SEED: A System for Entity Exploration and Debugging in Large-Scale Knowledge Graphs

Jun Chen, Yueguo Chen, Xiaoyong Du, Xiangling Zhang, Xuan Zhou
School of Information, Renmin University of China, Beijing, China
Key Laboratory of Data Engineering and Knowledge Engineering (Renmin University of China), MOE, China
chenyueguo@ruc.edu.cn

***Abstract*—Large-scale knowledge graphs (KGs) contain massive entities and abundant relations among the entities. Data exploration over KGs allows users to browse the attributes of entities as well as the relations among entities. It therefore provides a good way of learning the structure and coverage of KGs. In this paper, we introduce a system called SEED that is designed to support entity-oriented exploration in large-scale KGs, based on retrieving similar entities of some seed entities as well as their semantic relations that show how entities are similar to each other. A by-product of entity exploration in SEED is to facilitate discovering the deficiency of KGs, so that the detected bugs can be easily fixed by users as they explore the KGs.**

I. Introduction

As huge knowledge graphs (KGs) contain massive entities and abundant relations among the entities, they can be widely applied to applications such as semantic search, question answering and recommender systems [6]. Currently, public KGs such as DBpedia [3] and YAGO [4] contain millions of entities and hundreds of millions of facts/triples. They continuously grow as more facts are automatically discovered from the underlying sources or manually created by human.

The fast evolving of KGs poses some challenges to the effective usage of them. One challenge is how to effectively search for interesting content from KGs. SPARQL [7] is not user-friendly because it requires users to have a good knowledge of the underlying metadata (predicates and prefixes). Keyword search [9] is sometimes unable to accurately capture users' query intention. Another challenge is how to effectively detect and fix the deficiency of KGs.

We propose a system called SEED, which is able to partially address the above challenges. Similar to the notion of exemplar queries [5], SEED allows users to explore the KGs using a small number of seed entities (seeds), from which it automatically discovers the semantic relations (called semantic patterns) shared among the seeds, and accordingly obtains similar entities of the seeds. By examining the mappings between entities and semantic patterns (SPs), users can easily debug and fix some deficiency of KGs as they explore the KGs. The two tasks are seamlessly integrated by an explore-and-feedback process in SEED.

The procedure of entity exploration in SEED is actually an entity set expansion process, which aims at finding entities that are similar to a given small number of seeds. For example, given the seeds $Forrest\_Gump$, $Apollo\_13$, $The\_Polar\_Express$, we may expect to find entities such as $Cast\_Away$, because they are all movies starring Tom_Hanks. SEED can automatically discover the SPs shared among the seeds, such as $Tom\_Hanks : \underline{starring}$ and $Movie : \underline{type}$ (the left part is an anchor entity and the right part is an edge, further explained in Section 3.1). Entities satisfying the discovered SPs are treated as candidates, which are further ranked using an effective ranking model. SEED allows users to interactively modify the query (seeds) or select some specified SPs they prefer, so that entities can be explored with different semantic constraints.

During the process of entity exploration, a list of similar entities as well as their relevant SPs are discovered and maintained. SEED further helps users to detect the deficiency of the underlying KGs, by evaluating the possible mismatches between the entities and the SPs. For example, SEED discovers that $Cast\_Away$ is a relevant entity of the above SP. Meanwhile, it finds that this entity may lack the SP $United\_States : \underline{country\_of\_origin}$, which is widely shared by many other similar entities. In this case, SEED may treat it as a mismatch between the entity and the SP, according to a confidence score derived from an association rule mining algorithm. This certainly helps users to detect and correct the bugs as they explore the KGs.

II. System overview

SEED is designed using an architecture of three layers: the user interface layer, the query processing layer, and the data storage layer. They are shown in Figure 1.

A. User Interface

Entity exploration and debugging are supported by the web-based user interfaces called **Exploration Visual** and **Debugging Visual** respectively. To assist users in formalizing a query rapidly, a dropdown input box is provided on the middle of the UI. Users can search an entity by typing some keywords, based on which a list of relevant entities are recommended for picking. The input box shows the selected seeds, which can also be removed by users. Query results (entities and SPs) are returned and listed once a query is submitted. Users can also select seeds from the query results, or filter out irrelevant SPs to generate a new query. We apply ECharts [1] (a tool for graph visualization) to assist the exploration of entity attributes.

[1] http://echarts.baidu.com/index-en.html

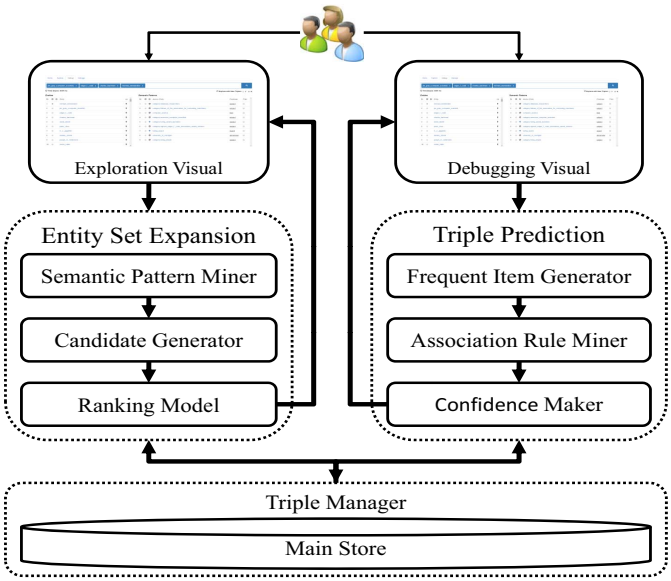

Fig. 1. System architecture.

In addition to the **Exploration Visual**, the results of the entity exploration are also shown on the **Debugging Visual**. Users are able to debug knowledge by creating new triples using simple click operations, with the recommendations generated from the **Triple Prediction** component.

B. Query Processing

Once the seeds are updated on the **Exploration Visual**, an entity set expansion process will be triggered, which first discovers SPs via a component called **SP Miner**. Then, the **Candidate Generator** finds candidate entities satisfying the discovered SPs. Those candidates will be further ranked by the **Ranking Model** component to evaluate their relevance to the seeds. The details will be discussed in Section III-A.

One missing triple can be generated from a mapping between an entity and a SP. The **Triple Prediction** component is therefore designed to evaluate the likelihood of the missing triples, which are quantified by the **Confidence Marker** based on the mined associate rules generated by the **Frequent Items Generator** and the **Association Rule Miner**. Missing triples of high confidence will be recommended in the **Debugging Visual** to assist users with knowledge debugging. The details of triple prediction will be discussed in Section III-B.

C. Data Storage

We apply a general database (MySQL) as the backend store for maintaining triples. Considering that the query performance is critical for interactive operations, to guarantee the performance of data exploration and knowledge debugging, special indexes are created for triples. In SEED, a SP consists of a predicate and an entity. We therefore create B^+-tree indexes on two combinations: S-P (subjects and predicates) and O-P (objects and predicates), which support fast retrieval of not only semantic patterns, but also subjects or objects respectively. A merit of using relational databases as the backend store is that it assists the users in manipulating triples (insert, delete and update, which are necessary for knowledge debugging) very efficiently.

III. Key Techniques

In this section, we present two key techniques used by SEED: (1) entity set expansion, for discovering and ranking candidate entities and their relevant SPs, (2) triple prediction, for assigning a confidence score to a potential triple that is discovered from a mismatch between an entity and a SP.

A. Entity Set Expansion

A triple in KGs can be represented as $< s, p, o >$. A subject or an object (not including literal objects) is treated as an entity in our study. For such a triple, the predicate actually defines the relation between two entities. In our system, we use a concept of semantic pattern to define the common features shared among entities. There are two types of SPs: $< e, p, x >$ and $< x, p, e >$, where x is an entity variable, e is called an anchor entity. The former SP $< e, p, x >$ (shorted as $e : p$) represents a triple pattern having e as the subject and p as the predicate; the latter one $< x, p, e >$ (shorted as $e : \underline{p}$) represents a triple pattern having e as the object and p as the predicate. We apply an underline to the predicate p to distinguish $< x, p, e >$ from $< e, p, x >$. In this study, we use κ to denote the RDF KGs, π to denote a SP, $e' \models \pi$ to denote that the entity e' satisfies the SP π, and $E(\pi, \kappa)$ to denote the set of entities in κ satisfying π. For instance, a SP $\pi = Tom\_Hanks : \underline{starring}$ represents the triple pattern of the entities that have *Tom Hanks* as a star, i.e., $< x, starring, Tom\_Hanks >$, and $Forrest\_Gump \models \pi$ simply because there is a triple $< Forrest\_Gump, starring, Tom\_Hanks >$ in κ.

In SEED, entity exploration is achieved through a process called entity set expansion which aims at finding a list of entities similar to a set of $m \geq 1$ seed entities $Q = \{e_1, e_2, \ldots, e_m\}$ from the KGs. We solve this problem by discovering the SPs satisfied by all (or part of) the seed entities. Then, the candidate entities satisfying those SPs will be ranked by their similarity score defined as:

$$S(e) = \sum_{\pi \in \Phi(Q) \ \wedge \ e \models \pi} R(\pi, Q) * D(\pi, \kappa) \tag{1}$$

where $\Phi(Q)$ is the set of relevant SPs of the seeds in Q. $R(\pi, Q)$ is the relevance score of π to the given query Q and $D(\pi, \kappa)$ is the discriminative score of π in KGs. Two components are further evaluated as follows.

Discriminability $D(\pi, \kappa)$: The number of entities satisfying π in κ is applied for evaluating the discriminability $D(\pi, \kappa)$. According to the Equation 2, the larger $|E(\pi)|$ the less discriminability that π has. For instance, for two SPs $\pi_1 = Tom\_Hanks : \underline{starring}$ and $\pi_2 = Movie : \underline{type}$, it is obvious that $D(\pi_1, \kappa) > D(\pi_2, \kappa)$ because π_1 is more discriminative than π_2.

$$D(\pi, \kappa) = \frac{1}{log|E(\pi)|} \tag{2}$$

Relevance $R(\pi, Q)$: Considering the deficiency of KGs, we allow the cases that some seeds in Q do not satisfy π.

Intuitively, the more seeds in Q satisfying π, the more relevant it is. There are two reasons that a seed does not satisfy π: 1) π is a true positive SP to Q. It is however due to the deficiency of the KGs; 2) π is a false positive SP. As a result, only part of the seeds satisfy it. We need to discriminate these two cases so that the first case can obtain more relevance score than the second case. The relevance $R(\pi, Q)$ is formalized as:

$$R(\pi, Q) = \prod_{e \in Q)} p(e, \pi) \tag{3}$$

where $p(e, \pi)$ is the probability of e satisfying π. Based on the idea of $collaborative\ filtering$ in recommendation systems [8], we evaluate $p(e, \pi)$ by considering the likelihood of e satisfying similar SPs of π:

$$p(e, \pi) = \begin{cases} 1 & if\ e \models \pi \\ \frac{\sum_{\pi' \in \psi(\pi)} I(e, \pi') w(\pi, \pi')}{\sum_{\pi' \in \psi(\pi)} w(\pi, \pi')} & otherwise \end{cases} \tag{4}$$

where $\psi(\pi)$ is derived by substituting the anchor entity (from e_a to any other e_x) or the predicate (from p to any other p_x) of $\pi = e_a : p$ respectively; $I(e, \pi') = 1$ if $e \models \pi'$, and zero otherwise; the weight of π', $w(\pi, \pi') = \frac{|E(\pi) \cap E(\pi')|}{|E(\pi)|}$.

B. Triple Prediction

SEED explicitly provides users the relevant SPs to show how entities are relevant to each other. When using the **Debugging Visual** interface, users often find some mismatches between some relevant entities and relevant SPs, which may be due to the deficiency of KGs. To deal with it, we give recommendations for users to modify these mismatches, by predicting the likelihood of an entity satisfying a SP. This is achieved by applying an association rule mining strategy [2], which is very efficient for the task of triple prediction.

Given a set of items $I = \{i_1, i_2, \ldots, i_m\}$, an association rule is an implication $X \rightarrow Y$ consisting of the itemsets $X, Y \subset I$ with $X \cap Y = \emptyset$. Given a set of samples (transactions) $T = \{t | t \subseteq I\}$. The minimum support of a rule $X \rightarrow Y$ denotes $s\%$ of the samples in T that contain $X \cup Y$. The minimum confidence of a rule $X \rightarrow Y$ denotes $c\%$ of the samples in T that contain X also contain Y.

Inspired by the idea of [1], to predict whether an entity e' satisfies a SP $\pi = e : p$ (note that if $e' \models \pi$, we will not predict it), we predict whether e' implies e and p respectively. If e' implies both e and p, it is very likely that e' is able to imply π, i.e., the triple $< e, p, e' >$ (or $< e', p, e >$ if $\pi = e : p$) is likely to be true. Otherwise, if neither e nor p can be implied by e', most likely, the triple does not exist.

We use the predicate p as an example to show how the association rule mining algorithm is applied to predict whether an entity e' implies p or not. A number of top relevant entities of entity set expansion are used as samples (transactions) of T. Let $P(e)$ be the set of predicates which have a subject e. The predicates in $P(e')$ are treated as items of a transaction e'. The predicate p in π is treated as Y. Firstly, we find the frequent itemsets from the set T. Secondly, for each frequent itemset X (a set of predicates), we check whether $X \rightarrow Y$ is satisfied based on the confidence of the association rule. Lastly, if there is an associate rule $X \rightarrow Y$ such that $X \subseteq P(e')$, then we conclude that e' implies p.

The prediction of whether an entity e' implies e is very similar to that of whether e' implies p. The neighbours of an entity e will be used as items of a transaction e', and e in π will be treated as Y. Finally, the prediction is made based on whether an entity e' implies e and p:

- High confidence: implies both e and p.
- Medium$^+$ confidence: only implies e.
- Medium confidence: only implies p.
- Low confidence: Neither e nor p is implied.

Note that the implication of e is more important than that of p, simply because e is more discriminative than p.

IV. Demonstration

The DBpedia v3.9 dataset is applied for demonstration, which includes the core subsets of DBpedia: Mapping-based Properties, Mapping-based Types, Articles Categories, Page Links and Redirects. The applied KGs finally contain nearly $40M$ triples and $6.5M$ entities. A component of coreference resolution is designed to remove/add the URIs of entities and predicates for the dataset. The demonstration will be focused on two key functions of SEED: data exploration and knowledge debugging.

A. Entity Exploration

The first function we provide to the audience is the entity exploration of the KGs. The audience can come up with a seed entity of interest (by typing some keywords or using the provided cases), SEED then automatically discovers the relevant SPs, based on which similar entities are discovered and ranked. If the audience want to learn more about the KGs, they can modify the inputs by selecting some new seeds from the answer list or remove some seeds to form a new query. Meanwhile, they can focus on some specified SPs (by clicking them), which will filter out the irrelevant entities that do not satisfy them. When the audience click a specific entity, a directed graph centered on the entity will be presented, showing the ingoing and outgoing edges of the entity.

B. Knowledge Debugging

During the entity exploration process, the audience may detect some mismatches between entities and SPs, due to the deficiency of the KGs. To deal with them, SEED provides the knowledge debugging function to fix the detected bugs with humans' intervention. With the help of user interface, the audience can freely switch to the **Debugging Visual** at any time of entity exploration. When the audience select an entity (or a SP), the SP list (or the entity list) will automatically generate the feedback (visualized by different colors) as recommendations to the triples formed by the entity (or SP) and the corresponding SPs (or entities). The audience can fix the bug by clicking a recommended plus sign, which leads to the creation of a new triple.

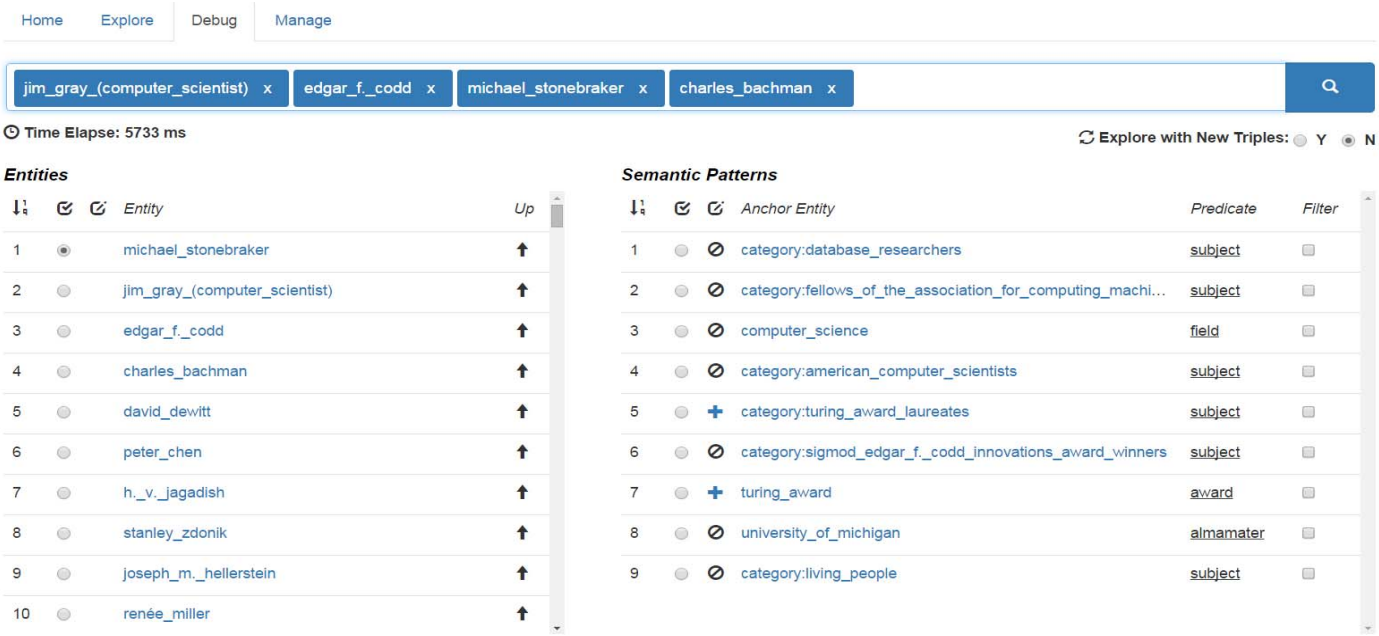

Fig. 2. The user interface of a use case

C. Use Cases

We designed some use cases for the audience to facilitate understanding the system. The audience can freely choose a seed from the provided topics, and initiate an entity exploration process. For example, given a famous database researcher Jim_Gray as input, we can quickly see the relevant SPs on the right side, including $Turing\_Award$: $award$, $Database\_Researchers$: $subject$, etc. The left side shows a list of similar entities. Through hovering the mouse, we can easily detect the relations between entities and SPs, where the focused entity satisfies those star-marked SPs, and vise versa. If we are more interested in database researchers with the title of ACM_Fellow, we can either provide more relevant seeds or focus on some specified SPs. For example, we further explore the KGs by selecting $Jeffrey\_Ullman$ as another seed via clicking the up-arrow sign of the result $Jeffrey\_Ullman$. However, we find that some of the top entities do not satisfy our need. We may directly focus on the SP of ACM_Fellow to filter results. Casually, we find $Edgar\_Codd$, Jim_Gray and $Michael\_Stonebraker$ are all $Turing\_Award$ winners in the database area. In order to discover more common relations among them, we can update seeds by choosing the above entities. After browsing, we detect that $Michael\_Stonebraker$ misses some SPs due to the deficiency of KGs.

As illustrated in Figure 2, when we click on the left radio button of $Michael\_Stonebraker$ (No. 1 entity), the right side will highlight those SPs that mismatch (with a plus sign) with it, implying that the KGs do not contain the corresponding triples connecting the SPs and the entity. The color of the plus signs indicates the confidence of the mismatch generated by SEED, and blue ones mean recommendations of highest confidence. We can fix the mismatch by clicking the plus sign, the system will automatically create a triple for it and insert it into the underlying KG as delta knowledge. The new triples will take effect immediately in the next round of exploration if we choose to explore with new triples.

Acknowledgment

This work is supported by National Basic Research Program of China (973 Program) No. 2012CB316205, the National Science Foundation of China under grant (No. 61472426, 61170010, 61272138), the Fundamental Research Funds for the Central Universities, the Research Funds of Renmin University of China No. 14XNLQ06, and a gift by Tencent.

References

[1] Z. Abedjan and F. Naumann. Improving RDF data through association rule mining. *Datenbank-Spektrum*, 13(2):111–120, 2013.

[2] S. R. Agrawal Rakesh. Fast algorithms for mining association rules in large databases. In *VLDB*, pages 487–499, 1994.

[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC*, pages 722–735, 2007.

[4] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. YAGO2: A spatially and temporally enhanced knowledge base from wikipedia. *Artif. Intell.*, 194:28–61, 2013.

[5] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas. Exemplar queries: Give me an example of what you need. *PVLDB*, 7(5):365–376, 2014.

[6] A. Passant. dbrec - music recommendations using dbpedia. In *ISWC*, pages 209–224, 2010.

[7] E. Prud-hommeaux and A. Seaborne. Sparql query language for rdf. In *W3C. https://www.w3.org/TR/rdf-sparql-query/*, 2008.

[8] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. Artificial Intellegence*, 2009:421425:1–421425:19, 2009.

[9] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In *ICDE*, pages 405–416, 2009.