and Facets. In *ISWC 2014*, volume 8797, pages 422–437.

Church, K. W. and Hanks, P. (1990). Word Association Norms, Mutual Information, and Lexicography. *Computational Linguistics*, 16(1):22–29.

De Virgilio, R. and Maccioni, A. (2014). Distributed Keyword Search over RDF via MapReduce. In *The Semantic Web: Trends and Challenges*, volume 8465, pages 208–223. Springer International Publishing, Cham.

De Vocht, L. et al. (2013). Discovering Meaningful Connections between Resources in the Web of Data. In *Proc. of the 6th Workshop on Linked Data on the Web*, page 8.

Dean, J. and Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Comm. of the ACM*, 51(1):107–113.

Fang, L., Sarma, A. D., Yu, C., and Bohannon, P. (2011). REX: explaining relationships between entity pairs. *Proc. VLDB Endowment*, 5(3):241–252.

Herrera, J. E. T. (2017). *On the Connectivity of Entity Pairs in Knowledge Bases*. Doctoral Dissertation, PUC-Rio, Rio de Janeiro, Brazil.

Herrera, J. E. T. et al. (2016). DBpedia Profiler Tool: Profiling the Connectivity of Entity Pairs in DBpedia. In *IESD 2016*.

Herrera, J. E. T. et al. (2017). An Entity Relatedness Test Dataset. In *The Semantic Web – ISWC 2017*, volume 10588, pages 193–201. Springer International Publishing, Cham.

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.

Husain, M. et al. (2011). Heuristics-Based Query Processing for Large RDF Graphs Using Cloud Computing. *IEEE Trans Knowl Data Eng*, 23(9):1312–1327.

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.

Jiménez, J. G., Leme, L. A. P. P., and Casanova, M. A. (2021). CoEPinKB: A Framework to Understand the Connectivity of Entity Pairs in Knowledge Bases. In *Proc. Integrated Software and Hardware Seminar (SEMISH)*, pages 97–105. SBC. ISSN: 2595-6205.

Järvelin, K. and Kekäläinen, J. (2002). Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446.

Le, W., Li, F., Kementsietsidis, A., and Duan, S. (2014). Scalable keyword search on large RDF data. *IEEE TKDE*, 26(11):2774–2788.

Lehmann, J. et al. (2015). DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195.

Milne, D. and Witten, I. H. (2008). An Effective, Low-Cost Measure of Semantic Relatedness Obtained from Wikipedia Links. In *Proc. 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 *ESWC*, volume 7117, pages 65–79. Springer.

Pirrò, G. (2015). Explaining and Suggesting Relatedness in Knowledge Graphs. In *The Semantic Web - ISWC 2015*, volume 9366, pages 622–639. Springer, Cham.

Ragab, M. et al. (2021). An In-depth Investigation of Large-scale RDF Relational Schema Optimizations Using Spark-SQL. In *DOLAP 2021*, pages 71–80, Nicosia, Cyprus.

Schätzle, A., Przyjaciel-Zablocki, M., Skilevic, S., and Lausen, G. (2016). S2RDF: RDF Querying with SPARQL on Spark. *Proc. VLDB Endowment*, 9(10):804–815.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: Cluster Computing with Working Sets. *HotCloud*, 10(10-10):7.