What is the Schema of your Knowledge Graph? Leveraging Knowledge Graph Embeddings and Clustering for Expressive Taxonomy Learning

Amal Zouaq
amal.zouaq@polymtl.ca
Polytechnique Montréal
Montréal, Québec, Canada

Félix Martel
felix.martel@polymtl.ca
Polytechnique Montréal
Montréal, Québec, Canada

ABSTRACT

Large-scale knowledge graphs have become prevalent on the Web and have demonstrated their usefulness for several tasks. One challenge associated to knowledge graphs is the necessity to keep a knowledge graph schema (which is generally manually defined) that accurately reflects the knowledge graph content. In this paper, we present an approach that extracts an expressive taxonomy based on knowledge graph embeddings, linked data statistics and clustering. Our results show that the learned taxonomy is not only able to retain original classes but also identifies new classes, thus giving an up-to-date view of the knowledge graph.

CCS CONCEPTS

• **Information systems** → **Clustering**; • **Computing methodologies** → **Ontology engineering**.

KEYWORDS

Knowledge graph embeddings, expressive taxonomies, ontology learning, hierarchical clustering.

ACM Reference Format:
Amal Zouaq and Félix Martel. 2020. What is the Schema of your Knowledge Graph? Leveraging Knowledge Graph Embeddings and Clustering for Expressive Taxonomy Learning . In *Semantic Big Data (SBD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3391274.3393637

1 INTRODUCTION

The emergence of knowledge graphs as a structured knowledge representation has seen tremendous developments in academia and industry in the last years [7]. Knowledge graphs are composed of facts, consisting of triples in the form (head entity, property, tail entity). Knowledge graphs often rely on a manually-built schema that represents types or classes, which gives a well-defined meaning to the facts they contain. The recent development of knowledge graphs has given rise to several interesting applications for question answering, information retrieval, recommendations, and more generally for problem solving. In parallel, we have also witnessed the incredible success of neural networks that require continuous representations. Knowledge graph embeddings arise, among other reasons, from the interest of exploiting knowledge graphs in neural models. These embeddings map entities and relations to low-dimensional vector representations and the general assumption is that the proximity of some given embeddings in a vector space implies their semantic proximity. However, once a group of embeddings appears as a cluster, one challenge is to be able to label this cluster. Another challenge that is related to knowledge graphs in general is that their ontological structure or schema is defined manually a priori. Thus, it is hard to identify what type of new data is injected in the knowledge graph, whether some unknown classes of entities emerge, and how to query and formalize the given schema of a dynamic knowledge graph at any point in time. In this paper, we seek to answer the following research question: How can we derive an ontological structure from knowledge graph embeddings?

In particular, our goal is to investigate several knowledge graph embedding models and evaluate their appropriateness for the identification of a knowledge graph current taxonomy. Based on these embeddings, we propose a method that clusters automatically knowledge graph entities and rely on the statistical distribution of classes and properties to label each cluster. Next, the hierarchical clustering combined with these frequent axioms are used to generate an expressive taxonomy.

The paper is structured as follows. Section 2 presents a background and state of the art for the generation of schemas for knowledge graphs. In section 3, we present the various knowledge graph models used in this study and we describe an evaluation task on the ability of embedding models to separate different classes. Finally, we explain our methodology to generate an expressive taxonomy using hierarchical clustering, statistics and cluster labelling in section 4.

2 RELATED WORK

2.1 Knowledge Graphs Schema Extraction

The extraction of a knowledge graph structure or schema from data is not a new task and has been tackled under the name of ontology learning or taxonomy extraction for the last decade. This task can be decomposed into learning simpler components such as classes, taxonomical relations, properties, complex axioms or rules from various data sources such as corpora or knowledge graph triples. Ontology learning in its more general form has been used to generate an ontology from scratch [14, 21], to populate an ontology [4, 13] or to complete an ontology [8]. Various techniques have

SBD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-7974-8/20/06…$15.00
https://doi.org/10.1145/3391274.3393637

been proposed such as linguistic patterns [3], statistical distributions over linked data [12], formal concept analysis [5], inductive logic programming [2] or machine learning [14]. While knowledge graph representation learning has emerged in several applications such as knowledge base completion [11] or link prediction [17], it has seldom been used, to our knowledge, for the task of schema generation or mining. Given the aim of these models to produce low-dimensional vectors that group semantically-similar entities in the same regions of the space, this property can be effectively exploited by a clustering algorithm to identify classes of entities in a dynamic knowledge graph. In this paper, we focus on the comparison of various knowledge graph embedding models in pointwise Euclidean space and complex space for the task of separability, that is the ability of linearly separating instances of different classes based on these embeddings. We then use the best models to generate a taxonomy from a knowledge graph.

2.2 Taxonomy Induction

The use of low-dimensional knowledge graph embeddings has only been recently considered for the task of schema induction and more precisely taxonomy extraction. The most similar work to the one proposed in this paper, TIEmb [15], proposes the extraction of a taxonomy using the RDF2Vec embeddings of DBpedia. TIEmb determines the inclusion of a cluster inside another one based on the centroid and radius of these clusters. In this paper, we exploit hierarchical clustering to automatically identify these inclusion axioms. We also do not only generate a taxonomy of named classes but label clusters with complex definitions based on conjunctions and disjunctions of atomic concepts and restriction axioms. This allows us to discover new classes that were not previously defined in the original knowledge graph schema and to identify triples that are incomplete (e.g. with undefined property values).

3 KNOWLEDGE GRAPH EMBEDDINGS

3.1 Embedding Models

In this paper, we experiment with six embedding models that represent entities in Point-wise Euclidean space and Complex space: TransE [1], TransH [19], TransD [6], DistMult [20], ComplEx [18] and RDF2Vec [16].

TransE, TransH and TransD. The geometric model behind TransE [1] is that a relation r can be seen as a translation in the Euclidean space : a triple (h, r, t) is valid if $\mathbf{t}$ is the translation of $\mathbf{h}$ by the vector $\mathbf{r}$. This results in the following energy function:

$$E(h, r, t) = \|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_2 \quad (1)$$

As analyzed in [19], the geometric assumptions behind TransE are very strong and thus cannot handle properly all types of relations. For example, if r is a reflexive relation (meaning that (h, r, t) implies (t, r, h)), then the model will be trained to have both $h+r \approx t$ and $t + r \approx h$, i.e. $h \approx t$ and $r \approx 0$. To overcome these limitations, many extensions have been proposed, including TransH [19] and TransD [6]. In these two models, a relation is still modelled as a translation in the vector space, but this translation does not operate directly on the entity embeddings. Entity embeddings are first transformed, then translated. In TransH, embeddings are projected to a hyperplane H_r specific to relation r. The hyperplane H_r is defined by its normal vector $w_r \in \mathbb{R}^d$ and is learnt jointly with h, r, t. The projection $u_\perp$ of a vector $u \in \mathbb{R}^d$ on H_r is defined by:

$$u_\perp = u - (w_r^\top u) \cdot w_r \quad (2)$$

And the energy function of a triple is:

$$E(h, r, t) = \|\mathbf{h}_\perp + \mathbf{r} - \mathbf{t}_\perp\|_2 \quad (3)$$

Instead of a projection, TransD uses a linear transformation M, with M a transformation matrix depending on both the entity and the relation. For each entity or relation $u \in \mathcal{E} \cup \mathcal{R}$, the model learns two vectors of R^d: the embedding u itself, and a projection vector u_p. Then, the transformation matrix for an entity e and a relation r is dynamically built as: $M_{e,r} = r_p \cdot {e_p}^\top + I^{d \times d}$. The entity embedding can then be transformed using $e_r = M_{e,r} \cdot e$ and the energy function of a triple is :

$$E(h, r, t) = \|\mathbf{h_r} + \mathbf{r} - \mathbf{t_r}\|_2 \quad (4)$$

DistMult. In TransE, TransH and TransD, the embeddings of h, r, t are added or subtracted to/from each other. DistMult [20] proposes a multiplicative combination of these embeddings, with a score function:

$$\sigma(h, r, t) = h^\top D_r t \quad (5)$$

With D_r a diagonal matrix. The authors indicate that such multiplicative composition gives more expressivity to the model while keeping the same number of parameters as TransE.

ComplEX. ComplEX [18] extends DistMult to the complex space: h, r, t are in C^d instead of R^d, with a score function defined as:

$$\sigma(h, r, t) = R\left(h^\top W_r \bar{t}\right) \quad (6)$$

The use of real, purely imaginary or complex vectors can make the score function symmetric, antisymmetric or neither symmetric nor antisymmetric, so it can better reflect the symmetry of the relation.

RDF2Vec. Finally, RDF2Vec [16] applies the idea of Word2Vec [10] to knowledge graphs: random walks (that is, random sequences of connected entities and relations) are sampled in the graph to create a training set. Then, these walks are fed to a shallow neural network whose objective function is to accurately predict an entity or a relation given its context.

3.2 A Comparison of Knowledge Graph Embeddings using Separability

Since our goal is to use geometric similarity between embeddings to identify groups of entities that are semantically similar, we want to assess the ability of a model to embed entities from the same class to the same region of the Euclidean space. We formalize this by defining a class separability task, which aims at measuring if embeddings from different classes can be linearly separated.

The setup for this task is as follows: for two classes A and B, we sample N entities from A and N entities from B, which creates a dataset $D = \{\mathbf{e_i}, y_i\}_{i=1,\dots,2N}$ with $\mathbf{e_i} \in \mathbb{R}^d$ the embedding vector of entity e_i, and $y_i = 0$ if e_i is from class A, 1 otherwise. A linear SVM is trained over 75% of these samples, and then used to predict the labels of the 25% remaining samples. The predicted labels are then compared to the actual labels, using standard classification metrics: precision, recall, F-score. For each class, we sample 1,000 instances

if the class has enough instances, otherwise we use all the instances from that class.

Linear separability for entities of different classes is a strong requirement. An embedding model can perform well on standard evaluation benchmarks such as link prediction or triple completion without necessarily fulfilling this requirement. However, an embedding model that enforces as much as possible this linear separability constraint guarantees a straightforward relation between geometric similarity and semantic similarity, which makes the taxonomic extraction step much easier.

The challenge of separating class A from class B varies greatly with A and B: intuitively, it is harder to distinguish between a `College` and a `University` than between an `Aircraft` and a `Person`. To evaluate the complexity of the separability of the various classes of an ontology, we setup an experiment with three metrics: lexical distance measured using word embeddings, taxonomic distance based on an available taxonomy and class frequency in the knowledge graph. The first two metrics measure how difficult the task is based on class labels (lexical distance) and taxonomic distance. The last metric aims at measuring the effect of the size (in terms of instances) of a given class on the ability to produce embeddings that are useful for the separability task.

Finally, we assess the average separability scores (in terms of precision, recall and F-score) of the various knowledge graph embedding models presented in section 3.

Lexical Distance. We introduce a distance over classes, based on the semantic proximity of their labels. For a given class X, we split its label into words (e.g. `SportsTeamMember` becomes "sports", "team", "member"), and average the embeddings[1] of these words, which gives a vector representation $\vec{\mathbf{X}}$ of X. Then, we define the lexical distance between two classes A and B as the Euclidean distance between their representations:

$$d_{\text{lex}}(A, B) = \|\vec{\mathbf{A}} - \vec{\mathbf{B}}\|_2 \tag{7}$$

Taxonomic Distance. We also introduce a distance between classes based on their distance in the original taxonomy (in our experiments, we used DBpedia). Since the taxonomy is a tree, there is a unique (non-directed) path between any pair of classes $(A,\ B)$. We define the taxonomic distance between A and B as the length of this path, weighted by the depth of the edges in the path:

$$d_{\text{tax}}(A, B) = \sum_{e \in \text{path}(A,B)} \frac{1}{2^{\text{depth}(e)}} \tag{8}$$

Where $\text{depth}(e)$ is the depth of the edge in the tree: if an edge is connected to the root, it has depth 0, if it is connected to an immediate child of the root, it has depth 1, and so on. The weight of an edge decreases with its depth, because depth is an indicator of specificity in a taxonomy: the deeper a class is in the taxonomy, the more specific it is. Thus, two classes separated by only one edge are semantically closer if they are deep in the tree (e.g `MotorRace` and `Race`) rather than close to the root (e.g `Person` and `Agent`). Using the specific weighting scheme $1/2^{\text{depth}(e)}$ has the additional property that a class A is closer to its subclasses than to its siblings or to its superclasses.

[1]We used word embeddings from https://fasttext.cc/docs/en/english-vectors.html, trained on Common Crawl [9]

Table 1: Average precision, recall and F-score for different embedding models on the separability task.

Model	Precision	Recall	F-score
ComplEx	90.4	89.9	89.7
DistMult	92.5	91.6	91.6
RDF2Vec	**99.7**	**99.7**	**99.7**
TransE	99.4	99.1	99.2
TransH	93.6	92.1	92.5
TransD	85.0	83.1	83.5

Class Frequency. An entity involved in many triples will be seen many times during training, and thus may have a more reliable embedding vector. Since the embedding vector of an entity depends on the embeddings of the relations and entities that are connected to it in the knowledge graph, it is possible that entities from rare classes are involved in rare relations and linked to rare entities, so their own embeddings are less reliable than others. To assess this claim, we also investigate the effect of class size on separability scores. To provide a synthetic number for the sizes of a pair (A, B), we use the harmonic mean of the sizes of A and B. We find that using harmonic instead of arithmetic mean better handles the size imbalance that can occur between two classes: if N_A the size of A is orders of magnitude greater than the size of B, their arithmetic mean has the same order of magnitude as N_A, and thus does not reflect that B is rare comparatively to A.

We evaluate the six embedding models on 10,000 pairs of classes from DBpedia. We give the average separability score of each model in Table 1, and plot it for various lexical and taxonomic distances in Figure 1 and for various class sizes in Figure 2.

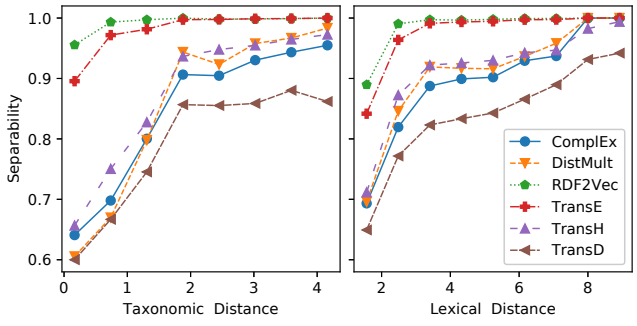

Figure 1: Average separability of two classes depending on the taxonomic distance (left) and lexical distance (right) between them, for different embedding models.

As expected, the separability score varies with the lexical and taxonomic distance between classes for all embedding models. All models but one handle correctly distant classes (lexical distance of 8 or above), but only two models have consistently high scores on most of the distance ranges: RDF2Vec and TransE, with RDF2Vec being slightly better than TransE especially for close classes. The average separability score of these two models drops below 95% when the lexical distance between classes is below 2, to reach 89% and 84% respectively.

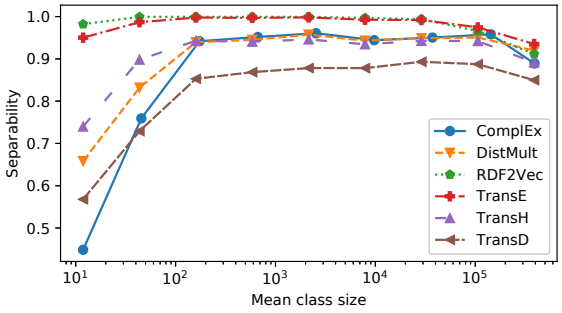

Figure 2: Average separability of two classes, depending on the harmonic mean of their number of instances, for different embedding models.

In Figure 2, we observe that rare classes are indeed less separable than more frequent classes, with some variability depending on embedding models. RDF2Vec is once again the best model, with TransE slightly behind. TransH, DistMult and ComplEx have close scores in the size range $[200, 10^5]$, but ComplEx is more sensitive to rare classes than DistMult, which is itself more sensitive than TransH. More surprisingly, we observe that scores also decrease for very frequent classes (mean size above 10^5 instances) for all embedding models. Both Pearson and Spearman's correlations between lexical distance and mean class size are close to zero, which rules out the possibility of a bias in the dataset. An explanation is that frequent classes are more likely to be parent of other classes in the dataset (for example, `dbo:Agent` is the second most frequent class of the dataset, and is a parent of half the classes in the DBpedia ontology), and separating a subclass from its parent is harder than separating sibling or unrelated classes. Finally, we observe a convergence of scores for frequent classes between all models, with TransE achieving the highest score.

In summary, two models achieve an average F-score above 95% on our separability task: RDF2Vec and TransE, with 99.7% and 99.2% respectively. In the remainder of this paper, we use TransE embeddings learned on the DBpedia knowledge base as the clusters that emerge using TransE appear more balanced in size.

4 EXPRESSIVE TAXONOMY LEARNING

Learning an expressive taxonomy is done by recursively repeating two steps: first, entity embeddings are clustered hierarchically, thus creating a structure over groups of semantically close entities. Then, clusters are labelled with axioms whose score is above a given threshold (see section 4.2). New entities are then sampled using these retained axioms, and the clustering and labelling steps are repeated with these new entities, until no relevant cluster is found.

4.1 Hierarchical Clustering

We start by sampling a set of n entities among those that verify a given axiom A_{start}. At first, $A_{\text{start}} = \top$, meaning that we sample among all entities in the knowledge graph. In later steps, and similarly to a bootstrapping approach, we sample entities that verify our newly extracted axioms.

These entities are first embedded into the Euclidean space $\mathbb{R}^d$. This $n \times d$ point cloud is then clustered using agglomerative clustering: at first, there are n clusters, each containing exactly one entity. Then, at each step, the two clusters that have the lowest average distance between their entities are merged together into a new cluster. The process is repeated until there is only one remaining cluster containing all entities. The result of this algorithm is a binary clustering tree, whose root is the whole dataset, and whose leaves are the entities. To avoid clusters that are too specific in the next step, we set a maximal depth D and remove clusters that are deeper than this. In our experiments, we used D in the range $[4-10]$. The purpose of this hierarchical clustering step is two-fold. First, it identifies groups of entities that are geometrically close, hence semantically coherent. This unsupervised, geometry-based class identification is a key point of our approach: the axiom mining step only occurs within groups of entities for which we already know that they are semantically similar. Second, it creates a hierarchy over these groups of entities, based only on the geometry of the embeddings. Once the axiom mining step is done, the clusters' hierarchy will naturally induce a hierarchy over axioms, that is, an expressive taxonomy.

4.2 Statistics based on Linked Data

Since relevant groups of entities have already been defined in the previous step, and since subsumption axioms are derived from the taxonomic structure of the clustering tree, the axiom mining step consists only in describing a cluster with one or several logical axioms. Here, we present a simple axiom mining method, but other approaches could be used. In this axiom mining scheme, we are interested in finding axioms that explain a clustering split.

Let C be a non-leaf cluster, L and R its left and right subclusters, that is $C = L \sqcup R$, with $\sqcup$ denothing the disjoint union. For an axiom A and an entity x, we write A(x) if x verifies axiom A. In order to explain the split between L and R, we want to extract an axiom A such that A is valid for all elements in L and none in R:

$$\forall x \in L, A(x) \wedge \forall x \in R, \neg A(x) \tag{9}$$

Or equivalently,

$$\forall x \in L \sqcup R, A(x) \oplus (x \in R) \tag{10}$$

With $\oplus$ denoting the exclusive or operation: an entity of $C = L \sqcup R$ cannot be in R and verify A at the same time, nor can it be in L without verifying A. We define the coverage of axiom A for the split $C = L \sqcup R$ as the proportion of elements in L that verify A, and the specificity of A as the proportion of elements in R that don't verify A. An optimal separating axiom has coverage and specificity equal to 1. We finally define a synthetic partition score for A as the arithmetic mean of coverage and specificity, weighted by the respective sizes of L and R. One can check that this partition score is indeed the proportion of elements $x \in C$ such that $A(x) \oplus x \in R$.

To find such an axiom, we first extract atomic axioms from a cluster, and then improve these atomic axioms by combining them with conjunctions, disjunctions and negations. In this paper, we only consider two types of atomic axioms: classes (concepts in description logic terminology), and existential restrictions. Three types of existential restrictions are considered: $\exists R.C$ with C a class, $\exists R.\{v\}$ with v an entity, and $\exists R.t$ with t representing litterals of

type t (e.g. `xsd:date`). For each entity x in our input cluster, we extract all the triples $(x,\ r,\ t)$ in the graph. If r is the `rdf:type` relation, then t is a class, and the triple $(x,\ r,\ t)$ is transformed into the atomic axiom t (concept axiom). Otherwise, the classes $C_1, \ldots, C_m$ of t are extracted, and the triple $(x,\ r,\ t)$ is transformed into atomic axioms $\exists R.\{t\}$ and $\exists R.C_1,\ \ldots,\ \exists R.C_m$. We end up with a list of atomic axioms for the entire cluster. As a last pruning step, we remove atoms that are verified by less than 10% of the entities.

Next, we generate candidate axioms using this set of atomic axioms. At first, candidate axioms are the atomic axioms themselves. Each candidate axiom is evaluated using the metrics of specificity and coverage. We experimentally define a score threshold δ, typically $\delta = 0.9$. If the coverage is below the threshold, the axiom does not cover enough entities. We thus start iteratively extending an axiom a by adding disjunctions (OR clauses): for each atomic axiom b that describes this cluster, we create a new candidate axiom $a \vee b$, which is in turn evaluated and extended if necessary. Conversely, if the specificity is below the threshold, the axiom does not separate well enough the two clusters, so we add conjunctions (AND clauses) as in the previous case. If both coverage and specificity are below the threshold, then the axiom is not a good candidate for explaining the split, and the search continues with other candidates. If both coverage and specificity are above the threshold, then the axiom is returned. At each step, only the N_{ax} candidate axioms with the highest scores are extended, to limit the size of the search space, where N_{ax} is a parameter. The search stops when the score does not improve enough from one step to the other, or when the number of steps is too high. This algorithm is described in Algorithm 1. For each cluster, the algorithm returns a list of axioms describing its left and right subclusters. Since at each step we consider equivalently atomic axioms and their negations, the algorithm is symmetric, so axioms in the right subcluster are the negation of axioms in the left subcluster.

ALGORITHM 1: Pseudo-code for iteratively improving a list of atomic axioms

```
Input: atoms, two clusters L and R
Output: list of axioms for the LR split
Parameters: max_steps, min_gain, score threshold δ, N_ax
found_axioms = ∅
to_improve = {None}
steps = 0
while steps <= max_steps do
    improved = ∅
    for axiom in to_improve do
        if axiom is None then
            OP = None
        else
            coverage, specificity, score = evaluate(a)
            if coverage < δ and specificity < δ then
                continue
            else if specificity < δ then
                OP = AND
            else if coverage < δ then
                OP = OR
        for atom in atoms do
            if OP is None then
                new_axiom = atom
            else
                new_axiom = atom OP axiom
            coverage, specificity, new_score =
              evaluate(new_axiom)
            if new_score > δ then
                found_axioms.push(new_axiom)
            else if new_score - score > min_gain then
                improved.push((new_axiom, new_score))
        if improved is empty then break
        to_improve := N_ax best axioms from improved
        steps := steps + 1
```

4.3 Taxonomy Learning

The final step is to extract the expressive taxonomy itself. For this, we inject the extracted axioms in the clustering tree. Clusters whose extracted axioms have low partition scores (below some threshold) are removed from the tree, meaning that they are not considered as a relevant level in the hierarchy, and their children are directly connected to their parents. If the extracted axioms do not cover all the n entities, it means that at least one subclass of A_{start} was not found, so we add a special class representing entities that verify A_{start} but none of the extracted axioms. For example, some cluster `Artist` might be divided into two labelled subclusters `Painter` and `Singer` and one unlabelled subcluster, leading to an extra class `Artist/...` of artists that are neither painters nor singers.

This results in a taxonomic tree $T(A_{\text{start}})$ over the entities that verify A_{start}. Since named classes were considered as atomic axioms, the resulting tree contains both named classes and expressive axioms. If $T(A_{\text{start}})$ is empty, there is no meaningful subclass of A_{start} and the search stops. Otherwise, for each new axiom B in $T(A_{\text{start}})$, we repeat the steps of sampling, clustering and axiom extraction: entities are sampled from B and clustered, axioms are extracted, and a taxonomy $T(B)$ is extracted. $T(B)$ is then inserted into $T(A_{\text{start}})$, and the search continues within $T(B)$. If $T(B)$ is empty, then there is no meaningful subclass of B, and the search stops. The algorithm can be further refined by decreasing the score threshold δ over time.

5 EVALUATION AND DISCUSSION

Assessing the quality of an expressive taxonomy is difficult because there is no gold standard available. In this section, we describe a quantitative evaluation of our approach on the "non-expressive" taxonomy extraction task, for which we have the DBpedia ontology as a gold standard, and we also provide a qualitative analysis of the expressive taxonomy that is obtained.

By restricting the extracted axioms to concepts (thus not considering existential relations), our algorithm can extract a taxonomy of named classes $\hat{T}$. As a gold standard, we use the DBpedia ontology[2] T^* and remove classes that have no instances, resulting in 455 classes. To compare $\hat{T}$ and T^*, we use standard evaluation metrics: precision (number of predicted axioms that are true), recall

[2]The latest version is available here: mappings.dbpedia.org/server/ontology/classes/

Table 2: Evaluation of the class subsumption axioms extracted, using DBpedia ontology as the gold standard. *Transitive* and *direct* indicates the evaluation with and without transitive closure respectively.

	Precision	Recall	F-score
Direct	68.94	68.79	68.87
Transitive	98.28	85.99	91.72

(number of true axioms that are predicted), and F-score (mean of precision and recall). Following [15], we also compute the transitive closure of $\hat{T}$ and T^*, and evaluate again the precision, recall and F-score. Results are shown in table 2. As we can see, the obtained F-score exceeds 90 percent, which indicates that DBpedia classes appear overall in the generated taxonomy but not necessarily at their same original depth in the DBpedia taxonomy.

We also provide a limited qualitative analysis of the expressive taxonomy[3]. Compared to the DBpedia taxonomy, which contains 455 classes, our taxonomy adds 106 expressive classes. We can notice that unpreviously defined classes emerge from the data. These emerging classes can subsume existing classes, such as the class `Work` $\wedge$ $\exists$`runtime.{xsd:double}` which contains all entities from `Film`$\vee$`TVShow`. We also obtain more specific subclasses such as `Language`$\wedge\exists$`spokenIn.Place` that becomes a subclass of `Language` and is a sibling of `ProgrammingLanguage`.

We can also note that, at the first level, we are able to retrieve all the previously known DBpedia classes (Place, Event, Agent,...). Some classes that emerge highlight the need to complete the data instances. For example, the class `Automobile` is associated with the properties `height`, `length` and `weight`, but at least one of these properties is missing for 50% of its instances. Frequent classes appear as more accurately described than rare classes, as expected.

Overall, we believe that our generated taxonomy can dynamically reflect the content of a knowledge graph and that this approach could be used to extract up-to-date schemas. Some limitations include the difficulty to automatically determine the threshold at which to stop the decomposition of a cluster into further clusters and whether a class that emerges is valuable or only reflective of data incompleteness.

6 CONCLUSION

In this paper, we presented a methodology to extract an expressive taxonomy that represents the current content of a knowledge graph. This work is part of approaches that aim at automatically completing or creating knowledge graphs. With the evolution of representation learning based on graphs and distributional semantics based on dense vectors, we have shown that it is indeed possible to extract the taxonomy of a knowledge graph as reflected by data instances. Our future work will tackle class definitions and enrich the set of axioms considered in the generation of the knowledge graph schema. We also aim at demonstrating how the obtained taxonomy can highlight data errors, inconsistencies or incompleteness, and contribute to the interpretability of the knowledge graph. Finally, we also plan on generating mixed representations based not only on knowledge graph embeddings but also on modern language models for a more accurate schema learning.

[3]Our expressive taxonomy can be found at labowest.ca/sdb2020

ACKNOWLEDGMENTS

This research was supported by Apogée Canada, Canada First Research Excellence Fund program.

REFERENCES

[1] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*. 2787–2795.
[2] Lorenz Bühmann, Jens Lehmann, and Patrick Westphal. 2016. DL-Learner—A framework for inductive learning on the Semantic Web. *Journal of Web Semantics* 39 (2016), 15–24.
[3] Lara Haidar-Ahmad, Ludovic Font, Amal Zouaq, and Michel Gagnon. 2016. Entity typing and linking using sparql patterns and DBpedia. In *Semantic Web Evaluation Challenge*. Springer, 61–75.
[4] Junheng Hao, Muhao Chen, Wenchao Yu, Yizhou Sun, and Wei Wang. 2019. Universal representation learning of knowledge bases by jointly embedding instances and ontological concepts. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1709–1719.
[5] Simin Jabbari and Kilian Stoffel. 2018. FCA-Based Ontology Learning From Unstructured Textual Data. In *International Conference on Mining Intelligence and Knowledge Exploration*. Springer, 1–10.
[6] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 687–696.
[7] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. 2020. A Survey on Knowledge Graphs: Representation, Acquisition and Applications. arXiv:cs.CL/2002.00388
[8] Na Li, Zied Bouraoui, and Steven Schockaert. 2019. Ontology completion using graph convolutional networks. In *International Semantic Web Conference*. Springer, 435–452.
[9] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in Pre-Training Distributed Word Representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.
[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*. 3111–3119.
[11] Heiko Paulheim. 2017. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web* 8, 3 (2017), 489–508.
[12] Heiko Paulheim and Christian Bizer. 2014. Improving the quality of linked data using statistical distributions. *International Journal on Semantic Web and Information Systems (IJSWIS)* 10, 2 (2014), 63–86.
[13] Rafael Peixoto, Thomas Hassan, Christophe Cruz, Aurélie Bertaux, and Nuno Silva. 2016. Hierarchical Multi-Label Classification Using Web Reasoning for Large Datasets. *Open Journal Of Semantic Web* (2016). https://doi.org/10.19210/1006.3.1.1
[14] Giulio Petrucci, Marco Rospocher, and Chiara Ghidini. 2018. Expressive ontology learning as neural machine translation. *Journal of Web Semantics* 52 (2018), 66–82. https://doi.org/10.1016/j.websem.2018.10.002
[15] Petar Ristoski, Stefano Faralli, Simone Paolo Ponzetto, and Heiko Paulheim. 2017. Large-scale taxonomy induction using entity and word embeddings. In *Proceedings of the International Conference on Web Intelligence*. ACM, 81–87. https://doi.org/10.1145/3106426.3106465
[16] Petar Ristoski and Heiko Paulheim. 2016. Rdf2vec: RDF graph embeddings for data mining. In *International Semantic Web Conference*. Springer, 498–514.
[17] Andrea Rossi, Donatella Firmani, Antonio Matinata, Paolo Merialdo, and Denilson Barbosa. 2020. Knowledge Graph Embedding for Link Prediction: A Comparative Analysis. arXiv:cs.LG/2002.00819
[18] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *Proc. International Conference on Machine Learning*.
[19] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *Twenty-Eighth AAAI conference on artificial intelligence*.
[20] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *Proc. 3rd International Conference on Learning Representations*.
[21] Amal Zouaq, Dragan Gasevic, and Marek Hatala. 2011. Towards open ontology learning and filtering. *Information Systems* 36, 7 (2011), 1064–1081.