Explaining and Suggesting Relatedness in Knowledge Graphs*

Giuseppe Pirrò

Institute for High Performance Computing and Networking, Italian National Research Council (ICAR-CNR), Rende (CS), Italy
pirro@icar.cnr.it

Abstract. Knowledge graphs (KGs) are a key ingredient for searching, browsing and knowledge discovery activities. Motivated by the need to harness knowledge available in a variety of KGs, we face the following two problems. First, given a pair of entities defined in some KG, find an explanation of their relatedness. We formalize the notion of relatedness explanation and introduce different criteria to build explanations based on information-theory, diversity and their combinations. Second, given a pair of entities, find other (pairs of) entities sharing a similar relatedness perspective. We describe an implementation of our ideas in a tool, called RECAP, which is based on RDF and SPARQL. We provide an evaluation of RECAP and a comparison with related systems on real-world data.

1 Introduction

Knowledge Graphs (KGs) maintaining structured data about entities are becoming a common support for browsing, searching and knowledge discovery activities. Search engines like Google, Yahoo! and Bing complement search results with facts about entities in their KGs. An even large number and variety of KGs, based on the Resource Description Framework (RDF) data format, stem from the Linked Open Data project [7]. Fig. 1 (a) shows information provided by the Google KG when giving the entity F. Lang as input; it reports some facts about the director along with relationships with other entities. Fig. 1 (b) and Fig. 1 (c) show information, encoded in RDF, about F. Lang taken from DBpedia and LinkedMDB, respectively. Note that the Google KG suggests entities like T. von Harbou as related to F. Lang with a short comment saying that T. von Harbou was F. Lang's former spouse. However, what is the mechanism behind this suggestion? What is the relationship between F. Lang and other entities like H. Hitchcock? KGs like DBpedia and LinkedMDB also fail short when it comes to both explain the relatedness between an arbitrary pair of entities and suggest related entities. We contend that the usage of a standard data format (i.e., RDF) and the availability of a standard querying infrastructure (i.e., SPARQL endpoints) open new perspectives toward explaining relatedness and querying KGs by using pairs of entities as input.

* Part of this work was done while the author was working at the WeST institute, University of Koblenz-Landau, Germany. This work was partially supported by the EU Framework Programme for Research and Innovation under grant agreement no. 611242 (SENSE4US) and by the Cyber Security Technological District financed by the Italian Ministry of Education, University and Research.

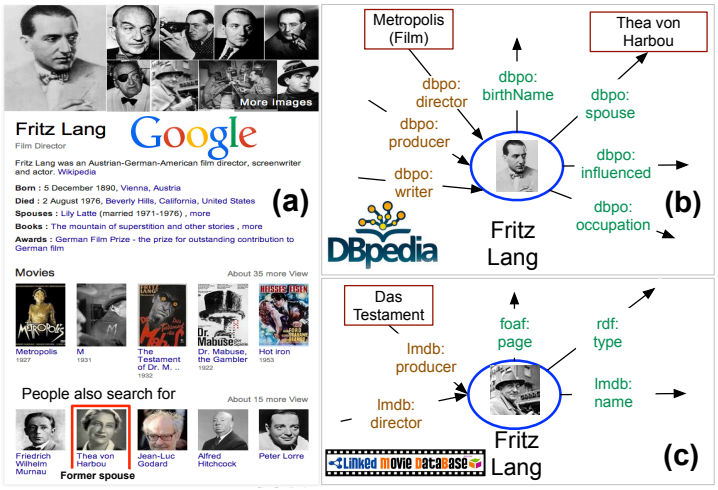

Fig. 1. F. Lang in the Google KG (a), DBpedia (b), and LinkedMDB (c).

The first problem that we face concerns how to build relatedness explanations. This has applications in several areas including: terrorist networks, to uncover the connections between two suspected terrorists [19]; co-author networks, to discover interlinks between researchers [5]; generic exploratory search. The need for relatedness explanations also emerged in the context of the SENSE4US FP7 project[1], which aims at creating a toolkit to support information gathering, analysis and policy modeling. Here, relatedness explanations are useful to investigate and show to the user topic connectivity[2], thus enabling to find out previously unknown information that is of relevance, understand how it is of relevance, and navigate it.

Although the problem of finding connectivity structures between entities has been studied (e.g., [3, 18]), existing approaches do not offer comprehensive mechanisms for building different types of relatedness explanations and controlling the amount of information to be included. Moreover, these approaches miss the possibility to query KGs.

The second problem that we tackle concerns querying KGs. KGs behind search engines (e.g., Google) provide limited querying capabilities, typically accepting one entity as input. KGs based on RDF (e.g., DBpedia) provide rich querying capabilities but require familiarity with languages like SPARQL [6] and the underlying data/schema [9]. Our strategy recalls the query by example approach; given a pair of entities as input, we leverage their relatedness explanation to learn a query pattern, which is used to identify other (pairs of) related entities. Our approach goes suggestionbeyond existing entity mechanisms mainly based on the syntactic analysis of query logs and pages [12]. We now provide an example about the two main challenges faced in this paper, that is, *how to build relatedness explanations* and *how to query KGs by giving entities as input*.

[1] `http://www.sense4us.eu`

[2] A module of the SENSE4US toolkit extracts topics from policy documents

1.1 Overview of the Approach

Syd is fond of science-fiction films; he has heard about two German directors named Fritz Lang and Thea von Harbou and is interested in their relatedness.

By giving F. Lang and T. von Harbou as input to RECAP, the tool implementing our framework, Syd gets the explanation in Fig. 2 (a). This explanation is more informative than the short comment (i.e., former spouse) provided by the Google KG and *combines* information from Freebase and DBpedia. The explanation includes the top-20 most informative paths (out of 240) at max. distance 2; informativeness is defined in terms of edge labels occurrences. RECAP allows to generate different types of explanations (Fig. 2 (b)) and also provides information about nodes/edges (Fig. 2 (c)).

RECAP goes beyond related approaches (e.g., REX [4], Explass [2]) that provide visual information about connectivity as it allows to build different types of explanations (e.g., graphs, sets of paths), thus controlling the amount of information visualized. RECAP has the advantage of not requiring any data preprocessing; information is obtained by querying (remote) SPARQL endpoints. Moreover, RECAP can combine information from multiple KGs. In the previous example, the combination of Freebase and DBpedia allowed to discover an additional episode of *Die Nibelungen* series (missing in DBpedia), that is, *Kriemhild's Revenge*, co-written by F. Lang and T. von Harbou. Last but not least, RECAP also allows to query KGs by using pairs of entities as input.

Given a pair of entities, RECAP finds other (pairs of) related entities by learning a SPARQL query from their relatedness explanation. By continuing our example, suppose that *Syd gives the pair (F. Lang, T. von Harbou) as input to RECAP with the aim to discover other entities*. Fig 3 (b) shows one possible explanation that RECAP uses to learn a SPARQL query. The explanation merges paths conforming to the pattern in Fig 3 (a). Fig 3 (c) abstracts the explanation by replacing nodes with variables.

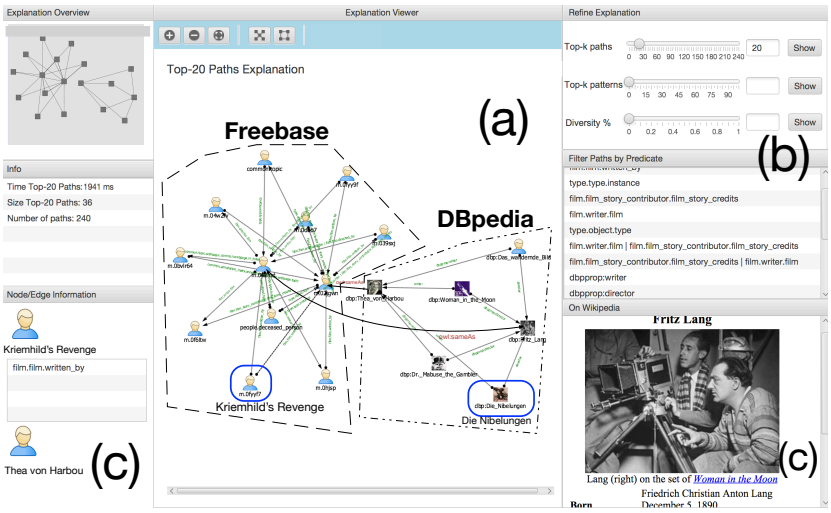

Fig. 2. The explanation perspective of the RECAP tool.

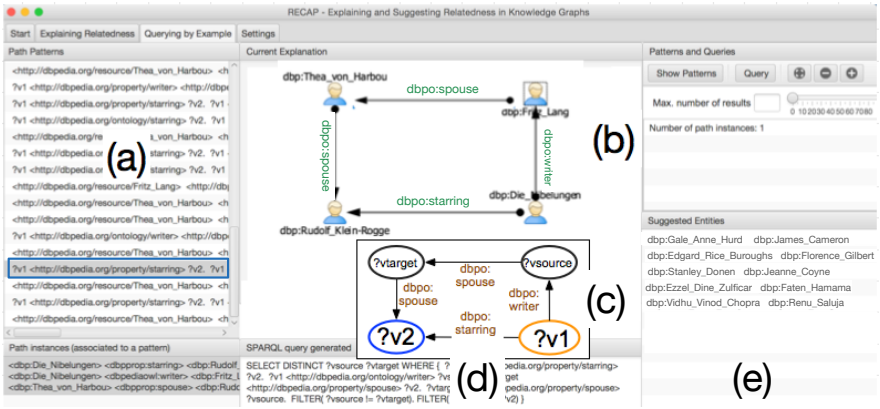

Fig. 3. The querying perspective of RECAP. Path patterns (a), explanation (b), query pattern (c), SPARQL query (d), and suggested entities (e) ranked by popularity (PageRank [16] in this case).

The SPARQL query generated (shown in Fig 3 (d)) allows to find pairs of related entities that can be locally ranked. The top-5 pairs of entities found by RECAP, and ranked by their popularity, are show in Fig. 3 (e). As an example, for the pair (Gale Ann Hurd, James Cameron) we can reconstruct a similar pattern as that shown in Fig. 3 (b): J. Cameron was married with G. A. Hurd, he wrote the movie The Terminator where L. Hamilton (also married to J. Cameron) starred.

1.2 Related Work

There is solid body of work about *(i)* finding structures (e.g., paths, subgraphs) connecting entities [3, 18, 11, 8, 14, 11]; *(ii)* learning relationships between entities; *(iii)* discovering and/or visualizing connectivity information between entities [8, 4, 2]. Differently from *(i)*, RECAP focuses on the problem of providing concise explanations by leveraging path informativeness and/or a diversity criterion. As for *(ii)*, systems like PATTY [15] mainly focus on learning semantic relationships. RECAP has a different departure point; it explains relatedness in the form of graphs that can be dynamically configured to include the desired amount of information. As for *(iii)*, Table 1 compares RECAP with related systems in terms of: KG supported (**KG**), output (**O**), filtering capabilities (**F**), querying capabilities (**Q**), requirement of local data (**L**). RECAP differs from related systems in the following main respects: as for **KG**, RECAP is *KG-independent*; it only requires the availability of a (remote) query endpoint. Moreover, RECAP can combine information from *multiple* KGs. As for **O** and **F**, RECAP focuses on building *different types* of explanations in the form of graphs or (sets of) paths by leveraging *informativeness* (to estimate the relative importance of edges), *diversity* (to include rare edges) and their combinations. Moreover, RECAP is the only approach that can be used to query KGs (**Q**). As for **L**, *neither* does RECAP assume *local availability* of data *nor* any data *preprocessing*. A more detailed comparison between RECAP and related systems on real data is discussed in Section 4.

Table 1. Comparison of RECAP with closely related systems.

System	KG	O	F	Q	L
REX [4]	Yahoo!	Graph	No	No	Yes
RelFinder [8]	DBpedia	Graph	No	No	Yes
Explass [2]	DBpedia	Paths	Yes (only paths)	No	Yes
RECAP	Any	Graph/Paths	Yes (paths and graphs)	Yes	No

1.3 Contributions and Outline

The framework that we are going to introduce poses several challenges, among which: *(i)* how to capture the notion of relatedness explanation between entities? we leverage informativeness of paths and a diversity criterion to construct different types of explanations; *(ii)* how to query KGs? we isolate the structure of an explanation to learn a SPARQL query; *(iii)* how to make RECAP readily available? We use RDF, the SPARQL query language, and SPARQL endpoints. The contributions of this paper are as follows: *(i)* a framework for building relatedness explanations; *(ii)* different path ranking strategies; *(iii)* a mechanism to query KGs by giving entity pairs as input; *(iv)* a KG-agnostic implementation of our framework; *(v)* an extensive experimental evaluation.

The remainder of the paper is organized as follows. Section 2 introduces the problem and gives some background. Section 3 presents the explanation framework. Section 4 discusses an evaluation of the performance of RECAP and a comparison with related work. We draw some conclusions and sketch future work in Section 5.

2 Problem Formalization

Motivation. The goal of this paper is to facilitate the discovery and explanation of knowledge in knowledge graphs. Part of this research was motivated by the SENSE4US project where explanations are a useful support to discover connectivity between topics emerging from policy documents.

Input. We consider as input a pair (w_s,w_t) of entities defined in some knowledge graph G. We focus on RDF knowledge bases K=$\langle G, O, A\rangle$ where G is a knowledge graph (KG), O is an ontology/schema, and A is a query endpoint.

Assumptions. The framework that we are going to introduce works on top existing knowledge bases. Our approach has to be flexible enough to be applied to different KGs as dictated by the SENSE4US project, which considers a variety of KGs in the LOD cloud. Hence, we consider the access to knowledge bases via the query endpoint A. Neither this requires local availability of the data (e.g., local copies) nor any complex data processing infrastructure from the user side. The computations are reduced to the evaluation of a set of queries against A plus some local refinement.

Desired output. Given K=$\langle G, O, A\rangle$ and a pair of entities $(w_s, w_t) \in G$, the output can be of two different types. It can be an explanation $G_e(w_s, w_t) \subseteq G$. To produce the graph G_e, our explanation algorithm only considers nodes/edges in the set of paths between w_s and w_t. Given a set of paths, we define different mechanisms to rank/select paths to be included in G_e .

When focusing on querying KGs, the output is a set of (pairs of) entities. We isolate the structure of an explanation into an explanation pattern. Given a graph G_e, representing an explanation, an explanation pattern considers a graph G_e^v where nodes in G_e are replaced with variables. G_e^v is used to generate a SPARQL query that is evaluated against A. Results of such query can be locally ranked (e.g., via PageRank [16]).

Basic Definitions and Background. We now define what a knowledge graph (KG) is and outline the fragment of the query language supporting the implementation of our framework. Although there are several KGs today available (e.g., Yahoo!, Google) we will focus on those encoded in RDF[3]. The choice of RDF is merely practical; data in RDF is widely and openly accessible on the Web for querying via SPARQL [6]. Let $\mathcal{U}$ (URIs) and $\mathcal{L}$ (literals) be countably disjoint infinite sets. An RDF *triple* is a tuple of the form $\mathcal{U} \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L})$ whose elements are referred to as *subject*, *predicate* and *object*, respectively. As we are interested in discovering explanations in terms of nodes and edges carrying semantic meaning for the user, we do not consider blank nodes.

Definition 1 (**Knowledge Graph**). Given a set T of RDF triples, a KG is a multigraph G=$\langle V, \mathcal{E} \rangle$ where V=$\{s \mid (s,p,o) \in T\} \cup \{o \mid (s,p,o) \in T\}$ and $\mathcal{E} = \{(s,p,o) \in T\}$.

For the purposes of this paper, we will consider the most basic form of SPARQL queries, that is, Basic Graph Patterns (BGPs). We shall also make usage of the `COUNT` aggregate operator. Let $\mathcal{V}$ be a set of SPARQL variables, that is, strings starting with the `?` symbol, $\mathcal{U}$ a set of URIs and $\mathcal{L}$ a set of literals. A *triple pattern* is a triple of the form $(\mathcal{U} \cup \mathcal{L} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{L} \cup \mathcal{V})$. BGPs are sets of triple patterns that can be combined via algebraic operators; we will make usage of the *join* operator (represented by the symbol `.` in the SPARQL syntax).

3 The RECAP Approach

We see an explanation as a concise representation of the relatedness between entities in terms of edges (carrying a semantic meaning via RDF predicates) and other entities. As graphs are a natural and flexible way to represent and visualize information about interlinked entities in a variety of scenarios, we represent explanations as graphs.

Definition 2 (**Explanation**). Given a knowledge base K=$\langle G, O, A \rangle$ and a pair of entities (w_s,w_t) where $w_s, w_t \in G$, an explanation is a tuple of the form E=(w_s, w_t, G_e), where $w_s, w_t \in G_e$, $G_e \subseteq G$, and G_e is connected.

The above definition is very general; it only states that two entities are connected via nodes and edges in a graph G_e, which is a subgraph of the knowledge graph G and has an arbitrary structure. The challenging aspect is how to uncover the structure of G_e by accessing G only via queries against the endpoint A. To tackle this challenge we shall characterize the desired properties of G_e. Consider the explanation shown in Fig. 4 (a); G_e contains two types of nodes: nodes such as n_1, n_3, n_4 that do belong to some path between w_s and w_t and other nodes such as n_2 that do not.

[3] A list is available at `http://lod-cloud.net`

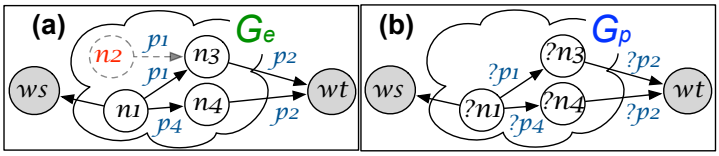

Fig. 4. An explanation (a) and a pattern graph (b).

Although the edge (n_2, p_1, n_3) can contribute to better characterize n_3, it is in a sense non-necessary as it does not directly contribute to explain how w_s and w_t are related. Hence, we introduce the notion of necessary edge.

Definition 3 (**Necessary Edge**). An edge $(n_i, p_k, n_j) \in G$ is necessary for an explanation $E=(w_s, w_t, G_e)$ if it is in a simple path (no node repetitions) between w_s and w_t.

The necessary edge property enables to refine the notion of explanation into that of minimal explanation.

Definition 4 (**Minimal Explanation**). Given $K=\langle G, O, A\rangle$ and a pair of entities (w_s,w_t) where $w_s, w_t \in G$, a minimal explanation is an explanation $E=(w_s, w_t, G_e)$ where G_e is obtained as the merge of all simple paths between w_s and w_t.

Minimal explanations enable to focus only on nodes and edges that are in some path between w_s and w_t thus preserving connectivity information only. The challenge is now how to retrieve minimal explanations.

Consider the explanation shown in Fig. 4 (a) (ignoring the dashed node and edge). G_e could be retrieved by matching the pattern graph G_p in Fig. 4 (b) (nodes and edges are query variables) against G. If the structure of G_p were available, one could find G_e. Unfortunately such structure, that is, the right way of joining query variables representing nodes and edges in G_p is unknown before knowing G_e. As the building blocks of minimal explanations are paths between w_s and w_t, finding these paths is crucial.

Generally speaking, paths between entities can have an arbitrary length; in practice it has been shown that for KGs like Facebook the average distance between entities is bound by a value $k \leq 5$ [20]. Considering paths of length k is also in line with the goal of providing explanations of manageable size that can be visualized/interpreted by users. Finally, related approaches like Explass [2] and REX [4] also considered bounded-length paths. Fig. 5 summarizes the explanation algorithm.

3.1 Finding Paths between Entities

We now describe the structure of queries used to retrieve paths via the endpoint A.

Definition 5 (***k*-connectivity Pattern**). Given $K=\langle G, O, A\rangle$, a pair of entities (w_s,w_t) where $w_s, w_t \in G$ and an integer k, a k-connectivity pattern is a tuple $\Pi=\langle w_s, w_t, \mathcal{Q}, k\rangle$ where $\mathcal{Q}$ is a set of SPARQL queries composed by joining k triple patterns.

Algorithm 1: Building Relatedness Explanations

Input: A pair (w_s,w_t) of entities, an integer k, the address of a query endpoint A
Output: A graph G_e representing an explanation

(1) *Find paths*: we describe in Section 3.1 an approach based on SPARQL queries against A to retrieve paths between w_s and w_t of length k.

(2) *Rank paths*: We describe in Section 3.2 different mechanisms to rank paths by considering informativeness and diversity.

(3) *Select and merge top-m paths:* we discuss in Section 3.3 different ways of selecting ranked paths to build an explanation.

Fig. 5. An overview of the relatedness explanation algorithm.

Note that SPARQL 1.1 supports property paths (PPs) [6], that is, a way for discovering routes between nodes in an RDF graph. However, since variables *cannot* be used as part of the path specification itself, PPs are not suitable for our purpose; we need information about all path elements (i.e., nodes and edges) to build explanations.

Example 6 (**Example of *k*-connectivity Pattern**). The 2-connectivity pattern between F. Lang (`:FL`) and T. von Harbou (`:TvH`) contains the following set of queries $\mathcal{Q}$:

```
SELECT DISTINCT * WHERE{:FL ?p1 ?n1. ?n1 ?p2 :TvH}
SELECT DISTINCT * WHERE{:FL ?p1 ?n1. :TvH ?p2 ?n1}
SELECT DISTINCT * WHERE{?n1 ?p1 :FL. :TvH ?p2 ?n1}
SELECT DISTINCT * WHERE{?n1 ?p1 :FL. ?n1 ?p2 :TvH}
```

Definition 7 (**Path**). Given K=$\langle G, O, A\rangle$ and a k-connectivity pattern Π=$\langle w_s, w_t, \mathcal{Q}, k\rangle$, a path π is a set of edges: $\pi(w_s, w_t)$=$w_s \overset{p_1}{-} n_1 \overset{p_2}{-} n_2 \overset{p_3}{-} n_3..n_q \overset{p_k}{-} w_t$, $n_i \in G\ \forall i \in [1, q]$, $p_j \in G\ \forall j \in [1, k]$ and $- \in \{\leftarrow, \rightarrow\}$.

3.2 Ranking Paths

The number of paths connecting two entities w_s and w_t can be large. Considering the merge of all paths, as done in minimal explanations (see Definition 4), can be an obstacle toward concise explanations. Therefore, we introduce different criteria to rank paths, a subset of which (e.g., top-m) can be merged to form an explanation.

Ranking By Path Informativeness
The first approach to estimate the informativeness of a path connecting a pair of entities (w_s,w_t)$\in G$ leverages the informativeness of its constituent RDF predicates [17].

Definition 8 (**Predicate Frequency Inverse Triple Frequency**). Given a knowledge graph G=$\langle V, \mathcal{E}\rangle$, an entity $w \in G$ and a predicate p appearing in some triple involving w, the incoming $\mathtt{pf}_i^w(p)$ and outgoing $\mathtt{pf}_o^w(p)$ predicate frequency are shown in equation (1) and equation (2), respectively. The Inverse Triple Frequency of p ($\mathtt{itf}(p)$) and the `pfitf` are shown in equation (3) and equation (4), respectively.

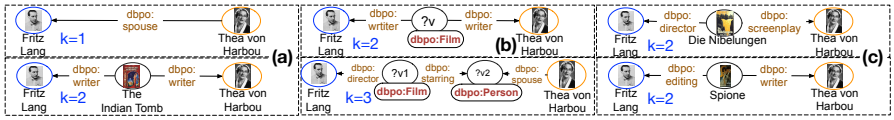

Fig. 6. Ranking: (a) *most* informative paths; (b) *most* informative patterns; (c) most *diverse* paths.

$$\texttt{pf}_i^w(p,G) = \frac{|\mathcal{E}_i(w)|_{\pi(p)}}{|\mathcal{E}_i(w)|} \quad (1) \qquad \texttt{pf}_o^w(p,G) = \frac{|\mathcal{E}_o(w)|_{\pi(p)}}{|\mathcal{E}_o(w)|} \quad (2)$$

$$\texttt{itf}(p,G) = log\ \frac{|\mathcal{E}|}{|\mathcal{E}|_{\pi(p)}} \quad (3) \qquad \texttt{pfitf}_x(p,G) = \texttt{pf}_x \times \texttt{itf} \quad (4)$$

where $|\mathcal{E}_i(w)|_{\pi(p)}$ (resp., $|\mathcal{E}_o(w)|_{\pi(p)}$) is the number of triples in G where the predicate p is incoming (resp., outgoing) in w, $|\mathcal{E}_i(w)|$ (resp., $|\mathcal{E}_o(w)|$) is the total number of incoming (resp., outgoing) triples including w. $|\mathcal{E}|_{\pi(p)}$ is the number of triples including p. In equation (4), $\texttt{pfitf}_x(p,G)$ can use $\texttt{pf}_i^w(p,G)$ or $\texttt{pf}_o^w(p,G)$.

Definition 9 (**Path Informativeness**). Let $\pi(w_s,w_t)$=$w_s \xrightarrow{p} w_t$ be a path between w_s and w_t in G of length k=1. The informativeness of π is defined as:

$$I(\pi,G) = [\texttt{pfitf}_o^{w_s}(p,G) + \texttt{pfitf}_i^{w_t}(p,G)]/2 \quad (5)$$

The informativeness of the path $\pi(w_s,w_t)$=$w_s \xleftarrow{p} w_t$ can be obtained by considering p as an incoming edge to w_s. For paths having length k>1, we have:

$$I(\pi,G) = \frac{I(\pi(w_s,w_1),G) + ... + I(\pi(w_k,w_t),G)}{k} \quad (6)$$

Ranking By Pattern Informativeness

We now introduce informativeness based on path patterns. A path pattern generalizes a path by replacing nodes with variables.

Definition 10 (**Path Pattern**). Given a path $\pi(w_s,w_t)$=$w_s \overset{p_1}{-} n_1 \overset{p_2}{-} n_2..n_q \overset{p_q}{-} w_t$, a path pattern is an expression of the form $\overline{\pi}(w_s,w_t)$=$w_s \overset{p_1}{-} ?v_1 \overset{p_2}{-} ?v_2..?v_q \overset{p_q}{-} w_t$, where $?v_i$ $i \in \{1,2,...q\}$ are variables and $q \leq k$.

As an example, the path in the bottom-part of Fig. 6 (a) is abstracted in the pattern in the top-part of Fig. 6 (b). The usage of variables in place of intermediate entities enables to represent in a more concise way information about a *set of* paths. The pattern in the top-part of Fig. 6 (b) enables to capture the fact that F. Lang and T. von Harbou have co-written 11 movies (bindings of the variable `?v`) according to DBpedia. Information in the ontology O (when available) can help to more precisely characterize the nature of intermediate entities by considering their `rdf:type` (Fig. 6 (b)). `RECAP` includes a pattern-based exploration of the connectivity between w_s and w_t along with the possibility to generate explanations including all paths matching a pattern (see Fig. 3 (a)).

Definition 11 (**Path pattern informativeness**). Let $P_{\overline{\pi}}$ be the set of patterns obtained from a set of paths P_π. The informativeness of a path pattern $\overline{\pi} \in P_{\overline{\pi}}$ is:

$$I(\overline{\pi}, G) = log\ \frac{|P_{\overline{\pi}}|}{|(\overline{\pi}, G)|} \tag{7}$$

where $|P_{\overline{\pi}}|$ is the number of patterns and $|(\overline{\pi}$,G)| is the number of paths sharing $\overline{\pi}$ in G.

Ranking By Path Diversity

The most informative paths of length k=2 between F. Lang and T. von Harbour often include predicates related to the fact that they have co-written movies (e.g., The Indian Tomb and Metropolis); this will potentially discard other predicates appearing in paths with low informativeness. To cope with this aspect, we introduce path diversity.

Definition 12 (**Path Diversity**). Given a source entity $w_s \in G$, a target entity $w_t \in G$ and two paths $\pi_1(w_s, w_t)$ and $\pi_2(w_s, w_t)$ we define path diversity as:

$$\delta(\pi_1, \pi_2) = \frac{|\texttt{Labels}(\pi_1) \cap \texttt{Labels}(\pi_2)|}{|\texttt{Labels}(\pi_1) \cup \texttt{Labels}(\pi_2)|} \tag{8}$$

where `Labels(`π`)` denotes the set of labels (RDF predicates) in a path. Fig. 6 (c) shows the two most diverse paths at distance 2 between F. Lang and T. von Harbou. As it can be observed, the predicate `dbpo:screenplay` is included; such predicate is never present in the top-10 most informative paths.

3.3 Selecting and Merging Paths

The last step of the explanation algorithm concerns path selection. Table 2 describes different strategies that given a value m select a subset (but $E^{\cup}$) of paths (patterns) according to one of the three approaches described in Section 3.2. Moreover, two strategies combine path (pattern) informativeness and diversity. The strategy in the last line of Table 2 does not merge paths and is used by `RECAP` to enable pattern-based explorations of the relatedness between w_s and w_t. We discuss an evaluation of the different strategies in Section 4.

Table 2. Path selection/merging strategies.

Symbol	Meaning
$E^{\cup}$	Merge all of paths
E^{π}_{m}	Merge the *top-m* most informative paths
$E^{\overline{\pi}}_{m}$	Merge paths belonging to the *top-m* most informative path *patterns*
E^{δ}	Merge paths whose value of diversity falls in $[(max - r)$,$max]$ where max is the max diversity and r is a % value.
$E^{\pi,\delta}$	Merge the results of E^{π}_{m} and E^{δ}
$E^{\overline{\pi},\delta}$	Merge the results of $E^{\overline{\pi}}_{m}$ and E^{δ}
$\mathcal{P}$	Set of all paths (no merge)

3.4 Querying KGs by Example

We now describe the second building block of our framework, that is, an algorithm (shown in Fig.7) to query KGs by giving a pair of entities as input. In what follows we outline the steps, but (1), of Algorithm 2 after introducing explanation patterns.

Algorithm 2: Knowledge Graph Querying

Input: A pair (w_s,w_t) of entities, an integer k, the address of a query endpoint A
Output: A set of ranked (pairs of) entities

(1) *Find an explanation* E=(w_s, w_t, G_e) between w_s and w_t by using Algorithm 1.

(2) *Build* the entity query pattern Q_e.

(3) Query the KG with Q_e (via A) and get a set of (pairs of) entities.

(4) Rank the answers to Q_e.

Fig. 7. An overview of the query answering algorithm.

Definition 13 (**Explanation Pattern**). Given an explanation E=$\langle w_s, w_t, G_e \rangle$, an explanation pattern is a tuple $\overline{E}$=$\langle ?w_s, ?w_t, G_e^v \rangle$ where G_e^v=$\{\texttt{TP}_1,\texttt{TP}_2,...,\texttt{TP}_k\}$ is a query graph and $\texttt{TP}_i$=$(\mathcal{U} \cup \mathcal{L} \cup \mathcal{V}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L} \cup \mathcal{V}), 1 < i < k$, is a triple pattern not containing variables in predicate position. Moreover, for $i > 1$ $|\texttt{var(TP}_i\texttt{)} \cap \texttt{var(TP}_{i-1}\texttt{)}| = 1$.

In the above definition, G_e^v is the query graph obtained from G_e by replacing *all nodes* into an explanation with query variables. Basically, an explanation pattern generalizes the structure of an explanation by keeping edge labels only. Explanation patterns are used to generate entity query patterns.

Definition 14 (**Entity Query Pattern**). An entity query pattern is a SPARQL query of the form: `SELECT DISTINCT ?ws ?wt WHERE{TP1. TP2. TPk.}`

In the above definition, `TPi`, $i \in [1, k]$ are triple patterns in G_e^v and `?ws` and `?wt` are variables used in lieu of the entities in input. Query patterns are *automatically derived*; our algorithm neither requires familiarity with SPARQL nor with the underlying data/schema. The evaluation of a query pattern returns a set of pairs of entities.

Ranking of Results

Our approach for querying KG learns an entity query pattern Q_e from a relatedness explanation. Since the evaluation of Q_e can return a large number of results, our algorithm includes a ranking component. The problem of ranking results of SPARQL queries has been already studied (e.g., [1, 13]) and is not the main purpose of the present paper.

Inspired by the Google KG, we consider a simple result ranking mechanism based on the popularity of entities; specifically, we leverage the PageRank [16] algorithm. Given a pair of entities (w_1, w_2) returned when evaluating Q_e (obtained from step (3) Algorithm 2), we estimate their popularity as $(PR(w_1) + PR(w_2))$ /2, where $PR(w_i)$ is the PageRank value of the entity i. We leave as a future work the investigation of more sophisticated result-ranking mechanisms.

4 Implementation and Evaluation

We have implemented our ideas in the RECAP tool, which uses JavaFX[4] for the GUI and the Jena[5] framework to handle RDF data and SPARQL queries.

4.1 Evaluating the Explanation Generation Component

We start by discussing the evaluation of the explanation component of RECAP.

Experimental setting. We considered two KGs: DBpedia (DB)[6] and Freebase[7] (FB). We adopt the dataset of 26 pairs used to evaluate Explass [2] and set k$\leq$4 as done in Explass. We use as reference graph for the computation of informativeness scores (see Def. 8 and Def. 11) the graph obtained by merging all paths. Experiments have been performed on a MacBook Pro with a 2.8 GHz i7 CPU and 16GBs RAM.

Experiment 1: Performance Evaluation: we investigate the performance of RECAP for *increasing values* of k[8] in terms of: (i) *obtaining paths*; (ii) *computing explanations*. Results that follow are the average of 5 runs. Fig. 8 (a) and (b) show the running times. Clearly, the higher k the higher the running time for path retrieval. The multi-thread implementation of RECAP allows to keep the time for finding paths on average around 6.4 secs for DB and 12 secs for FB when $k\leq$4. When executing the queries *sequentially* (results are not reported for sake of space) the running times can be up to 30 times higher. We observed in another experiment on DB (not reported for sake of space) that local data reduces the running times by $\sim$60% on average. However, this has the disadvantage that both a local processing infrastructure and local data are required.

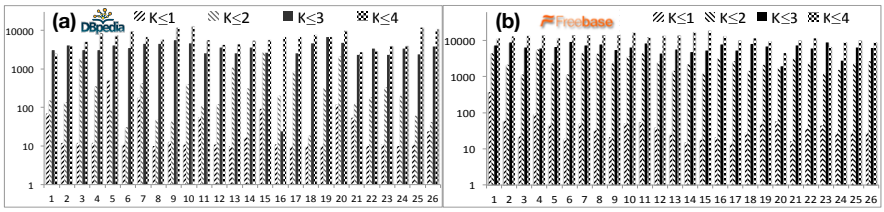

Fig. 8. Path retrieval in DB (a) and FB (b). Y-axis: time(ms) in log-scale; X-axis: entity pair.

Running times on DB for generating the different types of explanations described in lines 1-4 of Table 2 are shown in Figs. 9 (a). We report results on DB as this KG has been used in the (qualitative) comparison of RECAP with related approaches (see Section 4.1). Nevertheless, we report results on the combination DB-FB in Fig. 9 (b).

[4] http://docs.oracle.com/javafx/
[5] https://jena.apache.org/
[6] http://dbpedia.org/snorql
[7] http://lod.openlinksw.com/sparql
[8] In particular, for each k, all paths of length $\leq k$ are generated

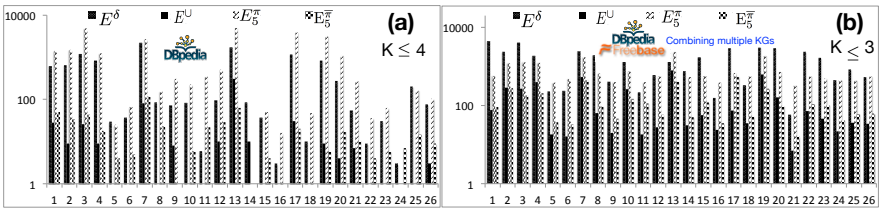

Fig. 9. Explanations in DB (a) and DB/FB (b). Y-axis: time(ms) in log-scale; X-axis: entity pair.

Generally speaking, $E^{\cup}$ explanations can be generated very fast; here, no path ranking/filtering is performed. However, $E^{\cup}$ can be very big, which makes the interpretation by users difficult, as we will discuss in Experiment 2. E^{δ} explanations that use diversity (we considered r=25%) are more expensive as they require the computation of distances between paths, for which RECAP leverages a multi-thread approach. Explanations based on path informativeness E_m^{π} (we considered m=5) require to compute pfitf scores; RECAP computes these scores in parallel and using the merge of all paths as reference graph thus not performing any remote query. Explanations based on pattern informativeness $E_m^{\overline{\pi}}$ (we considered m=5) are less expensive since they do not analyze the informativeness of all edges in a path. The most expensive explanations (not reported here for sake of space) are those combining path/pattern informativeness and diversity requiring ∼6 secs for $k \leq 4$. When compared to related system (see Section 4.1), RECAP has been judged the fastest system in the overall task of generating different types of relatedness explanation. In terms of size (results not reported for sake of space), $E^{\cup}$ are the biggest one; their size can include up to 4000 paths ($k \leq 4$) for pair 12 (C. Bale, C. Nolan) in FB.

Explanations of type E_5^{π} are smaller; the typical size is ∼8 nodes and ∼7 edges. $E_5^{\overline{\pi}}$ have variable size as it depends on the number of paths for each of the top-5 most informative patterns. In general these are bigger than E_5^{π} explanations (∼15 nodes and ∼12 edges). Note that $E_5^{\overline{\pi}}$ explanations enable to focus on specific aspects as they include all the instantiations of each of the top-5 most informative path patterns. The sizes of E^{δ} are in the same order of magnitude as $E_5^{\overline{\pi}}$; however E^{δ} explanations guarantee to also include rare edges potentially discarded by path or pattern informativeness. The typical size of an explanation combining (top-5) path/pattern informativeness and diversity (r=25%) is ∼20 nodes and ∼15 edges. The possibility, featured by RECAP, to decide the amount of information to be included into an explanation is crucial toward understanding relatedness.

Experiment 2: Interpreting explanations: this experiment aims at: *(i)* investigating whether RECAP *provides useful explanations to the user*; *(ii)* comparing RECAP against two related systems online available[9], that is, Explass [2][10] and RelFinder [8][11]. We used DB for the comparison as Explass and RelFinder only work on this KG.

[9] REX [4] is not available for public usage

[10] http://ws.nju.edu.cn/explass/

[11] http://www.visualdataweb.org/relfinder/relfinder.php

Setting. Twenty participants were assigned each six random pairs among the 26 entity pairs. They were shown how the three systems work and asked to use each system (with no other support) in order to understand the relatedness between entities in a pair. Following the methodology in [2] participants were given a set of six questions; the response to each question was given with an agreement value from 1 (min) to 5 (max). Q6 was not considered in [2]; we included it to understand how users perceive the performance of the systems in terms of running time. Results are reported in Table 3.

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

Question	RECAP	RelFinder	Explass
Q1: Information overview	4.55(0.65)	3.05(0.77)	3.82(0.75)
Q2: Easiness in finding information	4.45(0.55)	4.05(0.63)	3.85(0.67)
Q3: Easiness in comparing/synthesizing info	4.62(0.62)	3.10(0.82)	4.06(0.61)
Q4: Comprehensive support	4.81(0.73)	3.42(0.77)	4.15(0.79)
Q5: Sufficient support to the task	4.67(0.81)	3.28(0.86)	4.23(0.83
Q6: Running time	4.82(0.48)	4.12(0.72)	3.18(0.52)

According to questions Q1-Q5, users perceived RECAP and Explass as better supports to the explanation task. Users reported that RelFinder does not allow the flexible creation of explanation (e.g., by grouping paths into patterns), which makes it hard to control the amount of information shown. In general, RECAP was judged to be a more comprehensive solution; it provides both a graph-based and pattern-based exploration of results and several ways of controlling the amount of information to be shown. While RECAP and RelFinder quickly provide information immediately after retrieving paths, Explass requires a much longer time. On Q6 Explass was judged to be the less compelling system. RECAP was judged higher than the other two systems in all questions via LSD post-hoc tests ($p < 0.05$). The inter-annotator agreement was of 0.85.

Combining multiple KGs. We tested RECAP on the *combination* of DB and FB (see Fig. 9 (d)). Starting from DB, for the source/target entities we looked at owl:sameAs links to the corresponding FB entities. We then merged the set of paths from each KG by using owl:sameAs links. Users ($\sim$75%) perceived the combination of multiple KGs as very useful toward more comprehensive explanations. This is especially true when KGs cover the same domain with different levels of detail (FB was judged more comprehensive than DB). The combination also produces graphs of bigger size. Indeed, the functionality of RECAP allowing to filter information to be included into an explanation was judged very useful (participants thought $E^{\cup}$ were too big when k$\geq$ 3).

4.2 Evaluating the Querying Component

We now discuss the evaluation of the querying component of RECAP.

Experimental setting. We used the dataset of 18 pairs defined by Jayaram et al. [10] and considered DBpedia as KG. In order to rank query results, we compute PageRank values for the latest version of DBpedia and stored them in a local Lucene[12] index.

[12] https://lucene.apache.org/

Table 4. Accuracy of RECAP (m=10).

Pair	P@m	nDCG	Pair	P@m	nDCG
P_1	0.91	0.94	P_{10}	0.87	0.91
P_2	0.82	0.92	P_{11}	0.75	0.78
P_3	0.73	0.87	P_{12}	0.72	0.78
P_4	0.67	0.72	P_{13}	0.81	0.89
P_5	0.74	0.83	P_{14}	0.82	0.85
P_6	0.82	0.85	P_{15}	0.84	0.86
P_7	0.72	0.81	P_{16}	0.78	0.84
P_8	0.69	0.77	P_{17}	0.62	0.72
P_9	0.81	0.85	P_{18}	0.79	0.82

Table 5. Accuracy of RECAP (m=15).

Pair	P@m	nDCG	Pair	P@m	nDCG
P_1	0.78	0.82	P_{10}	0.81	0.83
P_2	0.78	0.79	P_{11}	0.72	0.74
P_3	0.71	0.72	P_{12}	0.65	0.71
P_4	0.62	0.68	P_{13}	0.71	0.74
P_5	0.68	0.73	P_{14}	0.68	0.71
P_6	0.64	0.72	P_{15}	0.70	0.71
P_7	0.62	0.71	P_{16}	0.68	0.72
P_8	0.61	0.68	P_{17}	0.62	0.67
P_9	0.78	0.81	P_{18}	0.67	0.74

Evaluation metrics. The aim of this experiment is to measure how precise are the results returned by RECAP as compared to a gold-standard. We measure the accuracy on a query by considering: **(i)** Precision-at-m (P@m): the percentage of the top-m results in the ground truth; **(ii)** Normalized Discounted Cumulative Gain (nDCG): the cumulative gain of the top-m results is DCGm=$rel_1 + \sum_{1=2}^{m} \frac{rel_i}{log_2(i)}$; it penalizes the results if the ground truth result is ranked low. DCG$_m$ is normalized by IDCG$_m$, the cumulative gain for an ideal ranking of the top-m results. Thus nDCGm=$\frac{DCG_m}{IDCG_m}$.

We report results for m=10 (in Table 4) and m=15 (in Table 5) that consider top-10 and top-15 entity pairs, respectively. We use E^{π}_{10} (top-10 most informative paths) explanations, at step (1) of Algorithm 2, to generate entity query patterns.

As it can be observed, the usage of PageRank scores, as a mechanism to weight the importance of query results, brings acceptable performance. In the majority of the 18 pairs, the P@10 score is above 0.75. In some cases like P_1 (i.e., Nike, Tiger Woods) RECAP was able to identify almost all the other entities (in the gold standard), among which M. Jordan, and K. Bryant (also sponsored by Nike). When the value of m increases performance decreases. However, usually providing top-10 results[13] is an acceptable compromise. Note that the nDCG is in most of the cases above 0.7; in this measure, the DCG emphasizes pairs of entities that appear early in the set of results. We leave as a future work the investigation of more sophisticated ranking mechanisms.

In terms of running time, the overhead introduced (besides explanation generation) by Algorithm 2 consists in the access to the Lucene index to retrieve PageRank scores and the computation of their average value. Typically, the overall running time for path finding, explanation generation and result ranking is $\sim$10 secs.

5 Concluding Remarks and Future Work

We have introduced a framework to generate different types of relatedness explanations, possibly including information from multiple KGs. Our work is motived by the SENSE4US FP7 project, where there is the need to find topic connectivity information.

We have faced another important problem: querying KGs by using entities as input. As of today, either KGs provide limited querying capabilities (e.g., by accepting

[13] Search engines usually provide top-10 results per page

one entity as input) or require familiarity with languages such as SPARQL besides the underlying schema/data. We have shown how the usage of the relatedness explanation between a pair of entities can help in learning SPARQL queries to find other pairs of related entities. We plan to investigate optimization mechanisms to reduce the running time for path finding. One approach could be to leverage the ontology O to generate candidate queries according to paths between entities at the schema level, rank these queries, and check the most promising.

References

1. K. Anyanwu, A. Maduko, and A. Sheth. SemRank: Ranking Complex Relationship Search Results on the Semantic Web. In *WWW*, pages 117–127, 2005.
2. G. Cheng, Y. Zhang, and Y. Qu. Explass: Exploring Associations between Entities via Top-K Ontological Patterns and Facets. In *ISWC*, pages 422–437. Springer, 2014.
3. C. Faloutsos, K. S. McCurley, and A. Tomkins. Fast Discovery of Connection Subgraphs. In *SIGKDD*, pages 118–127. ACM, 2004.
4. L. Fang, A. D. Sarma, C. Yu, and P. Bohannon. REX: Explaining Relationships between Entity Pairs. *VLDB*, 5(3):241–252, 2011.
5. V. Fionda, C. Gutierrez, and G. Pirrò. Knowledge Maps of Web Graphs. In *KR*, 2014.
6. S. Harris and A. Seaborne. SPARQL 1.1 Query Language W3C Recommendation, 2013.
7. T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, 1st edition, 2011.
8. P. Heim, S. Lohmann, and T. Stegemann. Interactive Relationship Discovery via the Semantic Web. In *ESWC*, pages 303–317. Springer, 2010.
9. H. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making Database Systems Usable. In *Int. Conf. on Management of Data*, pages 13–24. ACM, 2007.
10. N. Jayaram, M. Gupta, A. Khan, C. Li, X. Yan, and R. Elmasri. GQBE: Querying Knowledge Graphs by Example Entity Tuples. In *ICDE*, pages 1250–1253. IEEE, 2014.
11. G. Kasneci, S. Elbassuoni, and G. Weikum. Ming: Mining Informative Entity Relationship Subgraphs. In *CIKM*, pages 1653–1656. ACM, 2009.
12. G. Luo, C. Tang, and Y.-l. Tian. Answering Relationship Queries on the Web. In *WWW*, pages 561–570. ACM, 2007.
13. S. Magliacane, A. Bozzon, and E. Della Valle. Efficient execution of top-k sparql queries. In *The Semantic Web–ISWC 2012*, pages 344–360. Springer, 2012.
14. P. N. Mendes, P. Kapanipathi, D. Cameron, and A. P. Sheth. Dynamic Associative Relationships on the Linked Open Data Web. In *Web Science Conference*, 2010.
15. N. Nakashole, G. Weikum, and F. Suchanek. Discovering and Exploring Relations on the Web. *VLDB*, 5(12):1982–1985, 2012.
16. L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. 1999.
17. G. Pirrò. REWOrD: Semantic Relatedness in the Web of Data. In *26th Conference on Artificial Intelligence (AAAI)*, 2012.
18. C. Ramakrishnan, W. H. Milnor, M. Perry, and A. P. Sheth. Discovering informative connection subgraphs in multi-relational graphs. *SIGKDD Newsletter*, 7(2):56–63, 2005.
19. A. Sheth, B. Aleman-Meza, I. B. Arpinar, C. Bertram, Y. Warke, C. Ramakrishanan, C. Halaschek, K. Anyanwu, D. Avant, F. S. Arpinar, et al. Semantic Association Identification and Knowledge Discovery for National Security Applications. *Journal of Database Management*, 16(1):33–53, 2005.
20. J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The Anatomy of the Facebook Social Graph. *arXiv preprint arXiv:1111.4503*, 2011.