A Framework to Compute Entity Relatedness in Large RDF Knowledge Bases

Javier Guillot Jiménez[1], Luiz André P. Paes Leme[2],
Yenier Torres Izquierdo[1], Angelo Batista Neves[3,1], Marco A. Casanova[3,1]

[1] Tecgraf Institute, PUC-Rio, Rio de Janeiro, RJ – Brazil
{javiergj, ytorres}@tecgraf.puc-rio.br
[2] Institute of Computing, Federal Fluminense University, Niterói, RJ - Brazil
lapaesleme@ic.uff.br
[3] Department of Informatics, PUC-Rio, Rio de Janeiro, RJ – Brazil
{ajunior,casanova}@inf.puc-rio.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 article addresses such problem by combining distributed RDF path search and ranking strategies in a framework called DCoEPinKB, which helps reduce the overall execution time in large RDF graphs and yet maintains adequate ranking accuracy. The framework allows the implementation of different strategies and enables their comparison. The article also reports experiments with data from DBpedia, which provide insights into the performance of different strategies.

Categories and Subject Descriptors: H.2 [**Database Management**]: Physical Design; H.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

Keywords: entity relatedness, RDF graph, path search strategy, entity similarity, path ranking, DBpedia, SPARK

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 article 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".

Computing the relatedness between a pair of entities may return interesting results. Consider, for example, DBpedia [Lehmann et al. 2015] – a triplified version of Wikipedia – and the problem of discovering how the nodes that represent Albert Einstein and Kurt Gödel are related. Indeed, there are more than 10,000 paths in DBpedia between the nodes representing these scientists. One discovers, for example, that they were friends, neighbors (at Princeton, New Jersey), and colleagues (at the Institute for Advanced Study - IAS), that Gödel demonstrated the existence of solutions involving closed timelike curves, to Einstein's field equations in general relativity, and that Gödel was awarded (with Julian Schwinger) the first Albert Einstein Award in 1951[1].

[1] https://en.wikipedia.org/wiki/Kurt_Gödel#Princeton,_Einstein,_U.S._citizenship (Last accessed on May 26th, 2022).

However, computing the relatedness between pairs of entities over large RDF graphs is challenging due to the potentially explosive number of paths between a pair of entities, as the above example illustrates. As a further indication, the experiments described in this article used two versions of DBpedia with 21M and 45M edges. Furthermore, for the 21M and 45M versions, the average outdegrees are approximately 4.0, and 7.4 and the average indegrees are 4.5 and 7.5, respectively. This is a reasonable indication that the potential number of paths between a pair of entities in these graphs can be quite large.

Following Fang et al. [2011], Herrera [2017] and Jiménez et al. [2021], this article then 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* (BSH) [Le et al. 2014], which is a breadth-first search strategy that expands the paths starting from each input entity, 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, a threshold, and an 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.

The article describes a framework, called DCoEPinKB, which allows experimenting with various path search strategies over large RDF graphs, using different entity similarity measures, expansion limits, and path ranking measures. First introduced in [Guillot Jiménez et al. 2021] and unlike the approaches described in Herrera [2017] and Jiménez et al. [2021], DCoEPinKB is built on top of Apache Spark [Zaharia et al. 2010]. The experiments use a proof-of-concept standalone setup, leaving to future work testing the framework in a fully distributed environment. However, the methods used for transforming and partitioning the source datasets, as well as the data structures used for representing data, are the same regardless of the architecture used.

The article uses DCoEPinKB to evaluate a family of path search strategies over two large RDF knowledge bases extracted from DBpedia data, in two entertainment domains. The results provide insights into 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 article, therefore, are: (1) a flexible 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 article is an improved version of [Guillot Jiménez et al. 2021].

The remainder of this article 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 the experiments. Section 5 presents a performance evaluation of 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

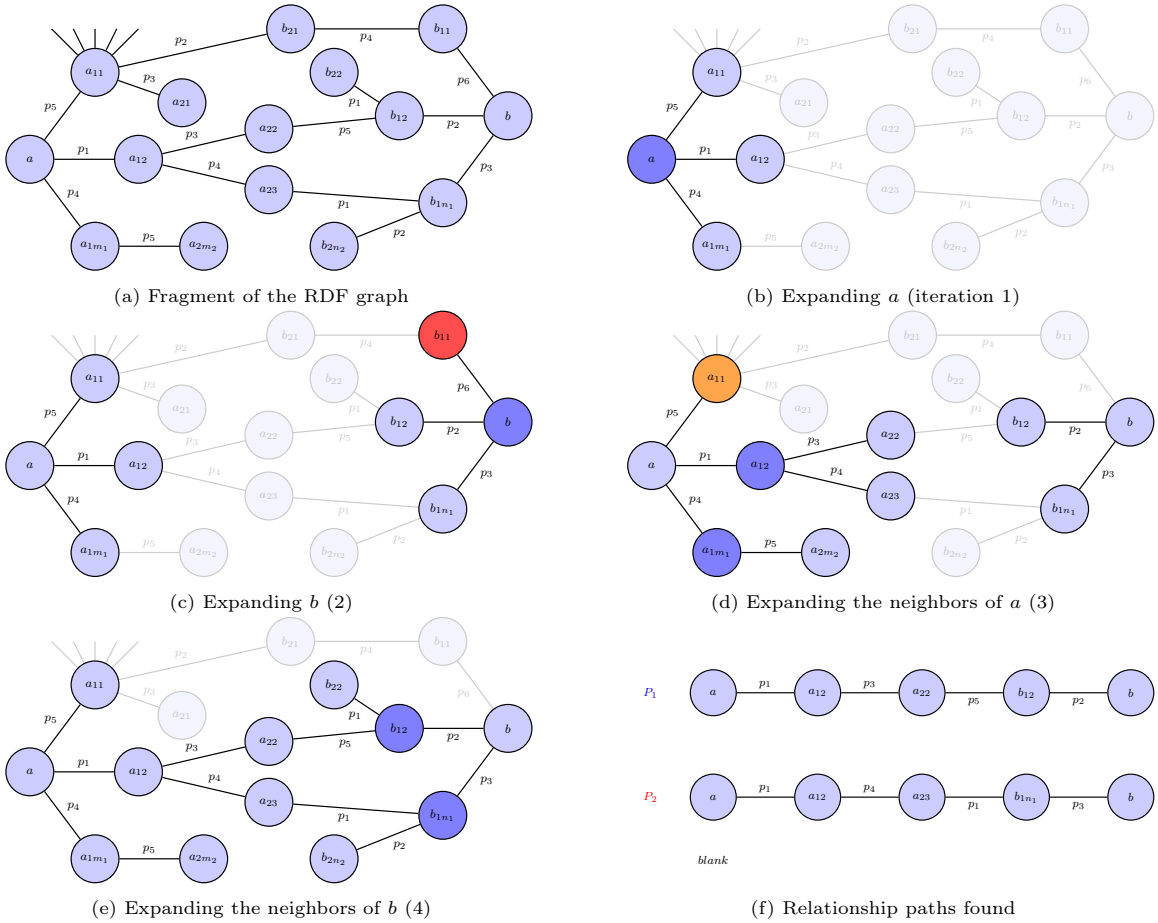
Let G be an RDF graph. We consider a family of *path search strategies* that receive as input a pair of target entities (w_0, w_k) and output a ranked list of paths in G from w_0 to w_k. Each path search strategy in the family has two basic steps: (1) find a set of paths in G from w_0 to w_k such that each path satisfies a set of *selection criteria*; (2) rank the paths found and select the top-k 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 $(w_0, w_1, w_2, \ldots, w_{k-1}, w_k)$ 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 related by similarity to the first entity, w_0, and the entities in the $right$ part are transitively related by similarity to the second entity, w_k. This strategy is implemented with a *Backward Search Heuristic* (BSH) [Le et al. 2014] that executes two breadth-first search (BFS) processes, alternating between left and right, in a single thread, to traverse the RDF graph starting from each input entity. In each expansion step, the BFS process uses an entity similarity measure σ and an expansion limit λ to move from a node w_{i-1} to a node w_i. The movement is allowed iff $\sigma(w_{i-1}, w_i)$ falls in the top λ values. A path is generated if both BFS processes reach a common entity or a target entity. Common entities are not expanded anymore. This article compares two entity similarity measures, the *Jaccard index* [Jaccard 1901] and the *Wikipedia Link-based Measure* (WLM) [Milne and Witten 2008], and various expansion limits.

As an example, Figure 1 shows a simple BSH execution to find relationship paths of maximum size equal to 4 between two entities a and b in an RDF graph. As argued by Pereira Nunes et al. [2014], paths longer than four would express unusual relationships, which users might misinterpret. It is important to note that the example only presents a fragment of the entire RDF graph (Figure 1a).



(a) Fragment of the RDF graph

(b) Expanding a (iteration 1)

(c) Expanding b (2)

(d) Expanding the neighbors of a (3)

(e) Expanding the neighbors of b (4)

(f) Relationship paths found

Fig. 1: Backward search execution example

The first iteration of the thread (Figure 1b) expands the entity a and finds the most similar neighbors

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

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 entities are related. This article considers 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].

Just to exemplify, we present some paths in DBpedia. The nodes that represent Elizabeth Taylor and Richard Burton are directly related by the path `(dbr:Elizabeth_Taylor, dbo:spouse, dbr:Richard_Burton)`, which shows that these entities are related because Elizabeth Taylor married Richard Burton. The following paths reveal more complex relationships:

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

The first path says that Elizabeth Taylor starred in the movie "Doctor Faustus", which Richard Burton produced, and the second says that Elizabeth Taylor starred in the movie "Love Is Better Than Ever", directed by Stanley Donen, who also directed the film "Staircase", which Richard Burton starred. Note that the paths pass through entities representing movies or film directors, which are not discarded by the path finding step because they have some degree of similarity, not based on their classes or information domains, but rather computed from the set of nearby entities.

This behavior is also illustrated when trying to discover the connectivity between The Beatles and The Rolling Stones:

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

The first path says that The Beatles had a special guest appearance by Brian Jones, which was a member of the Rolling Stones, and the second says that The Beatles recorded "Twist and Shout" on the same label, Atlantic Records, as the Rolling Stones.

Finally, Table I summarizes the six path search strategies evaluated in Section 5.

Table I: 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

The acronym DCoEPinKB[2] stands for a **D**istributed way of understanding the **C**onnectivity **o**f **E**ntity **P**airs **in** **K**nowledge **B**ases. The DCoEPinKB framework was implemented in Scala with the help of other technologies, such as Apache Spark, Redis for persistent cache, and the `scala-redis`[3] library for connecting to a Redis server.

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

Figure 2 shows the DCoEPinKB architecture.

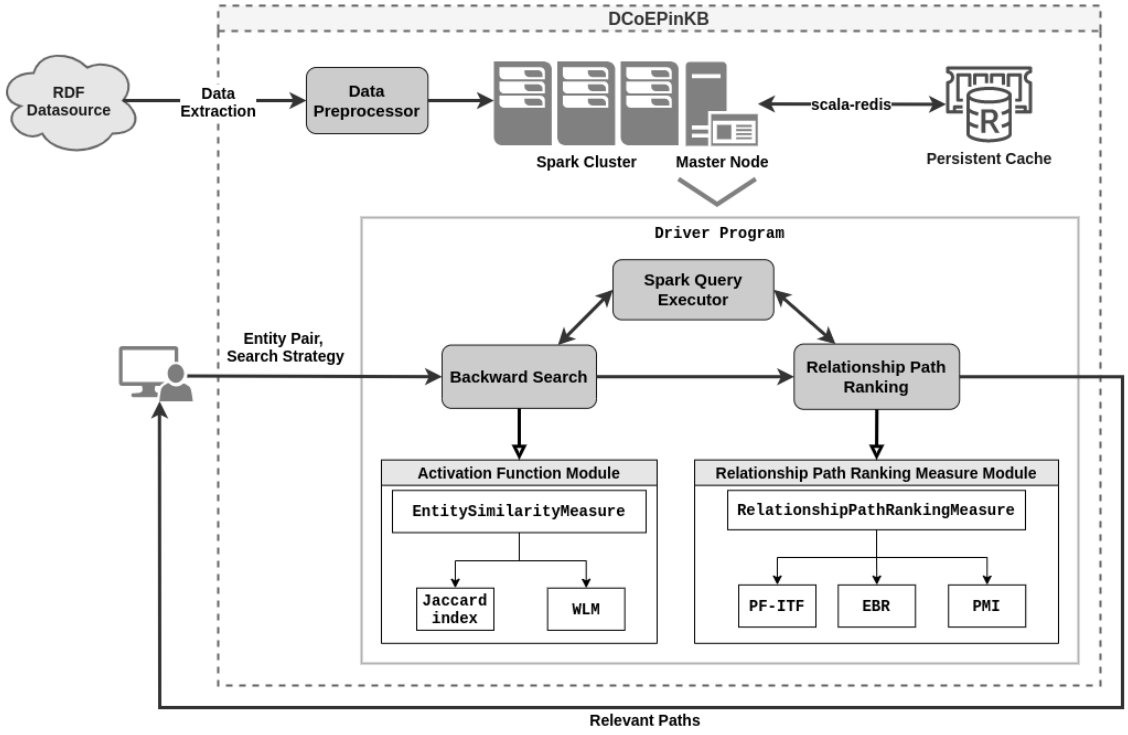


Fig. 2: DCoEPinKB Architecture

The DCoEPinKB framework stores the RDF knowledge base using the Statement Table scheme, which maps RDF data onto a table with three columns (subject, predicate, object), with each tuple

[2]The source code of DCoEPinKB is available at https://bitbucket.org/guillot/dcoepinkb/ (Last accessed on May 26th, 2022)
[3]https://github.com/debasishg/scala-redis (Last accessed on May 26th, 2022)

corresponding to an RDF statement. This scheme proved adequate due to the type of queries our framework performs and the particularities of the graphs used in experiments, which have a large number of properties.

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. Apache Parquet[4] is a columnar storage format that compresses the data and allows fetching specific columns from data as needed, resulting in less I/O usage and improving the performance of the queries. The fragments have similar sizes and are allocated considering a round-robin partitioning fashion, using the number of cores to estimate the number of resulting partitions.

The Spark Query Executor component interacts with DataFrames that represent views of RDF datasets stored in the distributed file system as Parquet files. Based on previous experience Jiménez et al. [2021], we implemented translations of the required SPARQL queries to Spark SQL queries. Spark SQL is one of the most popular modules of Spark, targeted for processing structured data, using the Datasets and DataFrames data abstractions, and provides support for reading and writing Parquet files.

DCoEPinKB uses Redis as a persistent cache to store the result of the queries executed during the expansion of the entities in the RDF graph. The structure of the keys in the cache was designed in a way that facilitates partitioning data on a cluster of Redis nodes using a concept called *hash tags*[5]. So, these *hash tags* can be used to force certain keys to be stored in the same hash slot.

The current implementation uses a centralized cache. However, the definition of the keys of the elements that are allocated in the cache considered the possibility of working with Redis Cluster, which provides the ability to perform the data sync automatically across multiple Redis nodes. In a Redis Cluster, the keys are conceptually part of the hash slots, and each node in the cluster is responsible for a subset of the hash slots. The implementation computes the hash slot for a certain key K as the CRC16 of K modulo 16384. Therefore, the key definitions use a pattern that allows grouping, in the same hash slot, data related to a given entity. In a distributed environment using Redis Cluster, we only need to use {} brackets on the already defined keys to use this feature, which forces multiple keys to be mapped into the same hash slot.

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

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

After the backward search algorithm finishes, the Backward Search component sends a list of relationship paths between the pair of entities to the Relationship Path Ranking component. Then, the second phase begins. Similarly to the previous phase, the Relationship Path Ranking

[4]https://parquet.apache.org/ (Last accessed on May 26th, 2022)
[5]https://redis.io/topics/cluster-spec (Last accessed on May 26th, 2022)

component communicates with the Spark Query Executor component requesting the required data to execute the path ranking algorithm. After the algorithm finishes, the Relationship Path Ranking component sends the list of ranked paths to the user through the user interface.

There are two main hot spots in the DCoEPinKB framework: the activation function, implementing the entity similarity measure, and the path ranking measure. The Backward Search and the Relationship Path Ranking components were designed using an architectural pattern based on interfaces, specifically using *traits* in Scala. As illustrated in Figure 2, the current version of DCoEPinKB implements two entity similarity measures, Jaccard Index and WLM, and three relationship path ranking measures, PF-ITF, EBR, and PMI.

Finally, DCoEPinKB runs in batch mode, and the user interacts with the framework using the console. A goal for future work is to improve the framework to run in interactive mode with a graphical user interface, probably using Apache Livy[6]. Currently, the user creates a simple text file that contains the input parameters of the algorithm and that the framework uses to execute the search strategy. The results are then returned in CSV files.

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, using Spark v2.4.3 in the Spark Standalone Mode and Redis v3.0.6. Therefore, as already mentioned in the introduction, the experiments ran on a proof-of-concept standalone setup. However, 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, remain 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[7], and the second dataset consists of data from Wikipedia Infoboxes, as it is, with some smart automatic parsing; this dataset[8] has better fact coverage than the first one but has less consistency. DBpedia21M contains the statements in the first source dataset, and DBpedia45M contains the union of the triples in both source datasets. In both cases, we excluded statements involving literals or blank nodes. For each dataset, Table II 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 (details in the next paragraph).

Table II: 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 represented each dataset, DBpedia21M and DBpedia45M, using the Statement Table schema, which we recall maps RDF data onto a table with three

[6]https://livy.apache.org/ (Last accessed on May 26th, 2022)
[7]https://downloads.dbpedia.org/repo/dbpedia/mappings/mappingbased-objects/2021.03.01/mappingbased-objects_lang=en.ttl.bz2 (Last accessed on May 26th, 2022)
[8]https://downloads.dbpedia.org/repo/dbpedia/generic/infobox-properties/2021.03.01/infobox-properties_lang=en.ttl.bz2 (Last accessed on May 26th, 2022)

columns (subject, predicate, object), in which each tuple corresponds to an RDF statement. For our proof-of-concept, using the Data Preprocessor component available in DCoEPinKB, we then partitioned and converted each Statement Table into 200 Parquet files.

Data partition was adopted to simulate a distributed environment in which fragments of the datasets reside on different machines. The initial phase of the experiments tested several configurations, including different numbers of partitions and partitioning by one or several columns of the data, which forces the Parquet files to be organized into subdirectories. However, the experiments did not contemplate the evaluation of these different configurations but rather adopted the configuration which obtained the best performance in these exploratory experiments.

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, ten entity pairs from each domain. Table III 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.

Table III: 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**

Configuration parameters. The following parameters were used:

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

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 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 criterion is combined with entity similarity because one can assume that high degree nodes influence the path search process with potentially very unspecific information [Moore et al. 2012].

Set of ignored properties: about ten properties were ignored during the exploration of the knowledge base because they describe relationships between entities that are irrelevant to our analysis.

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

Maximum number of paths: set to 50 because this value suffices to explore the connectivity between the entities [Cheng et al. 2014; Fang et al. 2011; Hulpuş et al. 2015; Pirrò 2015].

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, $nDCG_k = \frac{DCG_k}{IDCG_k}$.

Ground Truth. We needed a ground truth (or ideal ranking) to evaluate the ranking quality of the path search strategies for different expansion limit values using $nDCG_k$, for $k = 1$ to 50. We did not adopt the ranked lists of paths in Herrera et al. [2017] as our ground truth because the subsets of DBpedia we used were different from those in Herrera et al. [2017] – DBpedia indeed constantly changes. A new ground truth was then constructed as follows.

Let π_i, for $i = 1...6$, be one of the six path search strategies listed in Table I. Let (e_j, f_j), for $j = 1...20$, be one of the pairs of entities in Table III. Let D_m, for $m = 1...2$, be one of the DBpedia subsets adopted, i.e., D_1 =DBpedia21M and D_2 =DBpedia45M. Let λ_n, for $n = 1...6$, be one of the expansion limits to be tested, i.e., $1, 10, 15, 20, 25$ and 50%.

We created a separate ground truth path ranking $GT_{i,j,m}$ for π_i and (e_j, f_j) over D_m as follows:

(1) Execute π_i for (e_j, f_j) over D_m with two different expansion limits, $\lambda = 25$ and $\lambda = 50\%$, obtaining two sets of paths.

(2) Combine the two sets obtained in the previous step into one set of paths $P_{i,j,m}$.

(3) Rank the paths in $P_{i,j,m}$ using the path ranking measure adopted in π_i, and retain the top-50 ranked paths, which is then $GT_{i,j,m}$.

This process resulted in a dataset[9] that contains a total of 240 ranked lists, $GT_{i,j,m}$, with 50 relationship paths each (240 = 6 path search strategies × 20 pairs of entities × 2 DBpedia subsets).

The reasons for Steps (1) and (2) above are as follows. First, the execution of π_i with expansion limit 25 produces all paths that the executions of π_i with expansion limits smaller than 25 produce. Hence, π_i needs not to be executed for expansion limits smaller than 25. Second, the execution of π_i with expansion limit 25 may produce results different from the execution with 50% since the list of the top 25 nodes may be larger or smaller than the list of the top 50% nodes. Hence, the sets of lists produced by π_i with expansion limits 25 and 50% must be combined (Step (2)). Third, the execution of π_i without any expansion limit proved infeasible, since it generated too many paths in many cases. Hence, we opted to generate a large yet feasible set of paths containing all sets of paths generated using each of the expansion limits considered.

This strategy for constructing a ground truth suffices for the group of experiments in Section 5.2, which evaluate the impact of limiting the expansion limit on the ranking quality.

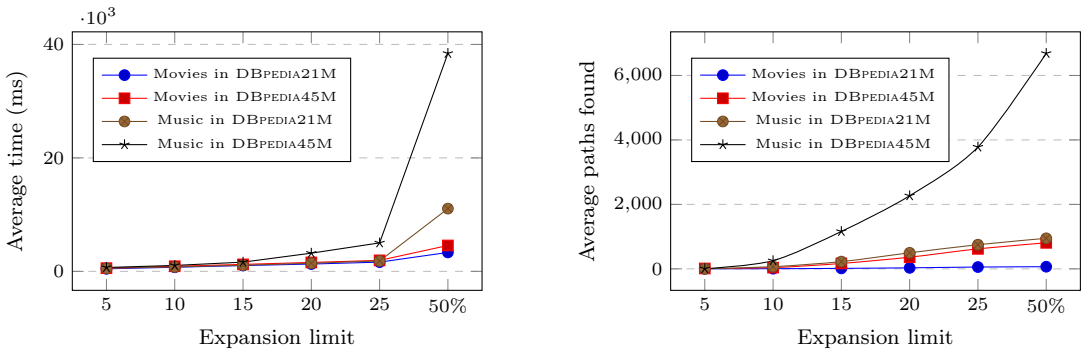
[9]The dataset containing the ground truth files is available at https://figshare.com/articles/dataset/Ground_Truth_for_Entity_Relatedness_Problem_over_DBpedia_datasets/15181086 (Last accessed on May 26th, 2022)

5. EVALUATION

5.1 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.

Figure 3a shows the average execution times of the J&E strategy, which achieved the best performance for finding relevant relationship paths in Herrera [2017], and the best average execution time in Jiménez et al. [2021]). For each pair of entities in each dataset, we searched 6 times for 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 five executions. We refer the reader to Jiménez [2021] for the full set of results.



(a) Average execution time

(b) Average paths found

Fig. 3: 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.

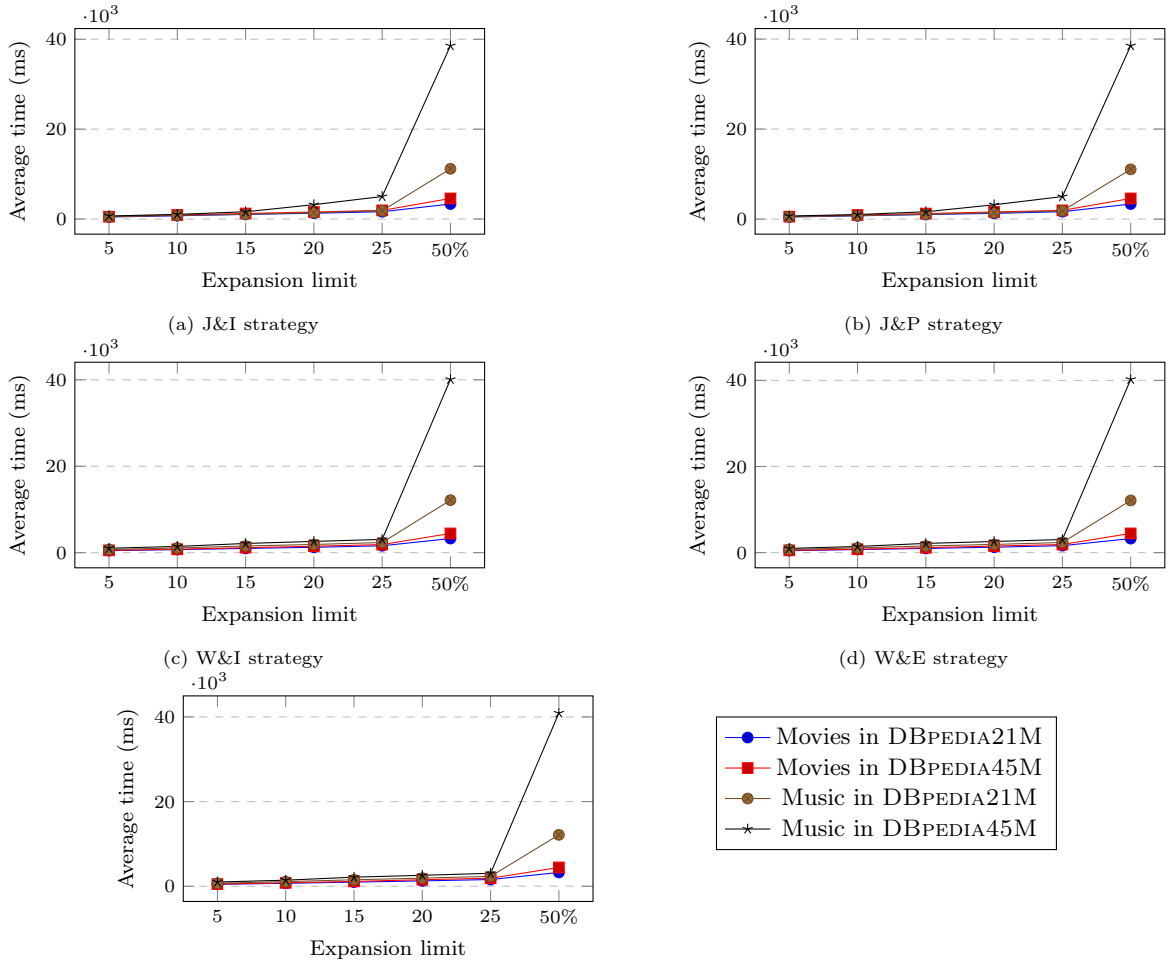
As the execution time depends on the number of paths found, we also show in Figure 3b the average number of paths found for different expansion limits using the J&E strategy. The number of paths found is closely related to the degree of the entities involved. Hence, by expanding the 50% most similar 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 3b, which implies that the running time also increases.

Figure 4 presents the average execution times of the other path search strategies.

5.2 Experiment 2 - Impact of the Expansion Limit on the Ranking Accuracy

The second set of experiments evaluated the ranking accuracy of the path search strategies, for different expansion limit values, as compared to the ground truth, using $nDCG_k$, for $k = 1$ to 50.

More precisely, as in the construction of the ground truth, let π_i, for $i = 1...6$, be one of the six path search strategies listed in Table I. Let (e_j, f_j), for $j = 1...20$, be one of the pairs of entities in Table III. Let D_m, for $m = 1...2$, be one of the DBpedia subsets adopted, i.e., D_1 =DBpedia21M



(a) J&I strategy

(b) J&P strategy

(c) W&I strategy

(d) W&E strategy

(e) W&P strategy

Fig. 4: Average execution time over all entity pairs in each domain and dataset for different strategies varying the expansion limit

and D_2 =DBpedia45M. Let λ_n, for $n = 1...6$, be one of the expansion limits to be tested, i.e., $1, 10, 15, 20, 25$ and 50%.

For each path search strategy π_i, for each expansion limit λ_n, for each dataset D_m, the experiments in this section will:

(1) for each pair (e_j, f_j),
   (a) Compute a ranked list of paths $RL_{i,j,m,n}$.
   (b) For $k = 1$ to 50, incrementally by 1, compute the $nDCG_{i,j,m,n}@k$ between $RL_{i,j,m,n}$ and $GT_{i,j,m}$.

(2) Compute the average $A_{i,m,n,k} = \frac{1}{10}(\sum_{j=1}^{10} nDCG_{i,j,m,n}@k)$, for the music domain (pairs $j = 1...10$), over D_m.

(3) Repeat the previous step for the movies domain (pairs $j = 11...20$).

In what follows, let "π_i with top-λ_n" indicate the strategy π_i expanding the top-λ_n most similar adjacent nodes.

Experiment 2.1 - Ranking Accuracy for the Path Search Strategies using the Jaccard Index

Figure 5 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 5a), and above 0.86 using DBpedia45M (Figure 5b), 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 5c), and above 0.84 using DBpedia45M (Figure 5d).

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 the two 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 reaches a low 0.44.

Figure 6 shows the average $nDCG_k$ for the J&I strategy for the movies and music domains in DBpedia21M and DBpedia45M. Using the PF-ITF measure for ranking the relationship paths makes it 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 6a), and above 0.73 using DBpedia45M (Figure 6b). Figure 6a 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 6b 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 6c).

Furthermore, by using the PMI measure to rank the relationship paths, it is possible to achieve acceptable performance in the movies domain in DBpedia21M using J&P with the top-25. Indeed, using DBpedia21M, the average $nDCG_k$ for J&P with top-25 was above 0.67 (Figure 7a).

Experiment 2.2 - Ranking Accuracy for the Path Search Strategies using the Wikipedia Link-based Measure

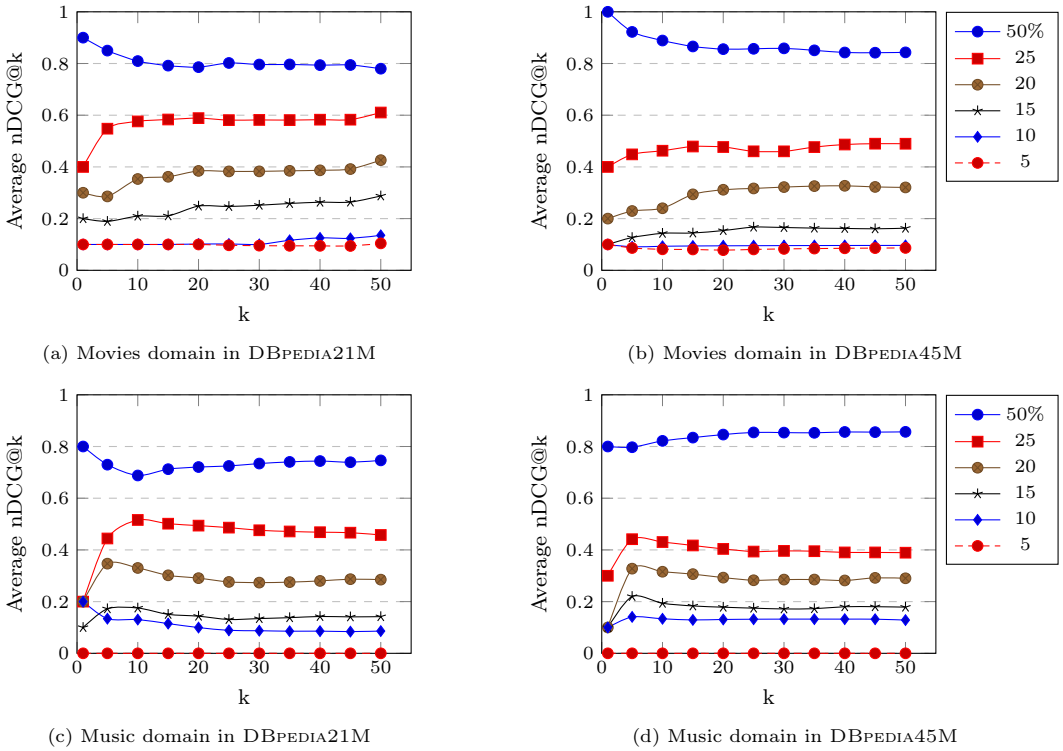
Very briefly, Figure 8 shows the results for the W&E strategy, which had good performance. Similar to the J&I with the top-25, in the movies domain, W&I with the top-25 achieves a very good performance. Indeed, the average $nDCG_k$ for W&I with top-25 was above 0.78 using DBpedia21M (Figure 9a), and above 0.77 using DBpedia45M (Figure 9b). Again, by using the PMI measure to rank the relationship paths, it is possible to achieve acceptable performance in the movies domain in DBpedia21M using W&P with the top-25. Indeed, using DBpedia21M, the average $nDCG_k$ for W&P with top-25 was above 0.69 (Figure 10a).

5.3 Summary of the Experiments

To summarize, the path search strategies that use an expansion limit of 50% (the top-50% most similar adjacent nodes) had very high average execution times and achieved the best ranking quality, in almost all cases. However, the path search strategies that use an expansion limit equal to 25 (the top-25 most similar adjacent nodes) had significantly lower execution times, comparable to those of strategies based on lower expansion limits, and yet achieved acceptable ranking quality, in some cases, even comparable to or better than the ranking quality of strategies with an expansion limit of 50%.

6. RELATED WORK

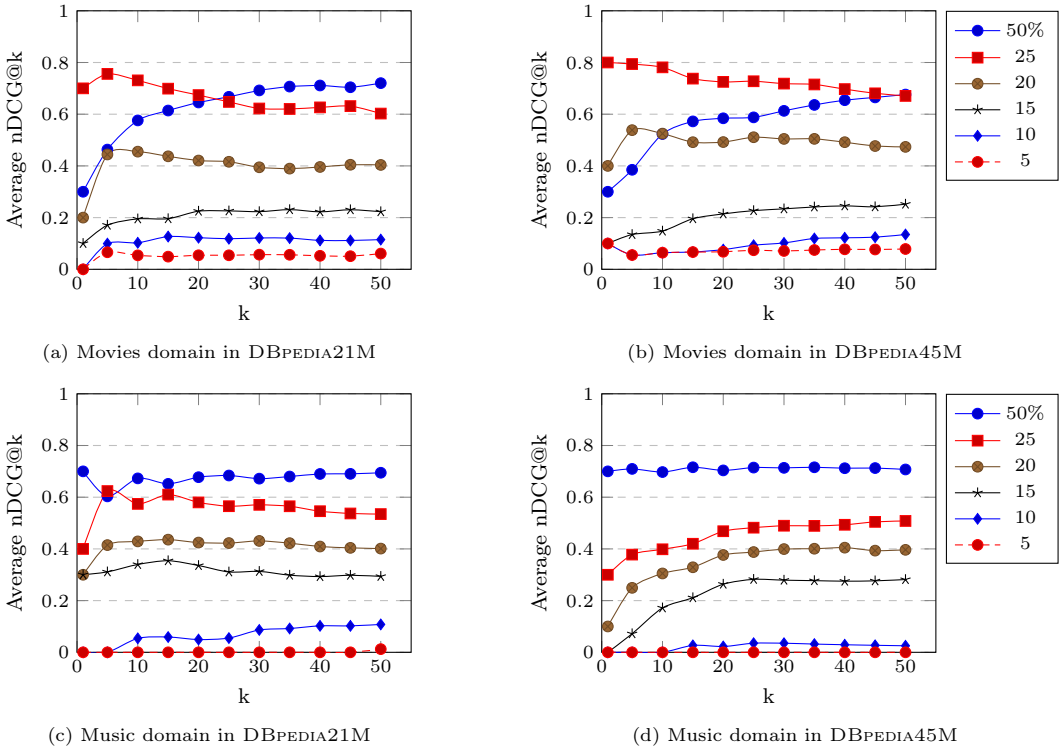
This section is organized in four subsections covering the topics related to the article: (1) Entity relationship discovery and ranking in Knowledge Bases; (2) Processing large RDF datasets in distributed environments; (3) Similarity-based operations on distributed query processing systems; and (4) Benchmarks. It ends with a summary of the contributions of the article.



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

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

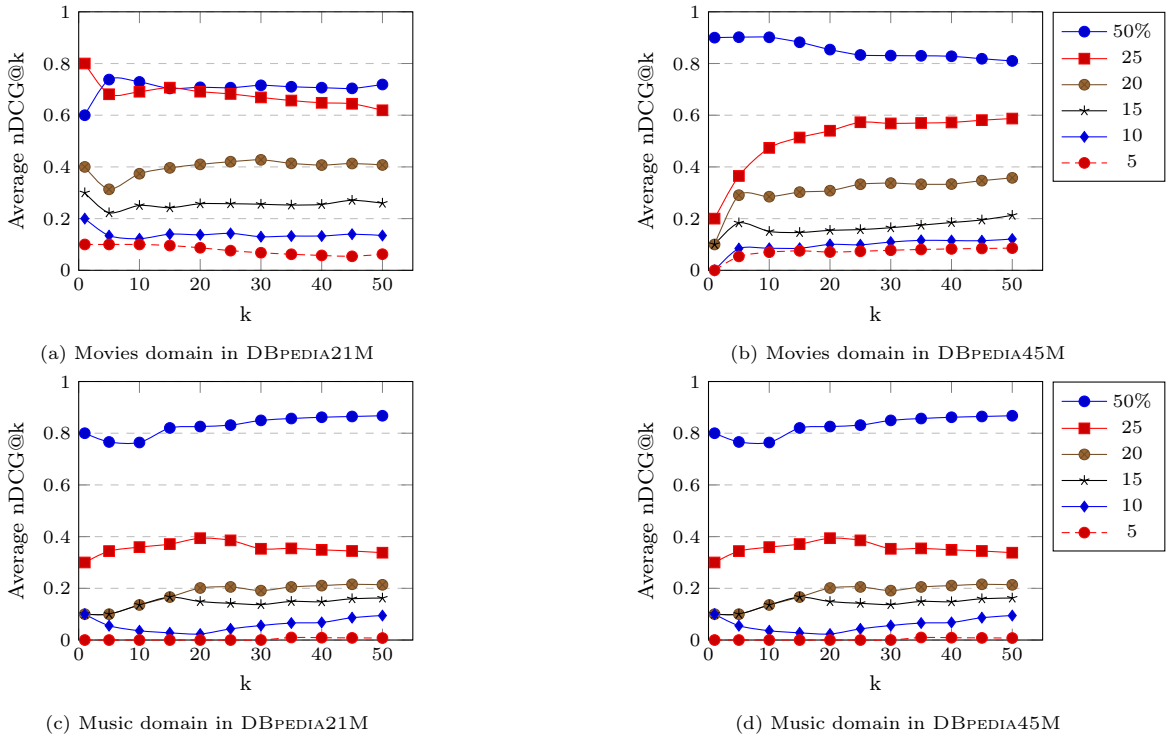
Fig. 5: Average nDCG@k for the J&E strategy varying the expansion limit



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

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

Fig. 6: Average nDCG@k for the J&I strategy varying the expansion limit



(a) Movies domain in DBpedia21M

(b) Movies domain in DBpedia45M

(c) Music domain in DBpedia21M

(d) Music domain in DBpedia45M

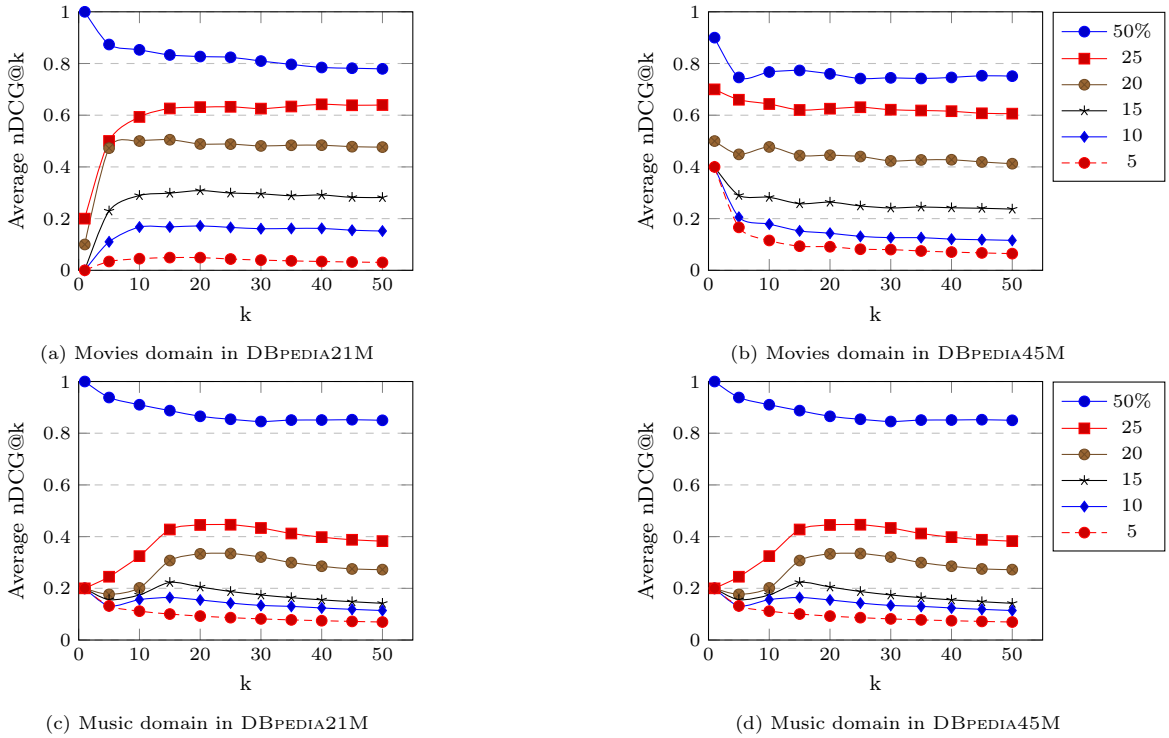
Fig. 7: Average nDCG@k over the movies and music domains for the J&P strategy varying the expansion limit

6.1 Entity Relationship Discovery and Ranking in Knowledge Bases

Several strategies and tools have been proposed to discover the semantic associations between a pair of entities in a knowledge base. Some approaches identify all possible relationships between two entities, using SPARQL queries to retrieve paths up to a certain length [Heim et al. 2009; Pirrò 2015; Herrera et al. 2016], and then rank the results based on 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 implemented in Python that takes a pair of entities in a given knowledge base as input and produces a ranked list of *relationship explanations*. The authors consider a relationship explanation a constrained graph pattern and its associated graph instances derivable from the underlying knowledge base. REX implements different algorithms for finding the relationship explanations, adapted from solutions proposed for the keyword search problem in databases. The PathEnumBasic algorithm is based on the backward expansion search introduced in BANKS [Bhalotia et al. 2002] and generates partial paths from input entities concurrently, with shorter paths being generated first. The second path enumeration algorithm PathEnumPrioritized is a direct adaption of the bidirectional search by Kacholia et al. [2005], an improved version of BANKS. Instead of always expanding the shortest partial paths, the degree of the nodes is used as an



(a) Movies domain in DBpedia21M

(b) Movies domain in DBpedia45M

(c) Music domain in DBpedia21M

(d) Music domain in DBpedia45M

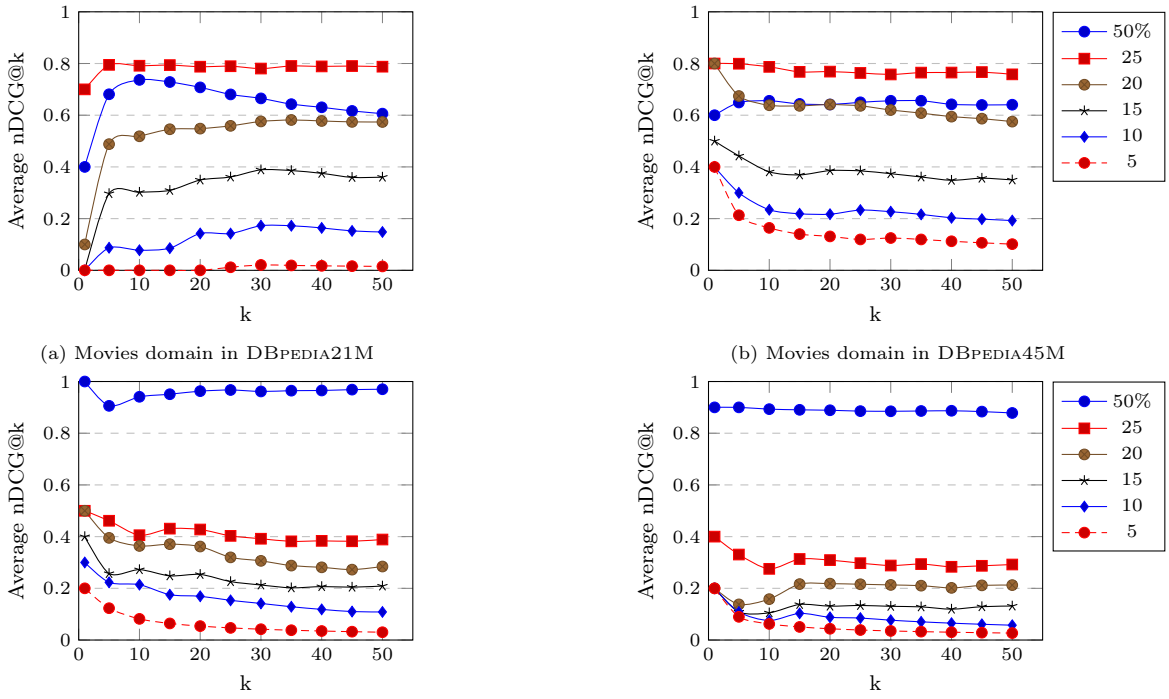
Fig. 8: Average nDCG@k over the movies and music domains for the W&E strategy varying the expansion limit

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 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 Cheng et al. [2014] explores a knowledge base searching for associations and provides a list of the top-k clusters, which are labeled with an association pattern that gives users a conceptual summary of the associations in the cluster. The clusters are obtained by formulating and solving a data mining problem, and then the top-k ones are found by formulating and solving an optimization problem. Explass integrates patterns with facet values, which are classes of entities and relationships that appear in associations, and that can be used by users to refine the search and better explore associations. The authors compared Explass with two existing related approaches by conducting a user study and tested the statistical significance of the results. Cheng 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]



(a) Movies domain in DBpedia21M

(b) Movies domain in DBpedia45M

(c) Music domain in DBpedia21M

(d) Music domain in DBpedia45M

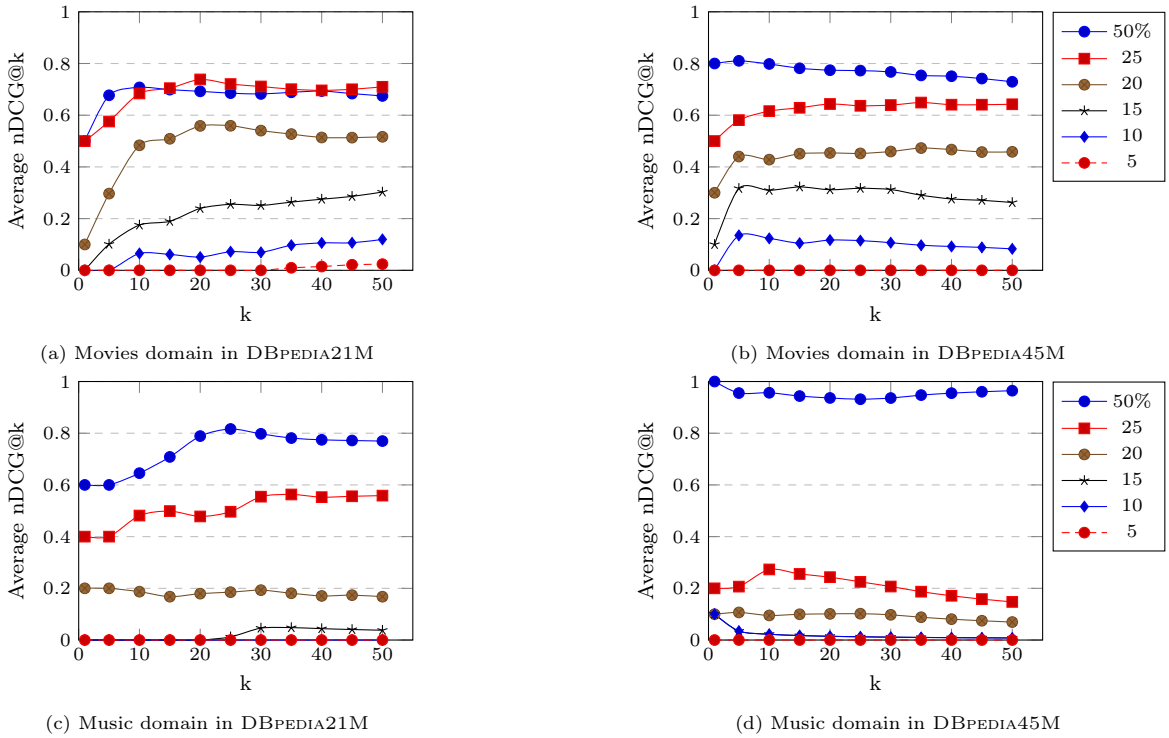
Fig. 9: Average nDCG@k over the movies and music domains for the W&I strategy varying the expansion limit

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.

RECAP Pirrò [2015] is a framework to generate different types of relatedness explanations between entities, possibly combining information from multiple knowledge bases. RECAP goes beyond related approaches such as REX and Explass. 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 Herrera et al. [2016] is a tool 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 did experiments to compare DBpedia Profiler with RECAP, and the results showed that DBpedia Profiler outperforms RECAP in terms of performance and usability.

Herrera [2017] introduced a generic search strategy based on the backward search heuristic proposed in 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.



(a) Movies domain in DBpedia21M

(b) Movies domain in DBpedia45M

(c) Music domain in DBpedia21M

(d) Music domain in DBpedia45M

Fig. 10: Average nDCG@k over the movies and music domains for the W&P strategy varying the expansion limit

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 in 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, and a tool with a graphical user interface that facilitates evaluating these strategies. The author also identified opportunities for future work, such as developing a framework for the entity relatedness problem, considering as points of flexibility the similarity measure between entities and the path-ranking measure to identify relevant paths. This article aimed at filling this gap.

6.2 Processing Large RDF Datasets in Distributed Environments

Many distributed RDF systems have been introduced to overcome the problem of indexing and querying large RDF datasets [Farhan Husain et al. 2009; Rohloff and Schantz 2010; Husain et al. 2011; Huang et al. 2011; Przyjaciel-Zablocki et al. 2012; De Virgilio and Maccioni 2014; Schätzle et al. 2016; Schätzle et al. 2016; Abdelaziz et al. 2017; Ragab et al. 2019; Ragab et al. 2020; Ragab et al. 2021]. These systems are generally built on top of distributed data processing frameworks, such as MapReduce [Dean and Ghemawat 2008] or Apache Spark [Zaharia et al. 2010], partition the RDF graphs among multiple machines to handle big datasets, and parallelize query execution to reduce query runtime.

SHARD is an open-source, horizontally scalable triplestore system proposed in Rohloff and Schantz [2010] and built using the Hadoop implementation of MapReduce. SHARD persists graph data as RDF triples and responds to queries over this data in the SPARQL query language. Husain et al. [2011] presented HadoopRDF, an scalable and fault-tolerant framework based on Hadoop that supports

data-intensive query processing. The authors proposed a storage schema to store RDF data in HDFS[10] and a new greedy algorithm that overcomes the limitations of the algorithm previously introduced by Farhan Husain et al. [2009]. The algorithm uses the MapReduce programming model and produces a query plan whose cost, i.e., the number of Hadoop jobs executed to solve the query, is bounded by the logarithm of the total number of variables in the given SPARQL query. Huang et al. [2011] presented a scale-out architecture for RDF data management using the Hadoop framework. The authors described data partitioning and placement techniques that reduced the amount of network communication at query time and provided an algorithm for automatically decomposing SPARQL queries into parallelizable Hadoop jobs. RDFPath is a declarative path query language for RDF proposed in Przyjaciel-Zablocki et al. [2012] that automatically transforms declarative path queries into MapReduce jobs and supports the exploration of graph properties such as shortest paths between two nodes in an RDF graph. De Virgilio and Maccioni [2014] presented a distributed approach to keyword search query over large RDF datasets that exploits the MapReduce paradigm by switching the problem from graph-parallel to data-parallel processing. This paradigm shift is necessary because MapReduce is an effective data-parallel paradigm for computing algorithms that require reading the data only once and, for this reason, it is not efficient to perform join-intensive tasks typical of graph algorithms.

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

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

The SPARKSQL RDF Processing Benchmarking[11] is a systematic benchmarking project on the performance of Spark SQL for processing vast RDF datasets. In the first phase of this project, Ragab et al. [2019] presented an analysis of the execution time of Spark SQL for answering SPARQL queries over RDF repositories on a centralized single-machine configuration. The authors conducted experiments on datasets with 100K, 1M, and 10M triples and evaluated the impact of using alternative relational schemas for RDF (i.e., ST, VT, and PT), various storage backends (i.e., PostgreSQL, Hive, and HDFS) and different data formats (e.g., CSV, Avro, Parquet and ORC). In the second phase of the project [Ragab et al. 2020], the experiments include a larger dataset than before in a distributed environment. The authors evaluated the impact of using different RDF-based partitioning techniques (i.e., subject-based, predicate-based, and horizontal-based partitioning). Ragab et al. [2021] extended the previous experiments with new proposed RDF relational schema representations:

[10]Hadoop Distributed File System
[11]https://datasystemsgrouput.github.io/SPARKSQLRDFBenchmarking/ (Last accessed on May 26th, 2022)

Extended Vertically Partitioned Tables (ExtVP) and Wide Property Tables (WPT).

6.3 Similarity-based operations on Distributed Query Processing Systems

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

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

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

6.4 Benchmarks

Herrera et al. [2017] 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)[12] and last.fm [13] to generate specialized relationship path rankings between entity pairs in the movies and music domains, respectively. For each domain, the dataset contains 20 pairs of entities, each with a ranked list with 50 relationship paths based on information about their entities found in IMDb and last.fm, and on information about their properties, computed from DBpedia.

However, the authors did not specify which version of the DBpedia they relied on to generate the ground truth, making it difficult to use their test dataset in our experiments, considering that the content of the DBpedia has changed since then. Section 5 of this article circumvented this problem by introducing a strategy to construct ground truths for any given version of the DBpedia.

6.5 Summary

As an extended version of [Guillot Jiménez et al. 2021], the article focuses on the problem of evaluating the performance of different path search strategies, in terms of execution time, which is lacking in the related work covered in Section 6.1. To facilitate the evaluation, the article proposes a framework, built on top of Apache Spark, called DCoEPinKB.

Although the experiments used a standalone implementation, the framework can be retargeted to a fully distributed environment. By contrast, the CoEPinKB framework [Jiménez et al. 2021]

[12]https://www.imdb.com/ (Last accessed on May 26th, 2022)
[13]https://www.last.fm/ (Last accessed on May 26th, 2022)

implements path search strategies using a multi-thread approach. Table IV 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 preprocesses any RDF dataset and transforms it into Parquet files. Regarding the architecture, DCoEPinKB is the only approach that is prepared to address the entity relatedness problem in a distributed manner, in line with systems designed to process large RDF datasets, summarized in Sections 6.2 and 6.3.

Finally, as pointed out in Section 6.4, to facilitate the experiments, the article also introduces a strategy to construct ground truths for any given version of DBpedia.

Table IV: 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.

7. CONCLUSIONS

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

The main contribution of this article is the proposal and implementation of a framework, called DCoEPinKB, to empirically evaluate path search strategies that combine entity similarity and path ranking measures. The framework has some hot spots for developers to easily add new entity similarity and relationship path ranking measures to generate new path search strategies and work with different knowledge bases.

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

Finally, a third contribution is an extensive experimental evaluation on the music and movies domains over real data, collected from DBpedia, to assess the correctness and the performance of a family of path search strategies using the DCoEPinKB framework. The experiments showed that reducing the expansion limit when finding the paths between entities with a high degree can improve the execution time, as expected, and maintain acceptable ranking quality.

As future work, we plan to deploy and test a fully distributed implementation of the framework along the lines discussed in Section 3 and investigate effective optimization techniques in Spark to reduce the execution time for path search strategies. We also plan to explore effective optimization techniques in Spark to reduce the execution time for path search strategies.

REFERENCES

Abdelaziz, I., Harbi, R., Khayyat, Z., and Kalnis, P. A survey and experimental comparison of distributed SPARQL engines for very large RDF data. *Proceedings of the VLDB Endowment* 10 (13): 2049–2060, 2017.

Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., and Sudarshan, S. Keyword searching and browsing in databases using BANKS. In *Proceedings 18th International Conference on Data Engineering*. IEEE Computer Society, Los Alamitos, CA, USA, pp. 431–440, 2002.

Cheng, G., Liu, D., and Qu, Y. Fast Algorithms for Semantic Association Search and Pattern Mining. *IEEE Transactions on Knowledge and Data Engineering* 33 (4): 1490–1502, 2021.

Cheng, G., Shao, F., and Qu, Y. An Empirical Evaluation of Techniques for Ranking Semantic Associations. *IEEE Transactions on Knowledge and Data Engineering* 29 (11): 2388–2401, 2017.

Cheng, G., Zhang, Y., and Qu, Y. Explass: Exploring Associations between Entities via Top-K Ontological Patterns and Facets. In *Proceedings of the 13th International Semantic Web Conference (ISWC 2014)*, P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. Knoblock, D. Vrandečić, P. Groth, N. Noy, K. Janowicz, and C. Goble (Eds.). Vol. 8797. Springer International Publishing, Cham, pp. 422–437, 2014.

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

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

De Vocht, L., Beecks, C., Verborgh, R., Mannens, E., Seidl, T., and Van de Walle, R. Effect of Heuristics on Serendipity in Path-Based Storytelling with Linked Data. In *Human Interface and the Management of Information: Information, Design and Interaction*. Vol. 9734. Springer International Publishing, Cham, pp. 238–251, 2016.

De Vocht, L., Coppens, S., Verborgh, R., Sande, M. V., Mannens, E., and de Walle, R. V. Discovering Meaningful Connections between Resources in the Web of Data. In *Proceedings of the 8th Workshop on Linked Data on the Web (LDOW 2013)*. CEUR-WS.org, Rio de Janeiro, 2013.

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

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

Farhan Husain, M., Doshi, P., Khan, L., and Thuraisingham, B. Storage and Retrieval of Large RDF Graph Using Hadoop and MapReduce. In *Cloud Computing*. Vol. 5931. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 680–686, 2009.

Guillot Jiménez, J., P. Paes Leme, L. A., Torres Izquierdo, Y., Batista Neves, A., and Casanova, M. A. A distributed framework to investigate the entity relatedness problem in large RDF knowledge bases. In *Anais do XXXVI Simpósio Brasileiro de Banco de Dados (SBBD 2021)*. Sociedade Brasileira de Computação - SBC, Rio de Janeiro, pp. 121–132, 2021.

Heim, P., Hellmann, S., Lehmann, J., Lohmann, S., and Stegemann, T. RelFinder: Revealing Relationships in RDF Knowledge Bases. In *Semantic Multimedia*. Vol. 5887. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 182–187, 2009.

Herrera, J. E. T. *On the Connectivity of Entity Pairs in Knowledge Bases*. Ph.D. thesis, Pontifícia Universidade Católica do Rio de Janeiro, 2017.

Herrera, J. E. T., Casanova, M. A., Nunes, B. P., Leme, L. A. P. P., and Lopes, G. R. An Entity Relatedness Test Dataset. In *Proc. of the 16th Int'l Semantic Web Conf. (ISWC'17)*. Vol. 10588 LNCS. Springer, Cham, Cham, pp. 193–201, 2017.

Herrera, J. E. T., Casanova, M. A., Nunes, B. P., Lopes, G. R., and Leme, L. 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)*. Springer-Verlag Berlin Heidelberg, Kobe/Japan, 2016.

Huang, J., Abadi, D. J., and Ren, K. Scalable SPARQL querying of large RDF graphs. *Proceedings of the VLDB Endowment* 4 (11): 1123–1134, 2011.

Hulpuş, I., Prangnawarat, N., and Hayes, C. Path-Based Semantic Relatedness on Linked Data and Its Use to Word and Entity Disambiguation. In *The Semantic Web - ISWC 2015*. Vol. 9366. Springer International Publishing, Cham, pp. 442–457, 2015.

Husain, M., McGlothlin, J., Masud, M. M., Khan, L., and Thuraisingham, B. M. Heuristics-Based Query Processing for Large RDF Graphs Using Cloud Computing. *IEEE Transactions on Knowledge and Data Engineering* 23 (9): 1312–1327, 2011.

Jaccard, P. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bull Soc Vaudoise Sci Nat* vol. 37, pp. 547–579, 1901.

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

Jiménez, J. G. *Strategies to Understand the Connectivity of Entity Pairs in Knowledge Bases*. Ph.D. thesis, Department of Informatics, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil, 2021.

Jiménez, J. G., Leme, L. A. P. P., and Casanova, M. A. CoEPinKB: A Framework to Understand the Connectivity of Entity Pairs in Knowledge Bases. In *Anais do XLVIII Seminário Integrado de Software e Hardware (SEMISH 2021)*. Sociedade Brasileira de Computação - SBC, Online, pp. 97–105, 2021.

Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., and Karambelkar, H. Bidirectional Expansion For Keyword Search on Graph Databases. In *Proceedings of the 31st international Conference on Very Large Data Bases (VLDB 2005)*. VLDB Endowment, Trondheim, Norway, 2005.

Kim, T., Li, W., Behm, A., Cetindil, I., Vernica, R., Borkar, V., Carey, M. J., and Li, C. Similarity query support in big data management systems. *Information Systems* vol. 88, pp. 101455, 2020.

Le, W., Li, F., Kementsietsidis, A., and Duan, S. Scalable Keyword Search on Large RDF Data. *IEEE Transactions on Knowledge and Data Engineering* 26 (11): 2774–2788, 2014.

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

Lehmann, J., Schüppel, J., and Auer, S. Discovering Unknown Connections – the DBpedia Relationship Finder. In *Proceedings of the 1st Conference on Social Semantic Web (CSSW 2007)*. Gesellschaft für Informatik e. V., Bonn, pp. 99–109, 2007.

Milne, D. and Witten, I. H. An Effective, Low-Cost Measure of Semantic Relatedness Obtained from Wikipedia Links. In *Proc. AAAI 2008 Workshop on Wikipedia and Artificial Intelligence*. AAAI Press, Chicago, pp. 25–30, 2008.

Moore, J. L., Steinke, F., and Tresp, V. A Novel Metric for Information Retrieval in Semantic Networks. In *ESWC*. Vol. 7117. Springer, Berlin, Heidelberg, pp. 65–79, 2012.

Pereira Nunes, B., Herrera, J., Taibi, D., Lopes, G. R., Casanova, M. A., and Dietze, S. 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)*, V. Presutti, E. Blomqvist, R. Troncy, H. Sack, I. Papadakis, and A. Tordai (Eds.). Springer, Anissaras, Crete, Greece, pp. 461–466, 2014.

Pirrò, G. Explaining and Suggesting Relatedness in Knowledge Graphs. In *Proceedings of the 14th International Conference on The Semantic Web (ISWC 2015)*. Vol. 9366. Springer International Publishing, Cham, pp. 622–639, 2015.

Przyjaciel-Zablocki, M., Schätzle, A., Hornung, T., and Lausen, G. RDFPath: Path Query Processing on Large RDF Graphs with MapReduce. In *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 50–64, 2012.

Ragab, M., Tommasini, R., Awaysheh, F. M., and Ramos, J. C. An In-depth Investigation of Large-scale RDF Relational Schema Optimizations Using Spark-SQL. In *Proceedings of the 23rd International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP 2021)*. CEUR-WS.org, Nicosia, Cyprus, pp. 71–80, 2021.

Ragab, M., Tommasini, R., Eyvazov, S., and Sakr, S. Towards making sense of Spark-SQL performance for processing vast distributed RDF datasets. In *Proceedings of The International Workshop on Semantic Big Data*. ACM, Portland, Oregon, pp. 1–6, 2020.

Ragab, M., Tommasini, R., and Sakr, S. Benchmarking Spark-SQL under Alliterative RDF Relational Storage Backends. In *Proceedings of the QuWeDa 2019: 3rd Workshop on Querying and Benchmarking the Web of Data co-located with 18th International Semantic Web Conference (ISWC 2019)*. CEUR-WS.org, Auckland, New Zealand, pp. 67–82, 2019.

Rohloff, K. and Schantz, R. E. High-performance, massively scalable distributed systems using the MapReduce software framework. In *Programming Support Innovations for Emerging Distributed Applications on - PSI EtA'10*. ACM, Reno/USA, pp. 1–5, 2010.

Schätzle, A., Przyjaciel-Zablocki, M., Berberich, T., and Lausen, G. S2X: Graph-Parallel Querying of RDF with GraphX. In *Biomedical Data Management and Graph Online Querying*. Vol. 9579. Springer, Cham, pp. 155–168, 2016.

Schätzle, A., Przyjaciel-Zablocki, M., Hornung, T., and Lausen, G. PigSPARQL: A SPARQL Query Processing Baseline for Big Data. In *Proceedings of the 12th International Semantic Web Conference (ISWC 2013)*. CEUR-WS.org, Sydney, Australia, pp. 241–244, 2013.

Schätzle, A., Przyjaciel-Zablocki, M., Skilevic, S., and Lausen, G. S2RDF: RDF querying with SPARQL on spark. *Proceedings of the VLDB Endowment* 9 (10): 804–815, 2016.

Sun, J., Shang, Z., Li, G., Deng, D., and Bao, Z. Dima: a distributed in-memory similarity-based query processing system. *Proceedings of the VLDB Endowment* 10 (12): 1925–1928, 2017.

Sun, J., Shang, Z., Li, G., Deng, D., and Bao, Z. Balance-aware distributed string similarity-based query processing system. *Proceedings of the VLDB Endowment* 12 (9): 961–974, 2019.

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