Generating Examples of Paths Summarizing RDF Datasets

Jindřich Mynarz
Department of Information and Knowledge Engineering, University of Economics
W. Churchill Sq. 4
130 67 Prague, Czech Republic
jindrich.mynarz@vse.cz

Marek Dudáš
Department of Information and Knowledge Engineering, University of Economics
W. Churchill Sq. 4
130 67 Prague, Czech Republic
marek.dudas@vse.cz

Paolo Tomeo
SisInf Lab, Polytechnic University of Bari
Via Orabona 4
70125 Bari, Italy
paolo.tomeo@poliba.it

Vojtěch Svátek
Department of Information and Knowledge Engineering, University of Economics
W. Churchill Sq. 4
130 67 Prague, Czech Republic
svatek@vse.cz

ABSTRACT

As datasets become too large to be comprehended directly, a need for data summarization arises. A data summary can present typical patterns commonly found in a dataset, from which high-level understanding of the data can be obtained. Nonetheless, such abstract understanding can be improved by providing concrete examples of the summary patterns. If possible, the chosen examples should be diverse and representative of the patterns they instantiate. In this paper, we present three methods for generating examples of patterns discovered in RDF datasets. The patterns we consider are the most frequent path graphs that consist of classes of instances or data types of literals connected by RDF properties. We propose an RDF/S vocabulary for describing these path graphs and their instances. We present three methods for generating path examples, namely random, distinct, and representative selection, that are based on randomization, diversification, and clustering.

CCS Concepts

•**Information systems** → **Summarization;** *Resource Description Framework (RDF);*

Keywords

summarization, diversity, clustering, RDF

1. INTRODUCTION

Many datasets nowadays are too large to comprehend by examining all their data. Fortunately, most large datasets exhibit regular structure. Partial comprehension of such datasets can be derived from the common patterns manifested in their structure. In a way, a dataset's structure provides a high-level summary of the dataset's content. Nevertheless, discovering a dataset's structure is somewhat more difficult with data formalized using the Resource Description Framework (RDF) [5]. RDF is a graph data model that decomposes data into binary relations represented as RDF triples. RDF datasets usually do not have a schema fixed up-front. While traditional relational databases enforce ex-ante schema, schema in RDF datasets often emerges only ex-post as a result of combination of several RDF vocabularies and their specific way of use. The schemas of RDF datasets are thus descriptive rather than prescriptive.

Since schemata of RDF datasets are not formalized before use, they typically need to be discovered from data via statistical inference. An example of a tool that discovers and summarizes the structure of RDF datasets is LODSight [7]. LODSight offers a node-link visualization of the most common path graphs found in a summary of an RDF dataset. Path graphs are trees with n nodes, such that 2 nodes have vertex degree 1 and $n-2$ nodes have vertex degree 2, so that the graph can be drawn with all its nodes and edges lying in a single straight line [8, p. 18]. The nodes in these RDF paths in LODSight are types (either classes or data types), while the edges are predicates connecting instances of the types in the summarized dataset. Paths follow the direction from subject to object. For example, a path can connect the class `gr:BusinessEntity`[1] via the predicate `schema:address` to the class `schema:PostalAddress`, which is in turn connected to the data type `xsd:string` via the predicate `schema:streetAddress`. This path connects companies to their postal addresses that have literal street addresses.

The empirical schema of an RDF dataset embodied in the LODSight's paths may be rather abstract. In order to recover some of the understanding obtained by examining the data itself, it is possible to present the user of LODSight with

SEMANTICS 2016: Posters and Demos Track
September 13-14, 2016, Leipzig, Germany

[1]All namespace prefixes (`gr:`, `schema:`, etc.) used in this paper can be resolved to their corresponding namespace IRIs by using http://prefix.cc.

representative examples instantiating the patterns found in the dataset. Providing several diverse examples of a chosen pattern can complement the high-level understanding of a dataset with inductive understanding following from the concrete examples. LODSight offered a basic functionality for generating examples of RDF paths from its inception. It retrieved examples in a non-random fashion from an RDF store using the default sort order for SPARQL results, so that for each query the same examples are retrieved. In this paper we present an improvement of this functionality. It is implemented as an application that generates random and illustrative examples of RDF paths. In the following sections we present the methods used for example selection and their implementation.

Before proceeding with our work, we consider the research related to it. While a lot of related effort is dedicated to summarization of RDF datasets, generating examples from summaries is rarely covered. In cases where the research on RDF summarization is concerned with selection of the most representative features of the analysed datasets, it does so predominantly on the level of patterns instead of the level of pattern instances (e.g., [13, 15, 16]). Similar to our work, Presutti et al. [13] also propose a vocabulary for describing paths in RDF datasets.[2] Automatic selection of representative examples proposed by Böhm et al. [2] uses the resources described by the most RDF triples as representative examples linked via the `void:exampleResource` property from the Vocabulary of Interlinked Datasets [1]. To our knowledge the work of Dudáš et al. [7] on the previously-mentioned LODSight, on which this research builds, is the first to mention automatic selection of representative instances of patterns from RDF datasets.

2. EXAMPLE GENERATION METHODS

The application generating examples of RDF paths that is described in this paper offers 3 methods we developed for example selection of increasing sophistication, namely random selection, distinct selection, and representative selection. An integral part of all these methods is the representation of RDF paths, for which we propose the RDF Path vocabulary. Additionally, a fundamental part of the latter two methods is distance computation. We describe both these components before we explain how the example generation methods work.

2.1 RDF Path Vocabulary

We designed the RDF Path Vocabulary as a means to formalize the description of RDF paths. The development of a custom vocabulary was motivated by the goal of improving the representation of RDF paths. The result is a simple vocabulary[3] that is formalized using RDF Schema [3]. Unlike the similarly named RDF path languages,[4] it is a modelling vocabulary and not a query language. Paths are represented as instances of the `Path` class. Each path has at least 1 edge that instantiates the `Edge` class. A path is connected to its edges via the `edges` property. In order to maintain the order of edges, the object of this property is a collection (`rdf:Seq`) that wraps the edges constituting the path. Each edge has a starting and ending node (linked via the `start` and `end` properties, respectively) and a predicate that connects the nodes (linked by the `edgeProperty` property). There is a degree of isomorphism between the path edges and the RDF reification vocabulary [11]. In fact, the `Edge` class is derived as a subclass of `rdf:Statement`, while its properties are designed as subproperties of the properties used for reifying RDF statements. Both the start and the end node of each edge must instantiate either a class or a data type. Each instantiated class and data type must be explicitly typed as either `rdfs:Class` or `rdfs:Datatype`. Paths must be continuous, so that the edge's start node must be equal to the end node of the preceding edge. Both paths and their examples are represented as instances of the `Path` class, but they differ in the representation of nodes. While path nodes are identified by blank nodes interpreted as existentially quantified variables, nodes in path examples are concrete resources.

2.2 Distance computation

A key component of the non-random example selection is computation of distance between examples of paths. The distance of path examples is based on the distance of their constituent nodes. It is computed as a mean average of the distances of the paths' corresponding pairs of nodes. The distance is always normalized to the $[0, 1]$ interval, such that 1 is used for completely dissimilar resources, while 0 is used for equivalent resources.

In general, there are 2 kinds of node types: referents and literals. The distance function is polymorphic and dispatches based on the types of its operands. The way of computing the distance between literals is chosen based on their data types, which are either explicitly provided in the source data or inferred. A data type can be inferred by matching literal to a regular expression constraining the lexical space of the data type. If unequal data types are to be compared, we try to get their lowest common ancestor in the type hierarchy. If a common ancestor is found, the distance metric associated with it is used. For example, `xsd:negativeInteger` and `xsd:long` are compared as `xsd:integer`. We amended the hierarchy of XML Schema data types[5] to be rooted in `xsd:string`, so that the distance computation defaults to string comparison if no closer common ancestor is found.

The distance metrics are chosen based on the data types of the compared literals. For instance, numeric literals may be compared using a different metric than string literals. Some data types can be further decomposed and their distance can be computed from their constituent parts. For example, URLs can be split into domain names, protocols, port numbers etc., and their distance can be derived from the distances of their parts.

Type inference used in the distance computation supports referents and several data types defined by the XML Schema, including `xsd:decimal`, `xsd:date`, `xsd:dateTime`, `xsd:duration`, `xsd:boolean`, and `xsd:anyURI`, overall defaulting to `xsd:string`. Default distance measure for strings is the Jaro-Winkler metric, which computes the edit distance of the compared strings while taking into account their length. If the compared literals have the English language tag, the Porter stemmer is applied to them before the comparison. The distance of `xsd:anyURI` is computed as a weighted arithmetic mean of distances of the constituent URI parts measured by the Jaro-Winkler metric, so that higher weight is

[2] http://www.ontologydesignpatterns.org/ont/lod-analysis-path.owl
[3] https://w3id.org/lodsight/rdf-path
[4] https://www.w3.org/wiki/RdfPath
[5] https://www.w3.org/TR/xmlschema-2

given to more prominent parts of URIs, such as the domain name. The distance of ordinal values, such as `xsd:decimal` or `xsd:date`, is calculated as their absolute difference normalized by the maximum spread of values of the compared property p in a dataset:

$$dist_p(a, b) = \frac{|a - b|}{max_p - min_p} \quad (1)$$

The distance of non-identical referents (resources identified by IRIs or blank nodes) is computed as the distance of their descriptions. We use the concise bounded descriptions [14] of RDF resources. If IRIs of the referents are identical then their distance is 0. Otherwise, the referents are compared by value, i.e. their representation in RDF, which recursively includes representations of the linked resources if available. RDF resources are represented as sets of predicate-object pairs. We use an adjusted form of Jaccard index formalized in Equation 2 to aggregate the distances of pairs of resources a and b and sum the similarities of objects of properties found in both compared resource descriptions, subtract it from the number of distinct properties describing the objects, and divide the result by the number of distinct properties. The index was adjusted by using similarities in place of exact matches. $p(a)$ is a function that returns predicates of the resource a, while $o(a, p)$ is a function that returns objects of the resource a for the predicate p, and $dis(a, b)$ is a function that computes distance between the objects a and b.

$$d_J(a, b) = \frac{|p(a) \cup p(b)| - \sum_{i \in p(a) \cap p(b)} 1 - dis(o(a, i), o(b, i))}{|p(a) \cup p(b)|} \quad (2)$$

The computation of distance between referents is recursive. If referents are found as part of resources' descriptions, their distance is in turn computed as the distance of their descriptions. This way of distance computation can be considered as bottom-up, since the distance of referents proceeds from the distance of literals found in the referents' descriptions. The number of hops in the RDF graph that are followed in distance computation can be limited to control the runtime of example selection. In order to prevent infinite recursion, we delete cycles from the processed data prior to distance computation.

Should either of the properties shared in the compared resources have multiple objects, their distance would be computed for each combination of the objects and aggregated to minimum. The computation of pairwise distances between path examples has approximately quadratic complexity $O(\frac{n(n-1)}{2})$ based on the number of path combinations. The complexity is also determined by the path length and the number of hops followed when resolving referents.

2.3 Random selection

Our baseline approach is random selection of examples of RDF paths. Random selection provides a cursory view of the kinds of data involved in a given RDF path. Using this method we retrieve k random examples of the provided RDF path. This selection is based on random sort order of the retrieved path examples. Since the selection is random, the results may include near-duplicates or outliers, which do not represent well what the analysed dataset contains. Hence, we propose more sophisticated methods for example selection.

2.4 Distinct selection

The distinct selection method maximizes the diversity of the selected examples. Using this method we want to select k examples such that their mutual distance is maximal. In other words, *"given a set P of n items, we aim at selecting k items out of them, such that the average pairwise distance between the selected items is maximized"* [6, p. 1]. However, selecting the most diverse subset of items is known as an NP-hard problem [6, p. 1]. Thus, solutions to this task need to be approximated by using heuristics. We chose the greedy construction heuristic [6], since it achieves good diversity in short execution time. Using this heuristic we start by selecting 1 random example, then pair it with the most dissimilar example, and continue adding examples that have the highest total distance with the already included ones. To compute the distance between path examples we use the distance measure described in Section 2.2. Using this method we produce examples that provide broader coverage of an RDF path in the processed dataset than the random selection does.

2.5 Representative selection

In addition to the chosen examples being diverse we want them to be representative. The selection of examples with maximum diversity may not be the most representative of the summarized dataset. It may include misleading outliers that maximize the total distance of the selected examples. Instead, we want to pick examples that are both diverse and representative of the most common kinds of data in the analysed dataset. This reformulation of the example selection task lends itself to clustering, so that *"the goal now is to identify and recommend a set of representative items, one for each cluster, so that the average distance of each item to its representative is minimized"* [4, p. 897]. One suitable method for the selection of representative examples is k-medoids clustering. This type of clustering *"searches for k 'representative' objects called medoids, which minimize the average dissimilarity of all objects of the data set to the nearest medoid"* [9]. It offers efficient means of computing clusters from pairwise distances and produces medoids that can be considered the most representative members of the generated clusters. In fact, medoid is the *"most centrally located object in a cluster"* [12], so we select the generated medoids as the most representative examples of a given RDF path.

3. IMPLEMENTATION

Having described the methods employed for example selection, we now detail how we implemented these methods. The methods were implemented as a web service that exposes a single endpoint to which client applications may post an RDF path and get the path examples in return. The source code of the application is released as open source.[6]

We adopted JSON-LD as the data exchange format. The web service expects the RDF paths to be provided in JSON-LD syntax and the generated examples are also returned in JSON-LD. This allows us to harness the standard operations defined by the JSON-LD API [10], such as expansion, which makes it possible to coerce heterogeneous RDF graphs into

[6] https://github.com/jindrichmynarz/rdf-path-examples

regular data structures with predictable attribute names. Path examples are retrieved from SPARQL endpoints using the SPARQL 1.1 Protocol.[7] The obtained payload is converted to JSON-LD and expanded into a predictable structure. The results of example selection are serialized to JSON-LD and compacted using a JSON-LD context that is aware of the semantics of the RDF Path Vocabulary in order to ease manipulation with the results.

All the methods presented in Section 2 use random selection in SPARQL. The random method directly uses the examples generated by a SPARQL query with random order. The other methods randomly fetch a sample of path examples, out of which the k required examples are selected. The sample size can be configured by a sampling factor, which is used to multiply the desired number of examples, to obtain the size of the retrieved sample. For example, if k is 5 and the sampling factor is set to 20, then 100 path examples are retrieved.

If either the distinct or representative selection method is used, an additional SPARQL query is issued to retrieve a k-hop graph neighbourhood of the path example's nodes. A k-hop graph neighbourhood of a node is a subgraph that includes all adjacent nodes that are at most k hops away. For example, 2-hop neighbourhood contains the RDF triples in which a path node is in the subject position (1^{st} hop) plus the triples in which the subjects are objects of the triples included in the first hop (2^{nd} hop). The obtained dataset is used for computing the distances between referents used as nodes of path examples. Similarly to processing the path examples, we convert the node data into JSON-LD and expand it to obtain a predictable data structure.

The runtime of example selection is dominated by the distance computation, because the selection methods require distances between all pairs of path examples to be materialized. The runtime of example selection methods is negligible once a matrix of distances between path examples has been computed.

4. CONCLUSION

Exemplification is an important component of graph summaries of LOD datasets, as carried out by tools such as LODSight. We presented three methods for generating examples of RDF paths, which progressively build on each other. The baseline method uses simple random selection. The more sophisticated methods filter the random sample down to the most salient examples. The distinct selection method attempts to maximize the dissimilarity of the chosen examples, thus potentially covering a broader variety of instances of a given RDF path; there is however a risk of selecting extremal, outlier values. Finally, the representative selection method goes further by picking examples that are both diverse and representative of the provided path.

5. ACKNOWLEDGMENTS

This research was supported by the VŠE IGA project F4/28/2016.

6. REFERENCES

[1] K. Alexander, R. Cyganiak, M. Hausenblas, and J. Zhao. Describing linked datasets with the VoID vocabulary. W3C interest group note, W3C, 2011.

[2] C. Böhm, J. Lorey, and F. Naumann. Creating VoID descriptions for web-scale data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(3):339–345, 2011.

[3] D. Brickley and R. Guha. RDF Schema 1.1. W3C recommendation, W3C, 2014.

[4] P. Castells, N. J. Hurley, and S. Vargas. *Recommender systems handbook*, chapter Novelty and diversity in recommender systems, pages 881–918. Springer, Berlin; Heidelberg, 2^{nd} edition, 2015.

[5] R. Cyganiak, D. Wood, and M. Lanthaler. RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C, 2014.

[6] M. Drosou and E. Pitoura. Comparing diversity heuristics. Technical Report TR-2009-05, University of Ioannina, 2009.

[7] M. Dudáš, V. Svátek, and J. Mynarz. Dataset summary visualization with LODSight. In *The Semantic Web: ESWC 2015 Satellite Events. Revised Selected Papers*, pages 36–40, Berlin; Heidelberg, 2015. Springer.

[8] J. L. Gross and J. Yellen. *Graph theory and its applications*. CRC Press, Boca Raton (FL), 2^{nd} edition, 2006.

[9] L. Kaufman and P. J. Rousseeuw. *Statistical data analysis based on the L1-norm and related methods*, chapter Clustering by means of medoids, pages 405–416. Elsevier, New York (NY), 1987.

[10] D. Longley, G. Kellogg, M. Lanthaler, and M. Sporny. Json-ld 1.0 processing algorithms and api. W3C recommendation, W3C, 2014.

[11] F. Manola and F. Miller. RDF primer. W3C recommendation, W3C, February 2004.

[12] H.-S. Park and C.-H. Jun. A simple and fast algorithm for K-medoids clustering. *Expert Systems with Applications*, 36:3336–3341, 2009.

[13] V. Presutti, L. Aroyo, A. Adamou, B. Schopman, A. Gangemi, and G. Schreiber. Extracting core knowledge from linked data. In *Proceedings of the Second International Workshop on Consuming Linked Data*, volume 782 of *CEUR workshop proceedings*, Aachen, 2011. RWTH Aachen University.

[14] P. Stickler. CBD: Concise bounded description. W3C member submission, W3C, 2004.

[15] G. Troullinou, H. Kondylakis, E. Daskalaki, and D. Plexousakis. Ontology understanding without tears: the summarization approach. Under review.

[16] G. Troullinou, H. Kondylakis, E. Daskalaki, and D. Plexousakis. RDF Digest: efficient summarization of RDF/S KBs. In F. Gandon, M. Sabou, H. Sack, C. d'Amato, P. Cudré-Mauroux, and A. Zimmermann, editors, *Proceedings of the 12^{th} European Semantic Web Conference*, volume 9088 of *Lecture notes in computer science*, pages 119–134, Berlin; Heidelberg, 2015. Springer.

[7] http://www.w3.org/TR/sparql11-protocol