RECAP: Building Relatedness Explanations on the Web

Giuseppe Pirrò *
ICAR-CNR
Rende (CS), Italy
pirro@icar.cnr.it

Alfredo Cuzzocrea
DIA Department
University of Trieste&ICAR-CNR, Italy
alfredo.cuzzocrea@dia.units.it

ABSTRACT

We describe RECAP, a tool that, given a pair of entities defined in some Knowledge Graph (KG), builds an explanation, that is, a graph (of manageable size) reflecting their relatedness. Explanations enable to discover new knowledge and browse toward other entities of interest. We discuss different kinds of explanations based on information theory and diversity. The KG-agnostic approach adopted by RECAP, which retrieves the necessary information via SPARQL queries, makes it readily usable on a variety of KGs.

Keywords

Relatedness Explanation, SPARQL, RDF, Path Ranking

1. INTRODUCTION

Knowledge graphs (KGs) are becoming a common support for browsing, searching and knowledge discovery activities on the Web. Search engines like Google, Yahoo! and Bing complement the classical search results with facts about entities in their KGs. Fig. 1 (a) (resp., (b)) show information provided the Google KG (resp., Yahoo! KG) when giving the entity F. Lang as input; facts can also include relationship with other entities (e.g., Vienna). Fig. 1 (c) depicts information about F. Lang taken from DBpedia. Note that both Google and Yahoo! KGs suggest entities like T. von Harbou as *related* to F. Lang; in particular, Google provides a short comment saying that T. von Harbour was F. Lang's

it is more easily understandable because of its visual representation; (ii) the visualization can be dynamically adjusted to include more/less information; (iii) it enables to discover new entities like movie The Indian Tomb and its semantic relationships with F. Lang and T von Harbou.

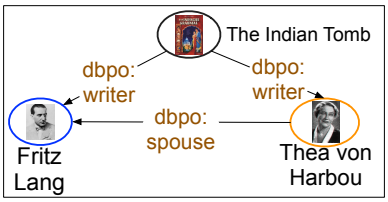

Figure 2: A Relatedness Explanation.

2. OVERVIEW OF THE APPROACH

Motivation. RECAP goes beyond existing KGs applications in terms of discovering and explaining knowledge on the Web. One major issue toward exploiting KGs is that either they provide limited querying capabilities (e.g., giving only one entity in input) or require knowledge of query languages such as SPARQL and underlying data/schema.

Input. The input of our problem is a pair (w_s,w_t) of entities defined in some knowledge graph G. In particular, we consider RDF knowledge bases $K=\langle G, O, A\rangle$ where G is a knowledge graph, O is an ontology/schema used to structure data in G, and A is a query endpoint.

Assumptions. RECAP works on top of existing knowledge bases; it retrieves data only via the query endpoint A. This has a significant advantage as it *neither requires* local availability of the data (e.g., by creating local copies) *nor* any complex data processing (infrastructure) from the user side. The computations performed by RECAP are reduced to the problem of executing a *set* $\mathcal{Q}$ of queries against A plus some algorithmic refinements. This makes RECAP knowledge-based agnostic and readily available to be used in a variety KGs about general knowledge (e.g., DBpedia, Freebase), entertainment (e.g., LinkedMDB, Jamendo), bioinformatics (e.g., Bio2RDF), and so forth.

Output. Given and a pair of entities (w_s,w_t), the output is a graph $E(w_s, w_t) \subseteq G$, which explains their relatedness and includes paths connecting w_s and w_t via semantic relationship and other entities. We call such a graph the *relatedness explanation*.

2.1 Building Relatedness Explanations

A relatedness explanation is a graph that provides a (concise) representation of the relatedness between entities in terms of RDF predicates (carrying a semantic meaning) and other entities.

Definition 1. (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 $E=(w_s, w_t, G_e)$ such that $w_s, w_t \in G_e$ and $G_e \subseteq G$.

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 on the endpoint A. To tackle this challenge, we shall characterize the desired properties of G_e. Consider the explanation shown in Fig. 3 (a). G_e contains two types of nodes: nodes such as n_1, n_3, n_4 that belong to some path between w_s and w_t and other nodes such as n_2 that do not.

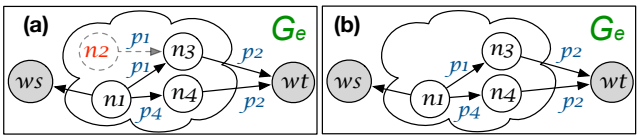

Figure 3: Explanation (a); Minimal explanation (b).

Although the edge (n_2, p_1, n_3) can contribute to better characterize n_3 such edge 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 *essential edge*.

Definition 2. (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 belongs to 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 3. (Minimal Explanation). Given a knowledge base $K=\langle G, A\rangle$ and a pair of entities (w_s,w_t) such that $w_s, w_t \in G$, a minimal explanation $E=(w_s, w_t, G_e)$ requires G_e to be the merge of all simple paths[1] between w_s and w_t.

Fig. 3 (b) shows a minimal explanation. Minimal explanations enable to focus on nodes and edges that are in some path between w_s and w_t only; hence, minimal explanations preserve connectivity information only.

After defining what an explanation is, the challenging question is how to retrieve it. Consider the minimal explanation shown in Fig. 3 (b). It could be retrieved by matching the *pattern graph* G_p shown in Fig. 4 (nodes and edges are query variables) against G. Hence, if the structure of G_p were available one could easily find G_e; however, such structure, that is, the right way of joining query variables representing nodes and edges in G_p is unknown before knowing G_e. Minimal explanations are built by considering (simple) paths between w_s and w_t; hence, the retrieval of such paths is the first step toward building explanations.

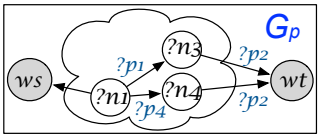

Figure 4: The Pattern Graph for the Minimal Explanation in Fig. 3 (b).

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 $k \leq 5$ [12]. The choice of considering paths of length k in our approach is reasonable on the light of the fact that we focus on providing explanations of *manageable size* that can be *visualized and interpreted* by the user. An overview of the algorithm to build relatedness explanations is shown in Fig. 5.

[1] A simple path does not allow node repetitions.

Algorithm 1: Building Explanations

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

(1) *Find paths*: retrieve paths between w_s and w_t of length k via SPARQL queries against the endpoint A.

(2) *Rank paths*: minimal explanations are constructed by merging *all* paths between w_s and w_e. To control the amount of information into an explanation we defined different mechanisms to rank paths.

(3) *Select and merge top-m paths:* we defined different ways of selecting ranked paths to build an explanation (see Table 1).

Figure 5: The explanation building algorithm.

In what follows, we provide an overview of the three main steps of the algorithm.

Finding Paths. A path is a sequence of edges (RDF triples) bound by a length value k. The assumption of our approach is to access a KG *only* via the query endpoint A.

Definition 4. (***k*-connectivity Pattern**). Given a knowledge base $K=\langle G, O, A\rangle$ a pair of entities (w_s, w_t), such that $w_s, w_t \in G$ and an integer k, a *k-connectivity pattern* is a tuple of the form $\Pi=\langle w_s, w_t, \mathcal{Q}, k\rangle$ where $\mathcal{Q}$ is a set SPARQL queries composed by joining k triple patterns.

Fig. 6 shows the structure of the SPARQL queries in $\mathcal{Q}$; here, both nodes (but w_s and w_t) and edges represent query variables. To model the structure of a path, each of the k triple patterns in Fig. 6 is joined with the subsequent via a shared node. Note also that, in the Figure, edge directions are not reported; each edge has to be considered both as incoming and outgoing, which corresponds to join triple patterns in all possible ways. We emphasize that queries to retrieve paths are automatically generated in RECAP.

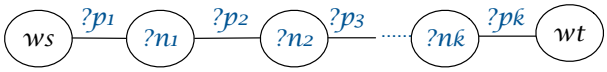

Figure 6: Query to Find k-length Paths.

Example 5. (**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 ?n1 ?p2}
SELECT DISTINCT * WHERE{?n1 ?p1 :FL. ?n1 ?p2 :TvH}
```

Ranking Paths. We outline three path-ranking strategies available in RECAP (see [10] for further details).

- *Path informativeness*: it is estimated by investigating the informativeness of RDF predicates in a path via the notion of *Predicate Frequency Inverse Triple Frequency* (pfitf) [9].
- *Path pattern informativeness*: a path pattern generalizes a path by replacing nodes with variables. Pattern informativeness is computed by counting the number of paths sharing a certain path pattern.
- *Path diversity*: it takes into account the variety of predicates in a set of paths; diversity guarantees to rank high paths that contain rare predicates.

Selecting and Merging Paths. Table 1 describes the last components of Algorithm 1, that is, different strategies to select a subset of ranked paths and merge them to build an explanation.

Table 1: Path selection strategies.

	Meaning
$E^{\cup}$	Merge all of paths without any pruning
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 pairs of paths whose value of diversity falls in $[max, (max - r)]$ where max is the max diversity value over all pairs of paths 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 (without merge)

2.2 The RECAP tool

RECAP has been implemented Java-based tool. RECAP leverages the Jena framework[2] to handle RDF data and JavaFX[3] for the GUIs; this allows the tool to be accessible across different platforms. SPARQL queries are sent to the query endpoint by using the HTTP protocol. Generally, the tool makes usage of SELECT SPARQL queries with the path ranking component also requiring the usage of COUNT. The implementation of our framework leverages multi-threading to execute the set of queries necessary to retrieve paths between entities; this reduces the overall running time to a few seconds. An extensive experimental evaluation is available in [10].

3. OVERVIEW OF THE TOOL

The GUI of the RECAP tool is shown in Fig. 7. It provides an intuitive way of selecting entities for which one wants to find a relatedness explanation. The auto completion function in Fig. 7 (a) enables to find the correct URI of the entities of interest; in particular, the Wikipedia infobox corresponding to each entity is loaded thus giving some quick information about the entities considered. The interface also enables to chose the maximum path distance (Fig. 7 (b)) to be considered; users can experiment with different path length thus having an idea of how the amount of retrieved information and the running time change.

After retrieving paths about the entities, users have the possibility to construct and visualize several types of explanations. Fig. 7 (c) shows the explanation build when considering the top-15 most informative paths; the type of explanation is also shown in the GUI (Fig. 7 (d)). The interface also provides statistics about the explanation building process such as number of paths and execution time (Fig. 7 (e)). When clicking on a node in an explanation, the user can visualize information about such node in Wikipedia (Fig. 7 (k)). When clicking on an edge information about the edge

[2] http://jena.apache.org
[3] http://docs.oracle.com/javafx

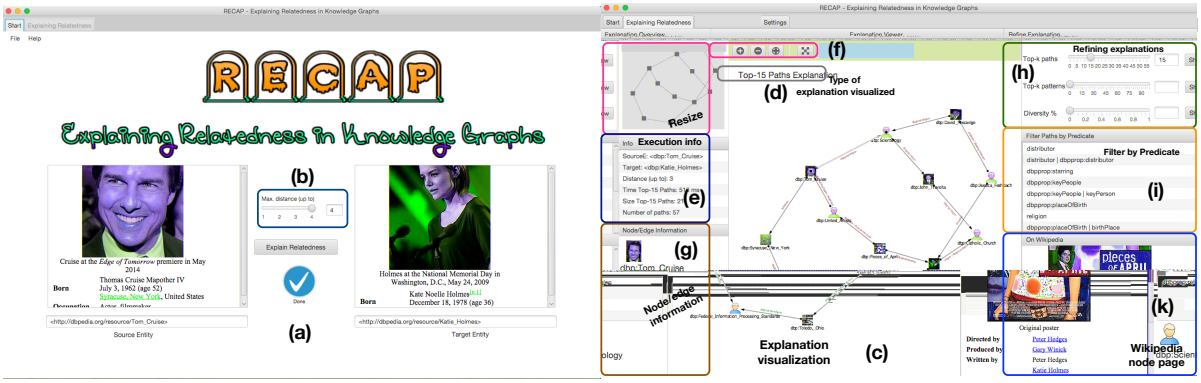

Figure 7: The RECAP main GUI (a); the Relatedness Perspective (b).

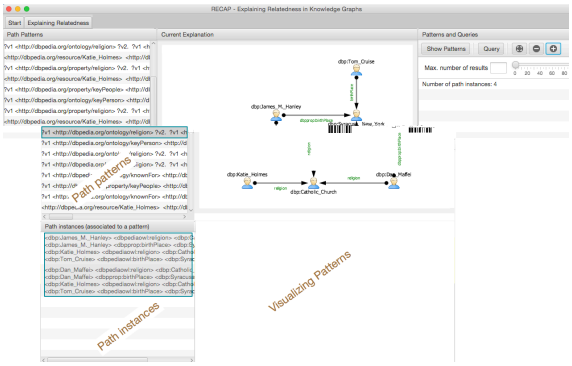

Figure 8: Path Patterns Exploration.

will be also visualized (Fig. 7 (g)). The visualization can be adjusted via the panel in Fig. 7 (f). Part (i) in Fig. 7 allows to filter an explanation according to certain types of RDF predicates. The portion of the interface that controls the explanation building process is shown in Fig. 7 (h); here it is possible to select the set of paths to be considered on the basis of path (resp., pattern) informativeness or diversity. Another portion of the RECAP GUI, shown in Fig. 8, allows to explore paths, patterns and visualize connectivity information by using these (without merging).

During the demo we will showcase examples of relatedness explanations in different domains including cinema, music and bibliography networks. We will provide a list of examples in such domains and help users in getting familiar with the RECAP tool. As an example we will consider pairs like (D. Knuth, L. Lamport), (A. Lincoln, J. Washington). Users can explore different ways of building relatedness explanations thus possibly discovering interesting things like the fact that T. von Harbou besides being F. Lang's former spouse co-directed with him 11 movies.

Our primary goal is to show how RECAP is flexible and can be applied to different knowledge domain without any data preprocessing. Indeed, data will be accessed online via SPARQL endpoints. We will also provide a local SPARQL endpoint with an excerpt of the Yago KG to tackle the possibility that some KGs maybe temporary offline.

4. REFERENCES

[1] Mario Cannataro, Alfredo Cuzzocrea, Carlo Mastroianni, Riccardo Ortale, Andrea Pugliese, et al. Modeling Adaptive Hypermedia with an Object-oriented Approach and XML. In *2nd Int. Workshop on Web Dynamics*, 2002.

[2] Mario Cannataro, Alfredo Cuzzocrea, and Andrea Pugliese. A Probabilistic Approach to Model Adaptive Hypermedia Systems. In *1st Int. Workshop on Web Dynamics*, page 50, 2001.

[3] Gong Cheng, Yanan Zhang, and Yuzhong Qu. Explass: Exploring Associations between Entities via Top-K Ontological Patterns and Facets. In *International Semantic Web Conference*, pages 422{437. Springer, 2014.

[4] Christos Faloutsos, Kevin S McCurley, and Andrew Tomkins. Fast discovery of connection subgraphs. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 118{127. ACM, 2004.

[5] Lujun Fang, Anish Das Sarma, Cong Yu, and Philip Bohannon. REX: Explaining Relationships between Entity Pairs. *Proceedings of the PVLDB*, 2011.

[6] V. Fionda, C. Gutierrez, and G. Pirro. Knowledge Maps of Web Graphs. In *14th International Conference on Principles of Knowledge Representation and Reasoning*, 2014.

[7] V. Fionda, G. Pirro, and C. Gutierrez. NautiLOD: A Formal Language for the Web of Data Graph. *ACM Transactions on the Web (TWEB)*, 9(1):5, 2015.

[8] Philipp Heim, Sebastian Hellmann, Jens Lehmann, Ste en Lohmann, and Timo Stegemann. RelFinder: Revealing Relationships in RDF Knowledge Bases. In *Semantic Multimedia*, pages 182{187. Springer, 2009.

[9] G. Pirro. REWOrD: Semantic Relatedness in the Web of Data. In *26th AAAI Conference*, 2012.

[10] G. Pirro. Explaining and suggesting relatedness in knowledge graphs. In *Proceedings of the 14th International Semantic Web Conference*, pages 622{639. Springer, 2015.

[11] Cartic Ramakrishnan, William H Milnor, Matthew Perry, and Amit P Sheth. Discovering informative connection subgraphs in multi-relational graphs. *ACM SIGKDD Explorations Newsletter*, 7(2):56{63, 2005.

[12] Johan Ugander, Brian Karrer, Lars Backstrom, and Cameron Marlow. The Anatomy of the Facebook Social Graph. *arXiv preprint arXiv:1111.4503*, 2011.