LODPeas: Like peas in a LOD (cloud)

Aidan Hogan, Emir Muñoz, and Jürgen Umbrich

Digital Enterprise Research Institute, National University of Ireland, Galway
firstname.lastname@deri.org

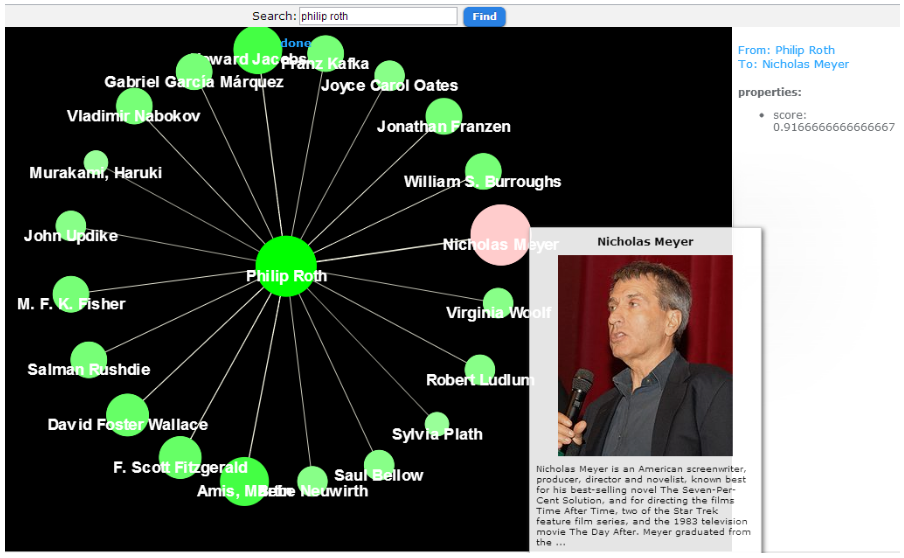
Abstract. We present LODPeas: a system for browsing entities that are found to share many things in common in an RDF dataset. The system first offers standard keyword search to locate a *focus entity*. Once a focus entity has been found, other entities that share a lot in common with it are displayed in a graph-based visualisation. The degree to which two entities have a lot in common—their level of *concurrence*—is scored by looking at attributes (property–value pairs) that they share: attributes that are shared by few other entities are given higher weight, and additional shared attributes imply a stronger score. LODPeas is designed to scale for billions of triples and is built in an (almost) entirely domain-agnostic fashion, built on top of the RDF standards themselves and not requiring any domain-specific input. Herein, we describe the functionality of LODPeas, how the system was built over the BTC'12, and discuss possible applications.

1 Overview

Though there have been some successful domain-specific or domain-selective applications created from/for Linked Data, generic applications operating over arbitrary collections of Linked Data (see [2, § 6.1.1.]) must face a number of challenges and suffer from a number of problems, including but not limited to:

1. the system is unintuitive to use:
   (a) the interface requires knowledge of syntax,
   (b) the user interface is too idiomatic to understand (esp. for users new to RDF);
2. the data are messy and unknown:
   (a) results may be unintuitive, incorrect, or missing,
   (b) human-readable information is only partly available,
   (c) the user does not know what kind of content is in there and as such, does not know what tasks can be performed, let alone how they can be performed;
3. miscellaneous, including slow response times, etc.

In particular, problems with data quality are a huge obstacle to overcome. In general, we believe that generic applications should avoid inundating users with vast quantities of diverse/messy data and complex interaction models over such data. Instead, we believe that generic applications should rather leverage the structure of the data to provide intuitive interactions over aggregate or



From: Philip Roth
To: Nicholas Meyer

properties:

- score: 0.9166666666666667

Nicholas Meyer

Nicholas Meyer is an American screenwriter, producer, director and novelist, known best for his best-selling novel The Seven-Per-Cent Solution, and for directing the films Time After Time, two of the Star Trek feature film series, and the 1983 television movie The Day After. Meyer graduated from the ...

Fig. 1. Screenshot of result for search `"philip roth"` showing details of the most closely related entity (being hovered over), "Nicholas Meyer"

meta-information derived from the description of groups of entities. Put another way, we see potential in generic applications that offer a different perspective to mainstream sites: a perspective built from *aggregate* views over *multiple* entities, their implicit context, relationships, etc. Importantly, views are computed from RDF data, not composed (primarily) of RDF data: hence, some of the noise and diversity of the data are (mercifully) hidden from the user.[1]

Herein, we present an application called LODPeas that computes views of similar entities—entities that share a lot in common—from the RDF data provided for the BTC'12 challenge, and use the results to power a visualisation tool. The core functionality of the system is straightforward: the user enters a keyword search for an entity and receives back a result pane as depicted in Figure 1. The result pane contains various "*peas*", each referring to a particular entity. The central/focus entity is that which was searched for (in this case "Philip Roth"). The surrounding entity peas (top-20) are those computed from the underlying RDF data to share a lot in common with the focus entity. The colouring and size of the peas indicate how much they share in common with the focus entity. Hovering over a pea gives a brief overview of the entity (where available) including a name, image and brief comment ($<$ 300 characters); this is shown in Figure 1 for Nicholas Meyer. Various parts of the graph visualisation can be hovered over to find out more information, including the edges present between peas. Surrounding peas can be clicked to make them the focus pea.

[1] As opposed to, e.g., our own system: `http://swse.deri.org/`.

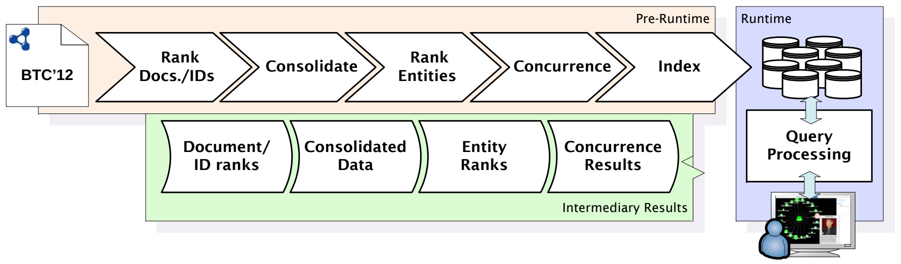


Fig. 2. Processing pipeline for building the LODPeas application from the BTC'12 data.

Herein, we first provide a brief overview of the processing and indexing steps involved in generating this prototype application from the BTC'12 dataset. We then describe actions that the underlying index scheme allows. Finally, we summarise how the system meets the requirements of the BTC'12 challenge.

An online working prototype of the system is available to use at `http://deri-srvgal28.nuigalway.ie/lodpeas/`**.** We intend to refine, improve and extend this system in the coming months.

2 Processing/Indexing the BTC'12

In Figure 2, we present the pipeline for processing the BTC'12 and creating the LODPeas prototype. We now briefly provide an informal description of each step. For the pre-processing and indexing phases, we re-use a lot of the SWSE architecture, which is built largely on top of standard Java libraries and which we have been developing (on and off) for the past seven years [3].

We process the entire BTC'12 corpus, but with one exception: we filter out RDF data produced internally within the crawler to represent response HTTP headers; these meta-information were embedded in the primary BTC'12 data, but are not part of the Web content itself. We develop our methods to deal with static data, bulk sorts and bulk loading. Though we do not discuss runtimes in detail, as a rough indication, the pre-processing phase took about one week on a quad-core server with 64GB of RAM.

2.1 Ranking Documents and IDs

Given a heterogeneous corpus of Web data, ranking is an important factor to decide which elements—be it documents, entities, properties, identifiers, etc.—have the most prominence. Ranking is particular important for the selective display and prioritisation of results for the end user.

In the first step, we apply a PageRank-style analysis over the documents in the corpus. We consider a link from one RDF document to another if and only if the source document mentions a URI (in any position of a triple) that dereferences to the target document. We thus build a graph of interlinked documents, considering only links between documents included within the corpus itself (i.e., we prune "deadlinks"). We then compute a PageRank score for each document (the details of this process are available in more detail in [3]). In the random surfer model of PageRank, the score for each document indicates (roughly) the probability of an agent traversing RDF documents through dereferenceable links being at that particular document at a given point after a long walk.

In the second step, we rank the RDF terms appearing in the data. The rank of a term can be succinctly defined as the sum of the PageRank of all the documents it is mentioned in. This score indicates the probability that the random surfer is—at some particular time after a sufficiently long walk—at a document that mentions that term. We only rank URIs and blank-nodes, where we use these term scores primarily in the next consolidation/canonicalisation phase.

2.2 Consolidation

Given that the data from the corpus come from multitudinous sources that may implement different naming schemes, we next *consolidate* the data to merge information about entities presented under different identifiers. In particular, we consolidate the data based on explicit `owl:sameAs` relations found within the corpus itself. We first scan the data to extract all such relations from the data and load them into a special in-memory index structure (details in [3]), which implicitly performs the transitive and symmetric closure of the relation using an inherent Union-Find algorithm. The result is a map from RDF terms (blank nodes and literals) to their set of aliases. Then, for each individual set of aliases, we select a canonical identifier (a pivot element) based on the highest rank term found from the previous phase. This ensures that the canonical identifier selected for the entity is the most "recognisable" one. In the second phase, we scan all of the data again and rewrite aliases (in the subject position or the object position of non-`rdf:type` triples) to their canonical form.

After consolidation, some aliases may remain unconsolidated. In other works, we have investigated using richer OWL semantics to perform consolidation including functional and inverse-functional properties [5]; we may incorporate these methods again at a later stage. Finally, we recognise that many readers may be concerned about the quality of `owl:sameAs` relations found in the wild, as per analyses such as by Halpin et al. [1]. Though we did encounter some nasty examples of erroneous consolidation (particular drug-related RDF data; cf. [5]), we generally find the precision of such relations to be quite high (we reported 97.2% precision after manual inspection in previous work [5]).[2]

2.3 Ranking Entities

Now that entities have been consolidated under respectively selected canonical identifiers, we can rank the entities themselves: we simply re-run the term

[2] The proof is in the pudding: `http://deri-srvgal28.nuigalway.ie/lodpeas/`

ranking procedure over the *consolidated* data to get the ranks of identifiers for different entities. These are used later to prioritise presentation of entities that are often mentioned in highly ranked documents.

2.4 Concurrence

In the next phase, we wish to rank how much various entities share in common. To do this, we use an existing method—previously introduced by us in [5,4]—called *concurrence*. The core intuition is straightforward: we compute a concurrence score between pairs of entities that increases (monotonically) for each property–value that they share in common. Property–value pairs are directionless, and can refer to predicate–object pairs shared by both entities, or predicate–subject pairs that form "inlinks" shared by both entities. The fewer other entities that share a property–value pair, the more "selective" the pair is considered and the higher the concurrence score boost: if n entities share a property–value pair, then they receive a $\frac{1}{n}$ pairwise boost to their concurrence score. When entities share multiple property–values in common, each individual boost score is aggregated into an overall concurrence score using a probabilistic sum. More details on the concurrence scoring and the computation thereof is available in [5,4]. Herein, it is sufficient to consider concurrence as a scoring of how much two entities share in common, which is proportionate to how exclusive those shared things are and how many they are.[3]

3 Indexing and User Interface

Having computed all of the interim results and performed the pre-processing necessary to power the final prototype, we then build the following indexes so as to support lookups for the required functionalities:

Inverted Keyword Index: This store maps keyword queries to a ranked list of entities. We use the popular Lucene software for this index. For each (consolidated) entity, we build a virtual document of text from the literal string values attached directly to it as object values. We store the content of this virtual document as an inverted index against the respective entity identifier. In terms of ranking elements, we index *label* (or name) literals with a high boost value to boost their relevance to the keyword query. Furthermore, we set a boost score for each entity as its rank (computed two phases previous). In each virtual document, we also embed meta-information about the preferred label and comment for display, a list of associated images (detected by analysing the file extension on object values) and ranking information. This allows us to generate a "snippet" view directly from keyword matches.

[3] For scalability reasons, we do not consider property–value pairs that are shared by more than 50 entities ($n > 50$) since the outcome is quadratic. However, these pairs would mostly have minimal effect on concurrence ($\frac{1}{50}$).

Quadruple Index: This store maps a (consolidated) subject to all quadruples about it, and all original aliases found for it. As such, the focus result index can be used to retrieve all RDF triples and their provenance available about a given entity (subject). For this, we use simple blocked and compressed files and skip lists, as used in SWSE (detailed in [3]). For rendering quadruple information, we also need labels and ranks for predicates and classes; these are merged into the block of results for an entity such that the entire package of data need to represent an entity can be retrieved in one index lookup (i.e., no joins are required).

Concurrence Index: Using the same low-level indexing scheme as for quads, this indexes the concurrence results computed in the previous step: for each entity, a ranked list of concurrent entities is indexed, along with a list of the shared property–value pairs that lead to the score and their individual weights. As such, given an entity identifier, in a single lookup we can retrieve all other entities that are concurrent with it ordered by concurrence score, the property–value pairs that the entities share in common, the weights for property–value pairs used in scoring, and the labels and term-ranks for the properties and values. This allows us to retrieve all information to generate the view of entities sharing a lot in common and to explain results to the user.

Importantly, to improve system response times and "interactivity", our indexes pre-compute all of the (static) blocks of data necessary to directly power user-interface interactions, thus avoiding joins and minimising the amount of lookups required. A query processor is then implemented as a Java Servlet, sitting on top of the three indexes and servicing HTTP `GET` requests. The user-interface itself is built using JSP and Javascript code that communicates with the underlying query processor on the server, where the "peas interface" (as shown in Figure 1) is rendered using the JavaScript InfoVis Toolkit (JIT) library.

4 LODPeas as a BTC'12 Challenge Entry

Herein, we discuss in detail how the LODPeas entry meets the BTC'12 challenge criteria.[4]

4.1 Minimal Requirements

The primary goal of the Billion Triple track is to demonstrate applications that can work on Web scale using realistic Web-quality data. / The tool or application has to make use of at least the first billion triples from the data provided by the organizers. It is desired that the tool or application uses the complete data set. We meet this requirement by using the entire BTC'12 dataset as provided by the challenge organisers (with the minor exception of filtering irrelevant HTTP header information written by the crawler software itself).

[4] http://challenge.semanticweb.org/2012/criteria.html

The functionality of the applications is left open: for example it could involve helping people figure out what is in the data set via browsing, visualization, profiling, etc., or inferencing that adds information not directly queryable in the original data set. We focus on the browsing and visualisation of entities within the corpus.

The tool or application does not have to be specifically an end-user application, as defined for the Open Track Challenge, but usability is a concern. The key goal is to demonstrate an interaction with the large data set driven by a user or an application. We make an end-user application interface available. We intend to further extend the current prototype over the coming months.

4.2 Additional Desirable Features

The application should do more than simply store/retrieve large numbers of triples. We believe that the generic idea of using RDF to generate a visualisation of entities that share many aspects common is a novel idea, and goes beyond a typical "engineering" challenge.

The application or tool(s) should be scalable (in terms of the amount of data used and in terms of distributed components working together). All methods presented are scalable and are demonstrated over the full BTC'12 dataset. We build read-optimised indexes. Previous works have shown in detail how the data-processing, indexing and query-processing primitives introduced herein can be distributed over shared nothing commodity hardware [3,5].

The application or tool(s) should show the use of the very large, mixed quality data set. This is also met through use of the BTC'12 dataset. It is important to note that our system uses no input other than the BTC'12 dataset and axiomatises nothing further than the details of (Semantic) Web standards themselves. We do not hard-code in specific treatment for any vocabulary or domain, and our methods can be ported to any arbitrary RDF dataset.[5] Interestingly, the diversity of the BTC'12 helps demonstrate the flexibility of our system; to take a few examples:

– searching for entities like cars (e.g., `toyota aygo`) will reveal other cars with similar specs detailed in DBpedia, Freebase; searching for entities like music bands (e.g., `animal collective`) or actors (e.g., `stan laurel`) will reveal performers that have co-starred or collaborated together from sources including DBpedia, Freebase, LinkedMDB, BBC;

[5] We make one small cheat where we manually add some common label and comment properties that are not automatically found to be sub-properties of the core `rdfs:label`/`rdfs:comment` properties. Primarily, the manually added properties are taken from the DC vocabulary. (Relevant properties from vocabularies such as FOAF and SKOS are stated in the data to be sub-properties of their RDFS counterparts.)

– searching for people (e.g., `marcelo arenas`) in the SW community will reveal people with which they share many publications in common (from the `semanticweb.org` "DogFood" domain); searching for papers (e.g., `towards a dynamic linked data observatory`) will reveal related papers by some of the same authors.
– searching for proteins (e.g., `lrrc8a`) reveals related proteins and other entities associated with the same illnesses or genes from sources like DBpedia, Bio2RDF, etc.

Of course, despite our best efforts to rank and organise data, noise is still present in the system (e.g., try `tim berners lee`). Still, examples such as those listed above prove the concept of LODPeas as a generic/flexible Semantic Web application that can handle diverse data elegantly and provide interesting insights about entities within or across multiple domains.

The application should either function in real-time or, if pre-computation is needed, have a real-time realization (but we will take a wide view of "real time" depending on the scale of what is done). We provide a real-time interactive interface at http://deri-srvgal28.nuigalway.ie/lodpeas/.

References

1. H. Halpin, P. J. Hayes, J. P. McCusker, D. L. McGuinness, and H. S. Thompson. When `owl:sameAs` isn't the same: An analysis of identity in Linked Data. In *International Semantic Web Conference (1)*, pages 305–320, 2010.
2. T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space (1st Edition)*, volume 1 of *Synthesis Lectures on the Semantic Web: Theory and Technology*. Morgan & Claypool, 2011. Available from http://linkeddatabook.com/editions/1.0/.
3. A. Hogan, A. Harth, J. Umbrich, S. Kinsella, A. Polleres, and S. Decker. Searching and browsing Linked Data with SWSE: The Semantic Web Search Engine. *J. Web Sem.*, 9(4):365–401, 2011.
4. A. Hogan, M. Mellotte, G. Powell, and D. Stampouli. Towards fuzzy query-relaxation for RDF. In *ESWC*, pages 687–702, 2012.
5. A. Hogan, A. Zimmermann, J. Umbrich, A. Polleres, and S. Decker. Scalable and distributed methods for entity matching, consolidation and disambiguation over Linked Data corpora. *J. Web Sem.*, 10:76–110, 2012.