Visualizing Implicit RDF Schema with the Help of Embeddings

Marios Meimaris
Information Management Systems Institute
Athena Research Center
m.meimaris@imis.athena-innovation.gr

ABSTRACT

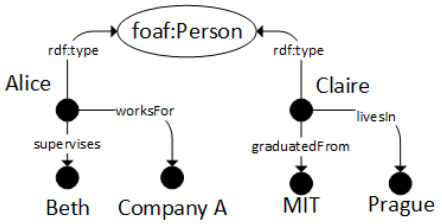
In this vision paper we present a direction for visualizing the implicit schema of a loosely-defined RDF graph in different zoom levels. RDF datasets do not necessarily adhere to strict structural guidelines, even though a schema in the traditional sense (e.g., an underlying ontology or simple *rdf:type* declarations) usually exists. Instead, there can be many different ways that instances of a particular type can appear in the same dataset. For example, two instances of the same type can have different properties, thus making their type equality semantically ambiguous. Visualizing this schema can lead to either non-intuitive depictions of the implicit types, or over-generalized visualizations that lack informative detail. In this paper, we propose our vision of an approach that tackles this issue by embedding the implicit schema into a vector space that captures generalizations and specializations of the implied types and relationships in an RDF graph.

1 INTRODUCTION

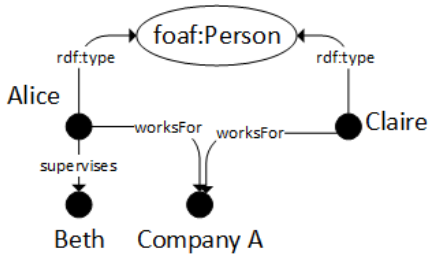
The Linked Open Data cloud offers increasingly large amounts of RDF[1] datasets, covering a wide range of domains and topics that include life sciences, demographics, government data and general encyclopedic knowledge. In contrast with the more traditional tabular data management paradigm that usually enforces structural restrictions, RDF represents relationships between concepts in the form of a graph that is often loosely defined; that is, arbitrarily diverse types of concepts and relationships can co-exist in the same RDF dataset without loss of consistency. Unfortunately, this heterogeneity in the schema, while diverse and dynamic, can lead to problems not only in storage, indexing and querying, but to schema comprehensibility and user-friendliness as well, as users cannot intuitively explore the dataset schema.

As an example, consider Figure 1(a), where two instances of *rdf:type foaf:Person* are depicted. While *Alice* and *Claire* are both instances of the *foaf:Person* class, they do not share the same properties, except for the rather generic *rdf:type* property. A question worth asking is: how do we visualize the *foaf:Person* class? In a relational context, this would amount to two different table definitions for the implied types of Alice and Claire, even though in the RDF dataset they are declared to belong in the same class. If we were to visualize just one class (i.e., *foaf:Person*), this would hide the fact that the structure of the instances of this class is vastly different[2]. Thus, it may be more intuitive to have two classes in the visualization, which would better capture the sets of relationships, or properties, coming out of each implied type.

Similarly, consider the case depicted in Figure 1(b). In this case, Alice and Claire again belong in the same class, but this time there is an overlap in their properties. More specifically, Alice and Claire both share the *worksFor* property, but Alice is also described as a supervisor of Beth. This can be interpreted as follows: the two instances belong in the same class at some height of the class's hierarchy, before they branch out to different specializations. For instance, Alice appears to be a supervisor or a manager, while Claire appears to be a more generic type of employee. Therefore, asking how to visualize the *foaf:Person* class is again worthwhile. Visualizing the schema with just one class is misleading, as there may be more hidden specializations. While this latter point can be addressed with merging/splitting the class node in different zoom levels, the examples aim to show that a somewhat trivial RDF dataset can hide more detailed schema information that what is explicitly declared.



(a) Alice and Claire have entirely different property sets



(b) Alice and Claire have overlapping property sets

Figure 1: Instances of the same class with different property sets

Recent works in the state of the art in RDF data management have shown that exploitation of an *implicit* schema of the data can be beneficial in several different directions, such as storage, querying and schema exploration [7, 9, 11, 13]. These works heavily rely on a well-known structural RDF abstraction, namely the notion of *characteristic sets*[12], i.e., the sets of unique properties that extend from all subject nodes. For example, in 1(a), the characteristic sets (CSs) of Alice and Claire are $c_1 = \{rdf : type, worksFor, supervises\}$ and $c_2 = \{rdf : type, graduatedFrom, livesIn\}$ respectively, while in 1(b), their CSs are $c_1 = \{rdf : type, worksFor, supervises\}$ and $c_2 = \{rdf : type, worksFor\}$ respectively.

In this sense, CSs can be thought of as implied types of nodes. However, they do not necessarily capture compact and comprehensible structural information. For example, some commonly

[1] https://www.w3.org/RDF/

[2] In this example, it appears that Alice is represented by her professional attributes, while Claire is not.

used and heavily cited RDF datasets include CSs in the order of $10^3 - 10^4$ [3]. Thus, while using CSs to visualize the RDF schema of such a dataset is more informative than just showing the structural definitions that explicitly exist in the dataset (e.g., all objects of *rdf:type*), such a visualization would neither be intuitive, nor comprehensible, as it would lead to huge graphs of implied types and relationships between them.

In order to both use CSs and CS relationships for schema summarization and consequent visualization, and address the issue of large amounts of implied types, in this paper we envision an approach that embeds CSs in a continuous vector space and uses the distances between vectors in order to provide a means for merging similar CSs (when zooming out) and splitting merged CSs to their different specializations (when zooming in). Our approach aims to take into account latent semantic similarities between implied types (CSs). For this reason, we propose to use a continuous vector space model where CSs can be embedded.

There are two main directions that can be followed here; first, we can use a pretrained model of word vectors, such as the Google News Corpus model, which contains a vocabulary of 3 million unique words, trained on an input corpus of 100 billion words. Second, we can build a vector space directly by embedding CSs, by following a graph embedding approach as is done in recent works[2, 15] and more recently in [16].

This paper is structured as follows. Section 2 describes the main concepts of the approach, section 3 describes different embedding methods and section 4 concludes the vision paper.

2 APPROACH OVERVIEW

In this work, we consider characteristic sets CSs as nodes in a graph, with an edge between two CSs denoting that there is a property that connects two subjects that belong in two different CSs. Formally, given an input RDF dataset D, a CS graph G_D is an ordered pair $G_D = (V_D, E_D)$, where V_D is the set of all CSs in D, and $E_D \subseteq V_D \times V_D$ is the set of all properties that connect pairs of CSs. The key idea of our approach is to extract the CSs of an RDF dataset, derive an implied schema and visualize the resulting *implied schema*. In this sense, we consider a CS to define an *implied type*, i.e., a non-explicitly defined class of instances. However, as was discussed in the introduction, this entails several problems. It does not suffice to extract all CSs and visualize them in a graph of CS nodes and CS-to-CS edges, because their heterogeneity in real-world datasets makes the approach prohibitive. It would not, for example, make sense to show 150 different CSs that all essentially describe the same class, but with slight variations in their property sets, especially when exploring the schema visually and at a distance (e.g., in high zoom levels); instead, it would be more meaningful to specialize/generalize the multiple CSs in different zoom levels. Consider the example of Figure 1(a) and 1(b). The CSs and how they are merged in different zoom levels can be seen in Figure 2.

Of course, it would not be reasonable to merge all CSs in one node at the top zoom level, because many of them describe different types, either implicit or explicit. For this reason, we need to apply a similarity metric that can be parameterized with the zoom level. Based on this metric, we can merge together only the CSs that are close to each other at each level of visualization. This procedure can be interpreted as a form of hierarchical clustering.

While graph visualization, hierarchical visualization and visual LOD exploration have been thoroughly studied recently

[3] DBPedia has more than 10,000 CSs, while Geonames and WordNet have ~1000

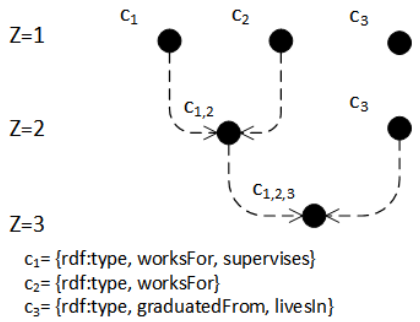


Figure 2: The CSs of the example. Dashed lines denote merging of similar CSs (i.e., generalizing). The zoom level is denoted with Z on the left.

[1, 3, 4, 14], there exist several problems when trying to merge different CSs. These have been discussed in recent works [8, 13]. These works rely on exploiting the property sets that define CSs, and merging closely related sets based on set overlaps and set hierarchies; that is, they only consider CS similarity when two CSs actually share common properties, or are subsets/supersets of one-another. However, this prevents CSs that do not share common properties to be associated, as these works mainly focus on exploiting CSs and their merging in order to derive a relational representation of RDF datasets in the hopes of improving storage and querying. In this work, we are not interested in such data management aspects and for this reason, we can loosen our definition of CS similarity and instead look for similar CSs based on their distances in a different space.

In this direction, we propose to use embeddings in order to position the CSs in a continuous space, and then rely on their geometric distances in order to attribute similarity or difference. The advantage of the use of the embedded space is that it can provide latent semantic similarity characteristics to the CSs and thus associate even seemingly unrelated CSs. For example, consider two CSs, $c_1 = \{worksFor, specializesIn, hasSalary\}$ and $c_2 = \{hasProfession, hasIncome, isEmployedAt\}$ that share no common properties. As can be seen though, there is a latent semantic association between the two CSs, as we can assume that they both describe people that are professionals. Thus, it would make sense to merge c_1 and c_2 together when visually exploring the schema at a distance. This type of similarity is easily captured with the help of embeddings [10]. In the following section, we describe two methods for embedding CSs, and discuss their tradeoffs.

3 EMBEDDING CHARACTERISTIC SETS INTO A CONTINUOUS VECTOR SPACE

The two main directions that will be pursued are (i) mapping CS nodes to an existing vector space, and (ii) embedding CS nodes directly in their own respective vector space. In what follows, these two cases will be discussed.

3.1 Mapping CSs to Word Embeddings

An embedding of an item (a word, a concept, a graph node etc.) in a continuous vector space is essentially the positioning of the item in a multidimensional space of arbitrary dimensions, where its locality is affected by the similarity of the neighboring items. In other words, the item is assigned a vector that represents its

position in the space, so that all surrounding items that are also represented by vectors are semantically similar.

There are many different ways to define similarity. One of the most adapted and promising ways is to define the similarity of two items based on their context. In the case of text, where each item is a word or a phrase, the context is defined as the word's or phrase's surrounding words. Thus, two words that commonly coexist in the same sentences will generally result in neighboring vectors. The word2vec model [10] takes into account this notion of context in order to produce vectors, and interestingly enough, it turns out that these vectors retain latent semantics that can be interpreted as geometric operations between them. A commonly cited example in this regard is something of the form $v(king) - v(man) + v(woman) = v(queen)$, which loosely means that if we replace the *man* characteristic of *king* with *woman*, we will get its female counterpart, i.e., *queen*.

In our approach, the key idea is to directly map CSs to vectors in a pre-trained linguistic vector model that already contains latent semantics for the whole English language. In order to do so, we need to follow the following steps:

(1) Assign a set of descriptive words to each CS
(2) Aggregate the vectors of the assigned words to derive a single vector
(3) Map the CS to the aggregate vector

For step (1), we can use the property set and the assigned *rdf:type* classes (if any) of the CS to come up with descriptive words. For example, for c_1 in Figure 1(b), we can use the words *person*, *works* and *supervises*. Then, for step (2), we first look up the existing word2vec model to find the vectors for *person*, *works* and *supervises*. Then, we can find a single vector representation of the three words by performing an operation such as *mean* or *sum* of their respective vectors. Finally, this aggregate vector will be the embedding of the CS. We can then define appropriate distance thresholds to find similar or dissimilar CSs.

3.1.1 Google News Corpus Example. To highlight the advantage of using such an NLP model to associate CSs in contrast with strictly relying on set operations on the CSs' property sets, let us study an example. As a pretrained NLP model, we will use the vectors created by word2vec for the Google News Corpus. This model contains vectors for a vocabulary of 3 million English words in a 300-dimensional continuous vector space, and was trained over more than 100 billion English words from news articles. Thus, it provides ready-to-use English language vectors with general, context-independent associated semantics.

For our example, let us consider an RDF dataset, whose resulting CS graph contains the following CSs:

(1) $c_1 = \{worksFor, specializesIn, hasSalary\}$
(2) $c_2 = \{hasProfession, hasIncome, isEmployedAt\}$
(3) $c_3 = \{worksFor, name, age\}$
(4) $c_4 = \{hasSubOrganization, registeredAt, hasCEO\}$
(5) $c_5 = \{hasMinister, belongsInGovernment\}$

By inspecting the CSs, a human analyst can conclude that c_1, c_2 and c_3 describe people, while c_4 and c_5 are more related to organizations (companies and ministries). Let us also assume that the first three CSs are also associated with the *foaf:Person* class in the RDF dataset, i.e., the subject nodes of the RDF graph that belong in these CSs are of the type *foaf:Person*, while c_4 and c_5 are of types *Company* and *Ministry* respectively. In recent approaches such as [8, 13], c_1, c_2, c_4 and c_5 would be completely unrelated with each other, while c_1 and c_3 would be deemed similar, because they share one common property (*worksFor*). However, to a human analyst, several conclusions can be drawn, such as that the first three CSs define implicit types that describe some aspects of human beings, while the latter two describe some form of legal organizations.

	c1	c2	c3	c4	c5
c1	1	0.61	0.42	0.37	0.21
c2	0.61	1	0.47	0.36	0.24
c3	0.42	0.47	1	0.17	0.16
c4	0.37	0.36	0.17	1	0.56
c5	0.21	0.24	0.16	0.56	1

Table 1: Pair-wise cosine similarities of mean vectors for $c_1 - c_5$

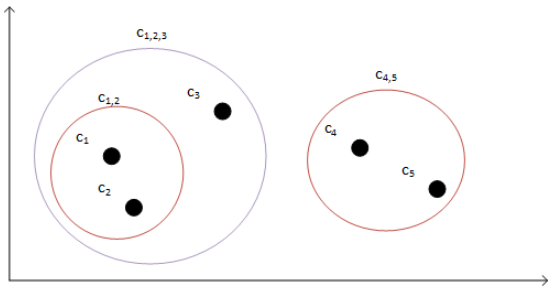


Figure 3: Visualization of the implied types in different zoom levels. In the lower zoom level, each CS is shown individually. In the middle zoom level, c_1, c_2 are grouped together, as are c_4, c_5. In the higher zoom level, c_1, c_2, c_3 are all grouped together to represent the Person type.

After some preprocessing for step 1 that was described before, we can derive descriptive words for the five CSs. Let us assume that these are c_1 : {*works*, *specializes*, *salary*, *person*}, c_2 : {*profession*, *income*, *employed*, *person*}, c_3 : {*works*, *name*, *age*, *person*}, c_4 : {*suborganization*, *registered*, *CEO*, *company*} and c_5 : {*minister*, *government*, *ministry*}. Feeding these into the pretrained Google News word2vec model and computing the mean vector for each of the five sets, we get the pair-wise similarity matrix shown in Table 1.

As can be seen, c_1 and c_2 are more closely related with each other than with c_3, even though they do not share any common properties. Interestingly, c_1 and c_2 are more similar than c_1 and c_3 even though the latter actually share a common property. Furthermore, c_1, c_2c_3 are closer with each other than with c_4, c_5, while c_4 and c_5 are more similar to each other than with the rest of the CSs. This intuitive knowledge that can be used to cluster together similar CSs based on their latent semantics would not have been captured using traditional methods that rely on property set operations. Using this approach, a potential visualization of this schema in different zoom levels can be seen in Figure 3.

Concluding this example, it should be mentioned that in order to serve more context-dependent data, such as specialized scientific data, we can define and configure the training corpus accordingly. For example, for life sciences datasets, it might not make much sense to use the Google News Corpus. Instead, it could be more useful to build a corpus by combining life science text sources, such as UMLS, Drugbank and related Wikipedia subsets, than rely on a generic and multi-purpose vector model.

3.2 Using graph embedding techniques

A recently growing body of work focuses on so called graph embeddings. The objective of these approaches is to embed a graph of nodes and edges into a continuous vector space, while preserving the inherent structure of the graph. Most of the related work focuses on preserving this structure for the nodes of the input graph [2, 5, 15], however, recent approaches also focus on encoding relationships (i.e., edges) between the graph's nodes. For example, in [16] the authors define the *context* of a node in an RDF dataset as the nodes that are adjacent to this node by either incoming or outgoing edges, and they construct the vector model by minimizing a customized error function based on this context. As a consequence, the properties (i.e., edges) of the RDF graph become mathematical translations between two nodes, much like the *king* and *queen* example of word2vec.

However, these approaches operate on the instance level and are not applied to the schema of the graph, as is our case. As a consequence, they train on the whole dataset's instance, and thus they rely on much larger corpora to build their vector models, in contrast with our case where the implied schema is usually orders of magnitude smaller than the size of the dataset. Thus, the encoded information and relationships will be supported by an abundance of data and hence will potentially be more accurate.

On the other hand, training on just the CS graph can be a computational advantage of using graph embedding techniques, in that the training of the model and construction of the vector space will be based only on the input CS graph, where in our case it is expected to be small and proportional to the amount of CSs in a dataset[4]. Thus, it would not be computationally costly to train the vectors or perform neighbourhood queries for a given CS. A problem with this approach, however, is that the output model is dependent on the input and will be very limited in discovering latent semantic similarities between seemingly unrelated CSs.

4 CONSIDERATIONS AND FUTURE WORK

In this vision paper, we have presented a potential approach for visualizing a loosely-defined RDF schema by combining characteristic sets with embeddings. To this end, we have proposed two directions, (i) by using NLP embedding models such as word2vec, and (ii) by using graph embedding techniques. We focus our vision on taking advantage of the latent semantics that can be provided by embeddings in order to provide visual generalizations and specializations of an RDF dataset's implied types in different zoom levels.

So far, we have only discussed grouping and visualizing implied types based on CSs. In the case of *relationships* between CSs, we can perform the same steps, but this time instead of CSs we will consider groups of properties that connect two CSs in the dataset. Then, we can group/ungroup the resulting edges accordingly, based on the same methodology as above.

Another point that requires addressing is the fact that proper differentiation of the implied types is not possible by just relying on metric distances. For example, the cosine similarity between c_1 and c_4, i.e., between human professionals and companies in our example is actually not very different than the similarity between c_1 and c_3. Thus, we would not be able to place humans and organizations in different groups by just relying on their similarity in the vector space. This would require auxiliary knowledge or better vector assignment. In our case, the existence of *rdf:type* properties can serve as the auxiliary knowledge. However, this information will not always be available, thus this point needs proper addressing in the future.

Finally, grouping at different levels will be addressed by an appropriate clustering approach, which will require tuning of the required hierarchical thresholds, or the ranges of similarities serving each zoom level. For example, in preliminary experimentation, trivial methods such as linear partitioning of similarity ranges can be used in order to assign ranges of similarity for different zoom levels (e.g., for the lowest zoom level we group together all CSs with centroid-based similarity larger than 0.9), and loosen this value the higher we go in the zoom level. As a final note, appropriate hierarchical or centroid-based clustering methods will be studied for the above purposes.

In conclusion, it is worth mentioning that the proposed approach can be applied in more fields than schema visualization. Schema summarization, indexing and query planning are a few domains that can take advantage of the proposed technique. For example, in [6] the authors utilize CSs and their relationships between them in order to provide better join cardinality estimations for query planning. Furthermore, this approach can be complemented by the CS merging works mentioned earlier to provide better grouping of implied types in different zoom levels.

Acknowledgements. We acknowledge support of this work by the project "Moving from Big Data Management to Data Science" (MIS 5002437/3) which is implemented under the Action "Reinforcement of the Research and Innovation Infrastructure", funded by the Operational Programme "Competitiveness, Entrepreneurship and Innovation" (NSRF 2014-2020) and co-financed by Greece and the European Union (European Regional Development Fund).

[4]Theoretically, a CS graph with n CSs can have up to n^2 edges, but this would mean that each implied type of concept is associated with all of the rest of the types, which is not realistically expected in common datasets

REFERENCES

[1] N. Bikakis and T. Sellis. 2016. Exploration and Visualization in the Web of Big Linked Data: A Survey of the State of the Art. In *LWDM*.
[2] M. Cochez, P. Ristoski, S. P. Ponzetto, and H. Paulheim. 2017. Biased graph walks for RDF graph embeddings. In *WIMS*. ACM.
[3] N. Bikakis et al. 2016. graphVizdb: A scalable platform for interactive large graph visualization. In *ICDE*.
[4] N. Bikakis et al. 2017. A hierarchical aggregation framework for efficient multilevel visual exploration and analysis. *Semantic Web* (2017).
[5] A. Grover and J. Leskovec. 2016. node2vec: Scalable feature learning for networks. In *ACM SIGKDD*. ACM.
[6] A. Gubichev and T. Neumann. 2014. Exploiting the query structure for efficient join ordering in SPARQL queries. In *EDBT*.
[7] M. Meimaris and G. Papastefanatos. 2016. Double Chain-Star: an RDF indexing scheme for fast processing of SPARQL joins.. In *EDBT*.
[8] M. Meimaris and G. Papastefanatos. 2018. Hierarchical Characteristic Set Merging for Optimizing SPARQL Queries in Heterogeneous RDF. *arXiv preprint arXiv:1809.02345* (2018).
[9] M. Meimaris, G. Papastefanatos, N. Mamoulis, and I. Anagnostopoulos. 2017. Extended Characteristic Sets: Graph Indexing for SPARQL Query Optimization. In *ICDE*.
[10] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, and J. Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*.
[11] G. Montoya, H. Skaf-Molli, and K. Hose. 2017. The Odyssey Approach for Optimizing Federated SPARQL Queries. (2017).
[12] T. Neumann and G. Moerkotte. 2011. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In *ICDE*.
[13] MD Pham and P. Boncz. 2016. Exploiting emergent schemas to make RDF systems more efficient. In *ISWC*.
[14] L. Po and D. Malvezzi. 2018. Community Detection Applied on Big Linked Data. *Journal of Universal Computer Science* (2018).
[15] P. Ristoski and H. Paulheim. 2016. Rdf2vec: Rdf graph embeddings for data mining. In *ISWC*. Springer.
[16] M. Wang, J. Wang, R.and Liu, Y. Chen, L. Zhang, and G. Qi. 2018. Towards Empty Answers in SPARQL: Approximating Querying with RDF Embedding. In *ISWC*. Springer.