Exploratory Querying of Extended Knowledge Graphs

Mohamed Yahya[†] Klaus Berberich[†] Maya Ramanath[‡] Gerhard Weikum[†]
[†]Max Planck Institute for Informatics, Germany [‡]IIT Delhi, India
{myahya,kberberi,weikum}@mpi-inf.mpg.de ramanath@cse.iitd.ac.in

ABSTRACT

Knowledge graphs (KGs) are important assets for search, analytics, and recommendations. However, querying a KG to explore entities and discover facts is difficult and tedious, even for users with skills in SPARQL. First, users are not familiar with the structure and labels of entities, classes and relations. Second, KGs are bound to be incomplete, as they capture only major facts about entities and their relationships and miss out on many of the more subtle aspects.

We demonstrate TriniT, a system that facilitates exploratory querying of large KGs, by addressing these issues of "vocabulary" mismatch and KG incompleteness. TriniT supports query relaxation rules that are invoked to allow for relevant answers which are not found otherwise. The incompleteness issue is addressed by extending a KG with additional text-style token triples obtained by running Open IE on Web and text sources. The query language, relaxation methods, and answer ranking are extended appropriately. The demo shows automatic query relaxation and has support for interactively adding user-customized relaxations. In both situations, the demo provides answer explanations and offers additional query suggestions.

1. MOTIVATION & INTRODUCTION

Digital knowledge about the world's entities and relationships has been compiled at a large scale in projects like DBpedia, Freebase, and Yago, and in commercial *knowledge graphs (KGs)* at companies like Google, Microsoft, Bloomberg, and many others. Most KGs are based on the RDF data model, where facts are expressed as *subject-predicate-object (SPO) triples* as shown in Figure 1. In RDF jargon, subjects and predicates are canonical *resources*, while objects can be either resources (e.g. `AlbertEinstein`) or literal values like strings, numbers, and dates (e.g. *'1879-03-14'*). Looking at such data as an entity-relationship graph, subjects and objects can be seen as entity nodes or value nodes, connected by relations as directed edges. The power of this model comes from the flexibility with which new data can be added without the need for a schema to be defined upfront.

Subject	Predicate	Object
AlbertEinstein	bornIn	Ulm
Ulm	locatedIn	Germany
AlbertEinstein	bornOn	*'1879-03-14'*
AlfredKleiner	hasStudent	AlbertEinstein
AlbertEinstein	affiliation	IAS
PrincetonUniversity	member	IvyLeague

Figure 1: Sample knowledge graph

Querying the data in a KG is based on structured SPO patterns, using query languages in the style of SPARQL. A query is a set of conjunctively combined *triple patterns*, where one or more components of an SPO triple are replaced with a variable. Occurrences of the same variable in multiple triple patterns indicate a join, requiring that the variable bind to the same graph node (or edge if the variable is in the P position). The flexibility with which data can be added to a KG comes at a price: users who are not familiar with the detailed structure and node and edge labels of the KG often have a hard time formulating proper queries.

Figure 2 shows the attempts of four users to formulate triple pattern queries for information needs they would like to answer from the KG shown in Figure 1. Before considering the KG data, the first three attempts seem reasonable, whereas only the last user fails to come up with a query.

Users A, B, and C will be disappointed by the answers returned by their queries. User A does not know that in the KG, people are born in cities, not countries. User B does not know that the predicate she actually needs is `hasStudent`, with `AlbertEinstein` as the object rather than the subject.

User C's case is more subtle. Strictly speaking, Einstein was not officially affiliated with any Ivy League university. He gave lectures at Princeton, and the Institute for Advanced Study (IAS), where he officially worked, was housed

A. *"Who was born in Germany?"*
```
?x bornIn Germany
```
B. *"Who was the advisor of Albert Einstein?"*
```
AlbertEinstein hasAdvisor ?x
```
C. *"Ivy League university Einstein was affiliated with."*
```
AlbertEinstein affiliation ?x ; ?x member IvyLeague
```
D. *"What did Albert Einstein win a Nobel prize for?"*
———

Figure 2: Questions and queries

Proceedings of the VLDB Endowment, Vol. 9, No. 13
Copyright 2016 VLDB Endowment 2150-8097/16/09.

in Princeton. The vocabulary of the KG, though, is not rich enough to support all these subtleties, resulting in an empty answer. A more useful answer would be `PrincetonUniversity` along with an explanation like the one above.

User D will be left dissatisfied, too, as she fails to formulate a query using the KG vocabulary. In her case, the KG lacks the necessary predicate.

The examples point out two major problems that users have in exploratory querying of KGs. First, the user is not familiar with the structure and vocabulary of the KG (users A, B, C). Second, the KG itself is incomplete, lacking specific knowledge required to satisfy a user's query or not supporting the query at all (users C, D). In some cases, multiple rounds of query reformulation may eventually lead to the desired answers, but users will likely view this as tedious.

To address these usability pain points, we have developed *TriniT*: a system that allows users to extend the KG to make up for missing knowledge, massages a user's query to better align it with the extended data, and can make suggestions regarding future queries [14]. TriniT supports query relaxation rules that are invoked to allow for relevant answers which are not found otherwise. To overcome the incompleteness problem, the KG is extended with additional text-style facts gathered by Open Information Extraction (Open IE) on additional sources like Web pages, news articles, etc.

2. FRAMEWORK

Extended Knowledge Graph (XKG) No KG will ever be complete. Many of the finer but interesting aspects of entities and relationships and all newly emerging knowledge will be missed because they are expressed only in hard-to-extract form in Web or enterprise contents like news, homepages, review forums, product descriptions, customer requests, hand-crafted HTML tables and spreadsheets, and other online media. However, we can run Open Information Extraction tools (e.g. ReVerb/OLLIE [2]) on these sources and collect textual triples consisting of two noun phrases (for S and O) and a verbal phrase connecting them (for P). In some case, tools for Named Entity Disambiguation (e.g. AIDA, Spotlight, or TagMe) can link the S or O phrases to entities in the KG. For example, the sentence *"Einstein won a Nobel for his discovery of the photoelectric effect"* yields a triple with `AlbertEinstein` as S (an entity, i.e., resource in the RDF sense), the string *'won a Nobel for'* as P, and the string *'discovery of the photoelectric effect'* as O.

We extend the traditional KG setting by these kinds of automatically extracted triples which can have textual *tokens* in any of the S, P, O slots. Note that these triples come with substantially lower confidence than the facts of the original KG, and are more vague to interpret. However, they are a valuable asset for filling gaps in the incomplete KG, and can also help users to formulate queries more easily. Figure 3 shows examples of additional triples in the XKG.

Subject	Predicate	Object
`AlbertEinstein`	*'won Nobel for'*	*'discovery of the photoelectric effect'*
`IAS`	*'housed in'*	`PrincetonUniversity`
`AlbertEinstein`	*'lectured at'*	`PrincetonUniversity`
`AlbertEinstein`	*'met his teacher'*	*'Prof. Kleiner'*

Figure 3: Sample knowledge graph extension

Extended Triple Patterns We also extend the query language to support triple patterns with textual tokens. For example, a user can issue the query `AlbertEinstein` *'won nobel for'* `?x`. Some queries can be directly evaluated on the XKG, while others require the ability to translate between the user's formulation and the XKG vocabulary, which we achieve by query relaxation as explained next.

3. QUERY RELAXATION

A relaxation rule replaces a set of triple patterns in the original query with a set of new patterns. Each rule has a weight $w \in [0, 1]$ that reflects the semantic similarity between the original set of triple patterns and their replacement, and is used to score answers and guide top-k query processing. The decision about which rules to invoke is adaptively made at run-time during top-k query processing.

Figure 4 shows examples of relaxation rules. Rules 1 and 2 replace KG predicates by alternative KG predicates with appropriate handling of argument order. Rules 3 and 4 allow query processing to move between the KG and the XKG.

#	Original	Replacement	Weight
1	`?x bornIn ?y ;` `?y type country`	`?x bornIn ?z ;` `?z type city ;` `?z locatedIn ?y`	1.0
2	`?x hasAdvisor ?y`	`?y hasStudent ?x`	1.0
3	`?x affiliation ?y`	`?x affiliation ?z ;` `?z` *'housed in'* `?y`	0.8
4	`?x affiliation ?y`	`?x` *'lectured at'* `?y`	0.7

Figure 4: Examples of relaxation rules

We provide concrete relaxation rules for rewriting predicates, which are mined from the XKG itself. We generate a rule rewriting the XKG predicate p_1 to the XKG predicate p_2 and assign it the weight $w(p_1 \mapsto p_2) = \frac{|args(p_1) \cap args(p_2)|}{|args(p_2)|}$, where $args(p)$ is the set of subject-object pairs connected by p in the XKG [14]. Additionally, relaxation rules can be specified manually, or automatically obtained using rule mining on the KG (e.g. [5]), paraphrase repositories (e.g. [10, 6]), and statistical/semantic relatedness measures (e.g. [4]). TriniT has an API for *relaxation operators*, which administrators and advanced users can use to plug in their code for generating relaxation rules and their weights.

4. ANSWER SCORING

With query relaxation, a query can return a potentially large number of answers. These answers differ in how well they capture the user's original intent. Computing relevance scores and providing the user with a list of ranked results is crucial for knowledge exploration and general usability.

We use a query-likelihood approach for scoring answers, which is standard in IR [18]. We adapt and extend this approach for our triple-pattern setting [14]. A triple pattern is viewed as a document that emits triples with certain probabilities. The probability assigned to an SPO fact in response to a triple pattern is proportional to the frequency with which the fact is observed (a tf-like effect) and inversely proportional to the total number of matches for the triple pattern (an idf-like effect corresponding to selectivity). Additionally, the scoring model takes into consideration the weights associated with a relaxation rule, so that answers obtained through a relaxation rule have their scores attenuated by the weight of the rule. Finally, as the same answer can be obtained through multiple sequences of relaxations, we define the score of an answer to be the maximal one obtained through any such sequence.

We have performed extensive experiments to evaluate our framework [14]. On a challenging set of 70 entity-relationship queries, we achieve an average NDCG at rank 5 of 0.775, with the next best state-of-the-art system achieving 0.419.

It is crucial to avoid exploring the entire space of possible rewritings, as this can be prohibitively expensive. TriniT uses a top-k approach to query processing that is an extension of the incremental top-k algorithm of [11], guided by scoring scheme outlined above. Top-k query processing is based on the ability to access answers for a triple pattern in sorted order of their scores, allowing us to go only as far as necessary into each triple pattern index list. Additionally, query processing utilizes incremental merging of triple patterns and their relaxed forms, invoking a relaxation only when it can contribute to the top-k answers.

5. DEMONSTRATION

Our demonstration shows TriniT from an end-user perspective and provides a look at its internals. The TriniT interface allows users to pose queries on the XKG, with a mix of traditional-SPARQL triple patterns and text-style token triple patterns. Answers can be explored in the browser, with links to the XKG. Users can define their own relaxation rules. For users interested in the details of query processing, TriniT can show internal steps and provides explanations of answers. Finally, TriniT can suggest query re-formulations of user queries that are better aligned with the KG.

Setting: We demonstrate TriniT using *entity-relationship search* as an application. The goal of entity search is to answer queries about people, products, organizations, etc. – and their relationships. In contrast to the entity-search capabilities that Google and Bing transparently provide for Web search, TriniT supports a much more expressive class of queries. In particular, we can handle queries that connect multiple entities by their relationships, potentially returning lists of entity pairs or entire tuples. Moreover, some queries that seek a single entity require joining triples from multiple sources, because no single Web page (or other data source) has the contents to match all query conditions. TriniT is specifically geared for these *join-intensive* queries, which are out of scope for most other entity-search systems. Such queries typically arise in the advanced information needs of journalists, market analysts, and other knowledge workers.

We use Yago2s as underlying KG, which combines factual knowledge extracted from Wikipedia infoboxes with ontological knowledge that comes from the combination of Wikipedia categories and WordNet classes. To form the XKG, we extract facts from Google's FACC1 annotations of the ClueWeb'09 Web crawl with Wikipedia entities, using Open IE techniques. Our XKG consists of a total of 440 million distinct triples: about 50 million from Yago2s, our KG, and 390 million from the extractions from ClueWeb.

TriniT is implemented in Java, with ElasticSearch serving as the storage backend for triple pattern answers and other meta-data required for the demonstration.

Query Interface: TriniT's query interface allows users to pose queries with multiple triple patterns, specify the number of results to be returned, and supply TriniT with relaxation rules invoked during query processing. An example is shown in the screenshot in Figure 5.

User input is eased by auto-completion, guiding users towards meaningful query formulations. Each of the SPO fields in a triple pattern accepts either a canonical KG resource or a textual token. The two rules shown in the screenshot correspond to rules 3 and 4 of Figure 4. Answers are returned in the form of variable bindings for projection variables. Advanced users can explore additional options.

Answer Explanation: Since the user's original query can be modified by relaxation rules to obtain more relevant answers, it is important for the user to understand how a specific answer was obtained. TriniT's answer explanation interface shows an answer's provenance.

The answer explanation provides three important pieces of information: (i) the KG triples that contributed to an answer, (ii) the XKG triples that contributed to an answer and their provenance, and (iii) the relaxation rules that were invoked to obtain an answer. In addition to showing the changes made to the user's query to obtain answers, answer explanation helps users better understand the schema of the underlying KG and its shortcomings. In the long term, this allows her to formulate queries better aligned with the KG.

Query Suggestion: Finally, TriniT suggests alternative formulations of the user's original query that are more suitable for the KG. This helps the user to learn more about the structure and node/edge labels of the underlying KG, making future queries easier to formulate.

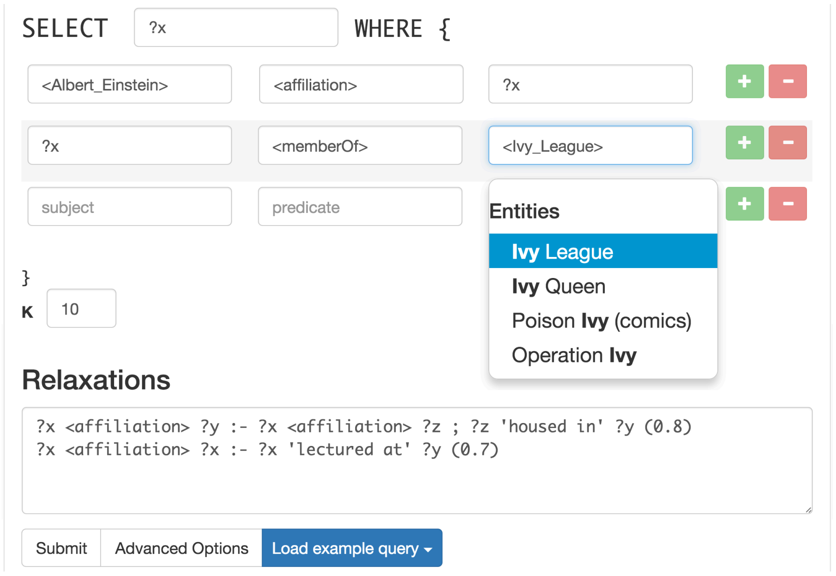
The extended query language, as explained earlier, allows textual tokens to be used as S, P, or O in a triple pattern. When TriniT determines that matches for these tokens have a significant overlap with matches for highly related KG resources (i.e., canonicalized entities or relations), these resources are suggested to the user for use in future queries. When a structural relaxation rule (e.g. a predicate inversion rule) is invoked and contributes to the final answer set, TriniT informs the user of this effect. This way, the user gradually gains a better understanding of the KG.

6. RELATED WORK

Exploratory querying of databases in general has recently received much attention [7]. Keyword queries over relational graphs have been investigated in a variety of projects including BANKS, DBExplorer, NAGA, and others; [17] gives an overview on this line of research. Related work also includes entity search over document collections; [1] gives an overview from an IR perspective. None of these approaches address the pain points of formulating queries and of coping with incomplete data or knowledge bases.

Natural language question answering (QA) has seen a major revival with the IBM Watson system. When the underlying data is structured, like a database or KG, the goal becomes translating a user question into a structured query [8]. The QALD benchmarking initiative [12], now in its fifth year, puts emphasis on linked open data in RDF format. Our work in this demo paper is orthogonal to the question translation, but TriniT would be a suitable platform for the queries into which user questions are mapped. In fact, we plan to use it as back-end for our own work on QA [13].

Recently, specific attention has been paid to graph data. Yang et al. [15, 16] developed SLQ, a framework for schemaless and structure-less graph querying. Li et al. [9] developed methods and tools for entity-relationship querying over the

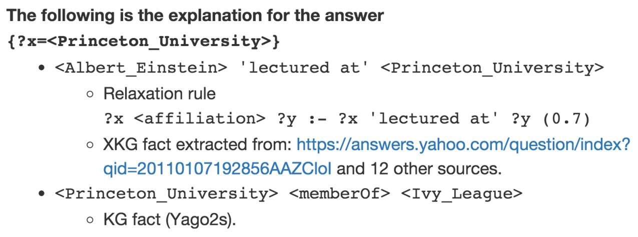


```
SELECT ?x WHERE {
<Albert_Einstein> <affiliation> ?x
?x <memberOf> <Ivy_League>
subject predicate
}
K 10
```

Relaxations

```
?x <affiliation> ?y :- ?x <affiliation> ?z ; ?z 'housed in' ?y (0.8)
?x <affiliation> ?x :- ?x 'lectured at' ?y (0.7)
```

Figure 5: TriniT query interface (screenshot)



The following is the explanation for the answer
`{?x=<Princeton_University>}`

- `<Albert_Einstein> 'lectured at' <Princeton_University>`
  - Relaxation rule
    `?x <affiliation> ?y :- ?x 'lectured at' ?y (0.7)`
  - XKG fact extracted from: https://answers.yahoo.com/question/index?qid=20110107192856AAZClol and 12 other sources.
- `<Princeton_University> <memberOf> <Ivy_League>`
  - KG fact (Yago2s).

Figure 6: TriniT answer explanation (screenshot)

Wikipedia full-text corpus. Both of these projects assume a fixed dataset; they do not gear for the incompleteness of the data and the need for extensions from a variety of sources (like an XKG). Also, none of this related work considers the power of query relaxation.

QaRS [3] provides a graphical querying tool for KGs that supports query relaxation, both automatic and manual, to tackle the discrepancy between the query and the KG. However, there is no attempt to address KG incompleteness.

7. REFERENCES

[1] K. Balog, M. Bron, M. de Rijke: Query Modeling for Entity Search Based on Terms, Categories, and Examples. *ACM TOIS* 29(4):22, 2011

[2] A. Fader, S. Soderland, O. Etzioni: Identifying Relations for Open Information Extraction. *EMNLP* 2011: 1535–1545

[3] G. Fokou, S. Jean, A. Hadjali, M. Baron: QaRS: A User-Friendly Graphical Tool for Semantic Query Design and Relaxation. *EDBT* 2015: 237–252

[4] E. Gabrilovich, S. Markovitch: Computing Semantic Relatedness Using Wikipedia-based Explicit Semantic Analysis. *IJCAI* 2007: 1606–1611

[5] L.A. Galarraga, C. Teflioudi, K. Hose, F.M. Suchanek: AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases. *WWW* 2013: 413–422

[6] R. Gupta, A.Y. Halevy, X. Wang, S.E. Whang, F. Wu: Biperpedia: An Ontology for Search Applications. *PVLDB* 7(7): 505–516, 2014

[7] G. Koutrika, L. V. S. Lakshmanan, M. Riedewald, K. Stefanidis. Exploratory Search in Databases and the Web. *EDBT/ICDT Workshop* 2014

[8] F. Li, H. V. Jagadish. Constructing an Interactive Natural Language Interface for Relational Databases. *PVLDB* 8(1): 73–84, 2014

[9] X. Li, C. Li, C. Yu. Entity-relationship Queries over Wikipedia. *ACM TIST* 3(4):70, 2012

[10] N. Nakashole, G. Weikum, F.M. Suchanek: Discovering and Exploring Relations on the Web. *PVLDB* 5(12): 1982–1985, 2012

[11] M. Theobald, R. Schenkel, G. Weikum. Efficient and Self-tuning Incremental Query Expansion for Top-k Query Processing. *SIGIR* 2005: 242–249

[12] C. Unger, C. Forascu, V. Lopez, A. N. Ngomo, E. Cabrio, P. Cimiano, S. Walter. Question Answering over Linked Data (QALD-4). *CLEF Conf.* 2014

[13] M. Yahya, K. Berberich, S. Elbassuoni, G. Weikum: Robust Question Answering over the Web of Linked Data. *CIKM* 2013: 1107–1116

[14] M. Yahya, D. Barbosa, K. Berberich, Q. Wang, G. Weikum. Relationship Queries on Extended Knowledge Graphs. *WSDM* 2016: 605–614

[15] S. Yang, Y. Wu, H. Sun, X. Yan. Schemaless and Structureless Graph Querying. *PVLDB* 7(7): 565–576, 2014

[16] S. Yang, Y. Xie, Y. Wu, T. Wu, H. Sun, J. Wu, X. Yan. SLQ: A User-friendly Graph Querying System. *SIGMOD* 2014: 893–896

[17] J. X. Yu, L. Qin, L. Chang. *Keyword Search in Databases.* Morgan & Claypool 2009.

[18] C. Zhai. *Statistical Language Models for Information Retrieval.* Morgan & Claypool 2008