Relationship Queries on Extended Knowledge Graphs

Mohamed Yahya[1], Denilson Barbosa[2], Klaus Berberich[1], Qiuyue Wang[3], Gerhard Weikum[1]
[1]Max-Planck Institute for Informatics [2]University of Alberta [3]Renmin University of China
{myahya, kberberi, weikum}@mpi-inf.mpg.de
denilson@ualberta.ca qiuyuew@ruc.edu.cn

ABSTRACT

Entity search over text corpora is not geared for relationship queries where answers are tuples of related entities and where a query often requires joining cues from multiple documents. With large knowledge graphs, structured querying on their relational facts is an alternative, but often suffers from poor recall because of mismatches between user queries and the knowledge graph or because of weakly populated relations.

This paper presents the TriniT search engine for querying and ranking on extended knowledge graphs that combine relational facts with textual web contents. Our query language is designed on the paradigm of SPO triple patterns, but is more expressive, supporting textual phrases for each of the SPO arguments. We present a model for automatic query relaxation to compensate for mismatches between the data and a user's query. Query answers – tuples of entities – are ranked by a statistical language model. We present experiments with different benchmarks, including complex relationship queries, over a combination of the Yago knowledge graph and the entity-annotated ClueWeb'09 corpus.

Keywords

Relationship Queries; Extended Knowledge Graphs; Query Relaxation

1. INTRODUCTION

1.1 Motivation and Problem

Searching for entities and associated properties has received much attention for both web contents and enterprise documents. Examples are queries for "musicians who contributed to movie soundtracks" or "companies that acquired Internet startups". IR-centric approaches typically associate entities with statistical language models in order to match and rank entity mentions and surrounding phrases in text corpora [3]. Semantic-Web-style approaches, on the other hand, rather tap structured knowledge graphs (KGs) such as Freebase or Linked Open Data (LOD) collections such as combinations of DBpedia, Yago, and MusicBrainz, and use SPARQL queries to retrieve relevant RDF triples [16].

WSDM'16, February 22–25, 2016, San Francisco, CA, USA.
© 2015 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-3716-8/16/02…$15.00
DOI: http://dx.doi.org/10.1145/2835776.2835795

Neither of these paradigms provides good support for *relationship queries* that connect multiple entities in a specific way and *return tuples of connected entities*. Consider, for example, the task of finding songs that appear in movies and returning a list of ⟨song, movie⟩ pairs. This cannot be fully expressed by IR-centric entity search which is bound to return spurious results where a movie is merely mentioned in the textual proximity of a song (e.g., "My Way" and the film "The Man with the Golden Arm" in a Frank Sinatra biography). On the other hand, the SPARQL language supports expressive and precise queries over RDF graphs of entity-relationship triples, but its results are limited by the properties (i.e., relation types, binary predicates) and facts (i.e., relation instances, predicate arguments) that the underlying KG or LOD collection contains. So none of the established paradigms can adequately cope with relationship queries.

Examples: In principle, the above example could be formulated by the following SPARQL query:

```
SELECT ?s ?m WHERE {
   ?s type song .  ?m type movie .  ?s musicInFilm ?m }
```

where `?s` and `?m` are variables and the second line contains three triple patterns over the subject-predicate-object (SPO) triples of the underlying KG. The query should return bindings for song-movie pairs that are in the desired relationship. However, this works only if the KG does indeed offer the predicate *musicInFilm* and that predicate is sufficiently well populated. If the KG instead contained predicates *filmHasSoundtrack* of type *movie* × *album* and *albumContainsSong* of type *album* × *song*, the user would need to formulate a very different query and non-expert users would typically fail to get this right. Even if the user succeeded in posing the best query formulation, the answers would be limited by the facts of the KG, while the web or social media could potentially hold many additional answers.

Similar cases arise in domains like business or sports, for example, when searching for "South-American football players and European championships that they have won" (with answers such as Lionel Messi and the UEFA Champions League). Here one challenge that users would struggle with is to properly formulate the two-hop join query with triple patterns `?p type player . ?p playedFor ?t . ?t won ?c . ?t type footballClub` as KGs associate championships with teams, not players. IR-style entity search would be more convenient for users, but loses precision and would be misled by co-occuring players and teams who were opponents in final matches.

Problem Statement: The above information needs are examples of *relationship queries*, which we define here as SPARQL-like SPO queries with two or more distinct variables joined by one or more relationships. Such queries typically have two or more variables in their projection list (the `SELECT` clause). This is in contrast to entity search queries where there is a single variable in the query, whose bindings are the query result. The problem addressed in this work is how to express and process *relationship queries* over both *structured KGs and unstructured text/web corpora*. We emphasize the coexistence and combination of KG and text, as they complement each other in terms of potential answers.

Although we cast this problem into a structured query language like SPARQL (and will extend this language), it should be noted that users such as analysts or journalists need a less formal UI. Our methods can provide a form-filling UI, which is user-friendly yet allows to generate (extended) SPARQL queries. Natural-language question answering (QA) [19] and the translation of questions into queries [7, 13, 35] would be a suitable UI as well.

1.2 Approach and Contribution

We developed a comprehensive system, TriniT (for "Triples & Text"), to address the problem of relationship queries. TriniT builds on three insights and novel contributions.

First, we extend a KG by running Open Information Extraction (OIE) techniques [24] on a large Web corpus and capturing triples of textual expressions along with their surrounding contexts. By means of Named Entity Disambiguation (NED) techniques [17], we obtain a combined dataset with structured facts and text triples as first-class citizens. Prior work [9–11] has looked into the interplay of KG and text corpora, but limited this to either using distant knowledge for enhancing text search or using textual witnesses for fact statistics in the KG.

Second, for querying the extended KG, XKG for short, we extend the SPARQL notion of triple patterns to allow each of the SPO components to be a URI denoting entities, classes or predicates, or a typed literal (e.g., dates), or a textual phrase. Note that the W3C standard for SPARQL requires S and P to be URI's and O to be a URI or a literal. By supporting text in each of the SPO positions, we can formulate queries that tap into both structured and textual triples simultaneously. For example, we can find more song-movie pairs, beyond what the KG offers, by the triple pattern `?s "appeared in" ?m` (in conjunction with other triple patterns). Prior work on extending SPARQL with text predicates [11] has treated text as annotations of entire SPO facts, typically based on keyphrases that are associated with the S and O entities, and advocated SPO+text (SPOX) quad patterns. On many queries, this loses precision as the system cannot distinguish anymore if the X component specifically refers to S, P, or O.

Third, despite the additional convenience of using text for SPO components, the potential mismatch between the user's vocabulary and the names and phrases in the KG and XKG still makes proper query formulation a difficult task. To overcome this obstacle, TriniT provides a framework for query relaxation rules, and algorithms to automatically apply rules to generate relaxed queries that yield higher recall. For example, a user-provided triple pattern `?p won ?c` for the (football player, championship) example query can be automatically rewritten into a text-based pattern `?p "has won" ?c` (which can find more matches in the text corpus with entity linkage to the KG) or into the more elaborate and appropriate patterns `?p playedFor ?t . ?t won ?c`. The answer-ranking model of TriniT considers the potential target drift of such relaxations, and computes overall rankings for the results of different rewritings. Prior work on query relaxation for RDF and XML data [1, 11, 12] was restricted to simple structural rewritings like replacing a predicate by a variable or lexical rewritings like replacing a class with a super-class.

The novel contributions of this work can be summarized as follows:

- an extension of SPARQL triple patterns supporting expressive search over extended KGs in a convenient manner, along with a judiciously designed answer-ranking model;
- a framework and algorithms for relaxation rules that support the automatic rewriting of user queries to improve recall while preserving the query focus;
- experiments with complex relationship queries over the combination of the YAGO knowledge graph and the ClubWeb'09 corpus, demonstrating the viability and benefits of our model and methods.

2. DATA MODEL AND QUERY LANGUAGE

2.1 Extended Knowledge Graphs

In the standard setting of an RDF-based knowledge graph (KG), we have two sets of objects: i) *resources* R encoded as URI's for entities, classes, or predicates (called properties in RDF), and ii) *literals* L for values like numbers, dates, etc. Predicates $P \subseteq R$ are a special kind of resource used to create facts that describe resources.

Definition 1 (Knowledge Graph (KG)). *A knowledge graph (KG) is a set of triples from* $(R \times R \times R) \cup (R \times R \times L)$.

The three components of a triple are called the *subject*, *predicate*, and *object*, or SPO for short. When O is a resource, like an entity or class, the triple captures a relationship (e.g., someone's spouse or membership in a semantic class); when O is a literal, the triple expresses an attribute value for an entity (e.g., someone's birthdate or salary). A predicate can be seen as a binary relation, whose arguments are the subject and object. Alternatively, the subject and object can be seen as vertices connected by an edge labeled with the predicate.

However big a KG could be, it is bound to provide incomplete knowledge, as there is always additional detail and finesse that could be found in additional text sources. The goal of extending a KG is to account for missing entities, classes, predicates, or facts. To do this, we combine the KG with one or more textual corpora annotated with entities and predicates, some of which may already exist in the KG whereas others are missing so far. The triples are extracted from text by information extraction (IE) techniques, and become additional knowledge which can be used to bridge the gap between the information needs of users and the KG. More specifically, we use Open IE methods like [24] to extract triples that consist of two noun phrases (for S and O) that are connected by a noun phrase, verb phrase, or a preposition (for P). We additionally employ methods for

Named Entity Disambiguation (NED) like [17] to map the two noun phrases into entities registered in the KG. This process is prone to error, but can be tuned to favor precision over recall. In our experimental studies, the IE and NED error rates are low, and their benefit clearly outweighs the errors.

As an example, suppose that the original KG contains the triples in the left column of Table 1 (for simplicity we use names for entities, classes, and predicates rather than URI's which would be required by the official RDF standard).

Additionally, assume that Open IE has identified further triples shown in the right column of Table 1 where some of the SPO components are mere text phrases as they could not be mapped to any entity, class, or predicate in the KG. The union of all these triples forms the extended knowledge graph, XKG for short. For an XKG we do no longer refer to resources and literals, but instead collectively call all of these simply tokens C.

Definition 2 (Extended Knowledge Graph (XKG))**.** *An extended knowledge graph (XKG) is a bag of triples over tokens; that is, triples are from* $C \times C \times C$.

The XKG being treated as a bag of triples takes into account that the IE process can produce the same triple many times from different documents. We exploit this kind of redundancy – not present in the original pure KG – in our ranking model.

2.2 Triple Pattern Queries

We now define the triple pattern query language for querying XKGs. Let V be an infinite set of variables, distinct from tokens, and always prefixed with a question mark.

Definition 3 (Triple Pattern and its Results)**.** *A* triple pattern *q is a triple from $V \cup C \times V \cup C \times V \cup C$. The results of a triple pattern are triples t from the XKG such that the tokens in q that are not variables are matched by t. For the variables in q, the corresponding tokens in t are called bindings.*

Examples of triple patterns are: `?x usedIn KillBill` (with a predicate and an entity as tokens) and `?x` *"appears in" "Kill Bill Vol 1"* (with two phrases as tokens).

Definition 4 (Query)**.** *A query $Q = \{q_1, ...q_n\}$ is a set of triple patterns q_i and a projection list $P(Q)$ of variables (typically with variables occurring in multiple q_i, forming join predicates). We require the join graph that is constituted by the q_i as vertices and edges between them whenever two of the triple patterns share a variable, to be a connected graph (to avoid computing Cartesian products). $P(Q)$ is a (usually proper) subset of the variables in Q, defining the output structure (typically entity tuples).*

Relationship queries are those containing at least two distinct variables, requiring one or more relationships to join them. In such queries, the projection list typically contains two or more variables ($|P(Q)| \geq 2$), in which case answers are proper tuples. This is in contrast with traditional entity search which have a single variable, hence $|P(Q)| = 1$.

Definition 5 (Query Answer)**.** *For query Q, an answer (a) is a mapping of the variables in Q to tokens in C. Applying an answer (a) to a triple pattern $q_i \in Q$ results in the triple t_i, we denote this by $a(q_i) = t_i$. The restriction of a query answer to bindings of variables in $P(Q)$ is called a projected answer, denoted a_P.*

Example query 1: Finding movies with British songs could be expressed as:

```
SELECT ?s ?m WHERE {
   ?s type song .  ?m type movie .  ?s usedIn ?m .
   ?s performedBy ?x .  ?x bornIn UK }
```

Note that the KG in Table 1 alone would have no results to this query, whereas the XKG yields the desired result `(WalterMitty, SpaceOddity)`. Alternatively to the triple pattern `?a` *"born"* `UK`, we could specify the condition `?a ?p` *"British"*. Neither of these is easy to formulate by a user who does not know the XKG data really well.

Example query 2: Finding movies with American songs and their singers requires both elaborate use of long paths in the XKG (i.e., join chains) and the use of text-based triples in the XKG:

```
SELECT ?m ?s ?x WHERE {
   ?s type song .  ?m type movie .
   ?m hasSoundtrack ?a .  ?a contains ?s .
   ?s performedBy ?x .  ?x ?p "American" }
```

Again, formulating such a sophisticated query is awfully hard for a user who does not know details of the underlying XKG. We will see in Section 3 that such sophisticated queries can be generated by automatically rewriting a simpler user query based on query relaxation rules.

3. QUERY RELAXATION

Query formulations by a user may fail to return the expected answer(s), either because the user has insufficient understanding of the predicates, classes, and entities in the XKG, or because no answer can completely satisfy all conditions in the query.

Our solution to the above problems is *query relaxation*, where one or more parts of the query are automatically rewritten in order to compute answers that are likely to satisfy an information need but cannot be obtained by the original query. We first define our framework for query relaxation, and then discuss specific choices of relaxation rules.

Definition 6 (Relaxation Rule)**.** *Given a query $Q = \{q_1, ..., q_n\}$, a relaxation rule is a triple $r = (\mathbf{q}, \mathbf{q}', w)$, where $\mathbf{q} \subseteq Q$ and is non-empty, $\mathbf{q}'$ is a set of triple patterns, and $w \in [0, 1]$ is a relaxation weight that captures the closeness between $\mathbf{q}$ and $\mathbf{q}'$.*

Definition 7 (Relaxation Rule Application, Relaxed Query)**.** *Given a query Q and a relaxation rule $r = (\mathbf{q} \subseteq Q, \mathbf{q}', w)$, the application of r to Q results in query $Q' = r(Q) = (Q \setminus \mathbf{q}) \cup \mathbf{q}'$ with $P(Q') = P(Q)$.*

Given a query Q and a sequence of relaxation rules $\mathbf{r} = (r_1, ..., r_m)$, a relaxed query is a query $Q' = r_m(...r_1(Q))$.

Rationale: As an example, reconsider the example queries of Section 2. The user may issue the seemingly perfect query:

```
SELECT ?s ?m WHERE {
  ?s type song .  ?m type movie .  ?s usedIn ?m .
  ?s performedBy ?x .  ?x bornIn UK }
```

to retrieve British songs in movies. However, this may not return any results at all. To retrieve good candidates, we could drop an entire triple pattern from the query such as the one about birth in the UK, or we can relax overly restrictive predicates like `bornIn` by a textual token like *"born"* or even by a variable that can be matched by any predicate or token. Likewise, if the specified entity token `UK`

KG Triples			XKG Triples		
S	P	O	S	P	O
BangBang	type	song	DavidBowie	*"born and lives in"*	UK
SpaceOddity	type	song	DavidBowie	won	*"best British singer"*
KillBill	type	movie	BangBang	*"by"*	NancySinatra
WalterMitty	type	movie	*"Sinatra's daughter"*	bornIn	USA
SpaceOddity	usedIn	WalterMitty	NancySinatra	*"an"*	*"American singer"*
SpaceOddity	performedBy	DavidBowie	*"Lonely Shepherd"*	*"appears in"*	KillBill
KillBill	hasSoundtrack	KillBillAlbum	*"Lonely Shepherd"*	performedBy	*"Zamfir"*
KillBillAlbum	contains	BangBang	*"Zamfir"*	bornIn	Romania

Table 1: Example triples in KG (left) and additional samples in XKG (right).

is not directly matched in the XKG, we could generate semantically related tokens like *"British"*, *"English"*, *"Scottish"*, *"from London"*, etc. All these variants can be automatically generated in our relaxation framework.

Obviously, some relaxations drift away from the original user request. For example, *"Scottish"* or *"from London"* seem to be less focused in approximating the original UK than *"British"* – so their relaxation weights should be lower than the weight for *"British"*. Of course, replacing UK by a variable would be an even stronger relaxation and should have an even lower weight.

The order in which relaxations are applied results in different queries. In Section 4 we discuss our query ranking and processing schemes. The two are designed to incrementally explore relaxations only if they can contribute to producing the top-k-scoring answers. This approach avoids explicit enumeration of all possible relaxations of a query, which can be prohibitively expensive.

Relaxations in TriniT: TriniT provides interfaces for users to implement their own relaxations. In this work we consider two concrete forms of relaxation that are relevant to the benchmarks we consider: structural relaxations and predicate paraphrasing. Structural relaxations result in replacing a triple pattern by a set of triple patterns that conceptually denote a path. We apply this to spatial predicates that connect an entity with a location (e.g. from (?x bornIn UK) to (?x bornIn ?y . ?y locatedIn UK)). We define an operator that instantiates such rules based on the query. Such rules can also be automatically mined from the XKG using the method of [14].

Predicate paraphrasing is the most important type of relaxation we consider in this work. We generate paraphrases for XKG predicates using the XKG itself. For each predicate (e.g., graduatedFrom), we generate paraphrases (e.g., *"went to"*) and inverse paraphrases (e.g., *"alumnus"*) by considering the overlap between predicate arguments in the XKG. Given two predicates (incl. textual ones), p_1 and p_2, where $args(p_i) = \{(s,o),(s,p_i,o) \in XKG\}$ (i.e., subject-object pairs co-occurring with p_i), the weight assigned to the relaxation $r = (\{?x\ p_1\ ?y\}, \{?x\ p_2\ ?y\}, w)$ is:

$$w = \frac{|\,args(p_1) \cap args(p_2)\,|}{|\,args(p_2)\,|}.$$

Inverse paraphrases are generated and weighted in the same manner as above, by matching the XKG with an inverted version of itself, where the subject and object components of a triple are switched. In our experiments, we do not consider paraphrases that are stop words, as these consistently hurt results. Table 2 shows examples of paraphrases and inverse paraphrases (indicated by the $^{-1}$ superscript) for both KG and textual relations extracted from our XKG.

Predicate	Paraphrase
graduatedFrom	*"graduated from"*
graduatedFrom	*"went to"*
graduatedFrom	*"alumnus"* $^{-1}$
"performed by"	*"recorded by"*
"performed by"	*"singer"*
"performed by"	*"performance of"* $^{-1}$

Table 2: Example predicate paraphrases

Approaches such as [15] can also be utilized to obtain more paraphrases.

4. ANSWER RANKING AND QUERY PROCESSING

The scoring model of TriniT is based on a statistical language model (LM) for triple patterns, the basic building blocks of a query. For composing the triples from different sub-queries into overall answers, we judiciously aggregate the scores of the retrieved triples. Upfront, it is not obvious how to do this in a principled and efficiently implementable manner. Our setting with joins, queries over many variables, and bindings for multiple variables in the projected output is very different from established language models for entity search [2]. Also, by treating triples obtained from OpenIE as first-class citizens instead of using text to contextualize triples, it differs from existing language models [11, 12] for ranking and relaxation in knowledge graphs.

Answers for Single Triple Patterns: In analogy to the traditional IR setting, we can view a triple pattern as a document which generates individual triples. In this generative setting, we define a language model for each such triple pattern using a mixture model as follows:

$$P(t \mid q_i) = \lambda \frac{\#t}{|q_i|} + (1-\lambda)\frac{\#t}{|XKG|},$$

where $\#t$ denotes the number of occurrences of triple t in the XKG, $|q_i|$ is the total number of triples matching q_i in the XKG. Likewise, $|XKG|$ is the total size (i.e., number of triples) of the XKG and λ is a tunable parameter between 0 and 1. The first term is defined to be non-zero only if t matches q_i. The above defines a proper probability distribution: for each q_i, summing up over all triples in the XKG will always give us 1.

There are some subtle differences between this setting and the traditional IR setting. Smoothing with a background model (the XKG in our case) serves two purposes in traditional IR, namely to avoid zero probabilities and to attain an *idf*-like effect [38]. In our setting, since we only consider triples t that match the triple pattern q_i, zero probabilities are not an issue. Further, a relative weighting of triple patterns, corresponding to the *idf*-like effect, is already obtained by considering triple-pattern selectivities $|q_i|$, that is, how many matching triples exist. For our mixture model, the parameter λ thus controls whether the probabilities $P(t|q_i)$ are only based on the number of occurrences of t (for $\lambda = 0$) or also consider triple-pattern selectivities (for $\lambda > 0$), resulting in a relative weighting of triple patterns in the query.

Answers for Entire Queries: With answer scores defined for single triple patterns, we compute the score of a complete answer a for a composite query Q as follows:

$$\mathit{score}(a, Q) = \prod_{q_i} P(a(q_i)|q_i),$$

where $a(q_i) = t$ is the XKG triple resulting from the application of answer a to q_i.

Multiple answers can produce the same projected answer a_P, which is the result the user is interested in. For example, in a query with variables for movies, songs, and artists, if the final output is required to be pairs of merely movies and artists, the songs are "projected away", and we may obtain duplicates of the same movie-artist pair.

Thus, for each binding of the projection variables we need to define how to aggregate the scores of the individual results (with bindings for all three variables) for the whole group of duplicates. While summing up scores seems a natural choice, it incurs two problems: i) inflating the score of answers with frequently occurring entities (e.g., artists with many songs in many movies), and ii) forcing the query processing to retrieve all duplicates for each projected answer as they contribute to the scoring. Especially the second point would be critical from an efficiency perspective and rule out early pruning when scanning posting lists during query processing. Therefore, we opt to use the maximum score for each group rather than the sum over all duplicates, and define the score of a projected answer of a query as

$$\mathit{score}(a_P, Q) = \max_{a: a_P \subseteq a} \mathit{score}(a, Q).$$

Scoring with Query Relaxation: We finally extend our scoring model to account for query relaxation. Starting with a user-provided query, Q, we relax it in one or more steps by applying a sequence of relaxation operators $r_1, ..., r_n$ to obtain $Q' = r_n(...r_1(Q))$. Each relaxation operator r_l carries a relaxation weight $w_l \in [0, 1]$ which reflects how much a relaxed query drifts away from the previous query.

The score of an answer obtained from a relaxation is defined as:

$$\mathit{score}(a, Q, Q') = \prod_{l=1}^{n} w_l \times \mathit{score}(a, Q').$$

Intuitively, the score of a with respect to Q' is attenuated to reflect the divergence of Q' from the original query Q. For the special case where $n = 0$, i.e. $Q' = Q$, with no relaxation operators invoked, $\mathit{score}(a, Q, Q') = \mathit{score}(a, Q)$.

Because it is possible to generate the same answer through multiple distinct relaxations, we define the score of an answer with respect to the original query and the space of all possible relaxations $\mathbf{r}$ as:

$$\mathit{score}(a, Q, \mathbf{r}) = \max_{\substack{Q' = r_n(...r_1(Q)),\\ r_l \in \mathbf{r}}} \mathit{score}(a, Q, Q').$$

Here, an answer is assigned the maximum score it can obtain from any of the possible relaxations of the query (including the original query). The rationale for this design decision is analogous to the above score aggregation over duplicates when variables are projected away and in line with [31]: we want to avoid unduly inflating the influence of seeing the same answer many times in different contexts and allow for early pruning in query processing.

Finally, the score of a projected answer in a setting with relaxation is defined as:

$$\mathit{score}(a_P, Q, \mathbf{r}) = \max_{a: a_P \subseteq a} \mathit{score}(a, Q, \mathbf{r})$$

Query Processing in TriniT is a natural fit for the top-k paradigm [20] where the goal is to produce the best k answers with the highest scores while accessing as little of the data as possible. Our implementation uses *inverted indexes* to retrieve matching triples in decreasing order of their probability $P(t|q_i)$ from the XKG. This score monotonicity ensures that the algorithm can maintain result candidates in a priority queue with upper bounds for their final scores, and apply a threshold test for early termination. Our query processor applies the *incremental merge* algorithm [31] to automatically invoke relaxations judiciously and combine answers for a triple pattern and its relaxations.

5. EXPERIMENTAL EVALUATION

We report on an experimental evaluation that demonstrates the effectiveness of TriniT and provides insights into answering relationship-centric queries. We make our results publicly available (http://mpii.de/yago-naga/TriniT).

5.1 Methods

We compare two different configurations of TriniT with three natural state-of-the-art baselines. Figure 1(c) shows an example query for each system. The first TriniT configuration (TriniT-Relax) processes queries without considering relaxations. We contrast this to TriniT+Relax where TriniT has access to relaxations that it can automatically invoke as needed during query processing.

The first baseline we consider (ES) is based on the work by Balog et al. [2] for entity search. Here, an entity is represented by two fields containing the semantic types it belongs to and a textual description. We use Model 4, which is most effective one that allows us to enforce type constraints that are crucial for good results. The ES approach cannot return tuples in response to relationship queries, so we formulate queries to ask about a single entity (a single variable).

The second baseline (ERS) is that by Li et al. [23] for entity-relationship search. Here, queries are evaluated over an entity-annotated corpus. The score of a match of a query condition (a textual description of a typed variable or textual relation connecting two such variables) depends on the proximity of phrases in the condition and variable bindings.

Finally, we compare TriniT with SPOX(SPO+teXt), an extended version of the approach by Elbassuoni et al. [11]. Each SPO triple pattern in the KG is associated with textual keywords, which are an aggregation of keywords associated with the S and O components of the triple. For example, the triple `RusselCrowe actedIn ABeautifulMind` would be associated with the set of keywords {*'john', 'nash', 'true', 'story', 'princeton','australia'* ...}. On the query side, SPO triple patterns can optionally come with teXtual conditions that specify constraints which cannot be expressed in structured SPO

`[ALGOL-JohnBackus-TuringAward]`

(a)

"Programming languages invented by a Turing Award winner."

(b)

TriniT:

```
SELECT ?x ?y WHERE {?x type programming_language .
 ?y type person .  ?y "invented" ?x .
 ?y won TuringAward}
```

ES:

```
type:(programming_language) text:(programming language
invented by a turing award winner)
```

ERS:

```
SELECT ?x ?y FROM programming_language ?x, person ?y
WHERE ?x:["won", "turing award"] AND ?x,?y:["invented"]}
```

SPOX:

```
SELECT ?x ?y WHERE {?x type programming_language .
 ?y type person .  ?x ?r ?y ["invented"] .
 ?y won TuringAward}
```

(c)

Figure 1: Query generation & formulation example

form over the KG. For example, the SPOX pattern `?x actedIn ?y`[*'true story'*] is used to for actors in a movie based on a true story. During query answering, the SPO parts of the query are first matched against the KG, and the answers are subsequently ranked using language models that consider witness counts of KG triples as well teXtual keyword conditions with their frequencies. The original SPOX approach includes a form of relaxation in which entities or relations in a query are replaced by variables. We improve on this by moving these entities or relations to the X component of a triple instead of completely discarding them. This way they can still influence the final ranking of answers. We report results from this improved approach. We do not consider [12] as a baseline since relaxations here are performed by replacing KG entities and relations with other KG relations, which almost always results in semantic drift, as KGs rarely contain redundancy.

5.2 Benchmarks

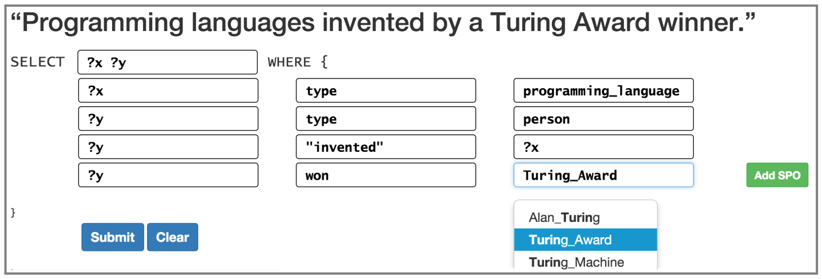
Existing entity-search queries tend not to be relationship-centric. A contribution of this work is a new set of 70 inherently relationship-centric information needs, referred to as COMPLEX queries here (e.g., *"Programming languages invented by people who won the Turing Award."*). We next describe how these queries were generated. A query was constructed starting from a *chain* of entities (e.g., `[ALGOL-JohnBackus-TuringAward]`) where some become part of the query and others serve as an answer. These chains are automatically sampled from domains within the XKG, where a domain is the set of entities that fall within a specified set of semantic types. The domains we consider are cinema, music, books, sports, computing, and military conflicts. The cinema domain, for example, includes entities of the types `actor`, `show`, `director`, `award`, and `producer`. Within each domain, we iteratively sample entities starting from a pivot entity to form a chain. The first pivot entity (`ALGOL` in our example) is sampled non-uniformly based on a popularity prior from the domain. Next, we find the 20 entities in the domain that have the highest coherence with the current pivot by the Wikipedia-link measure of Milne & Witten [25]. We then sample these entities non-uniformly, based on the number of XKG facts connecting them to the pivot to choose the next pivot. This process is repeated to obtain chains of size 2-4 (determined randomly). A human annotator then constructs a question from the chain asking for the first entity while containing multiple unknowns corresponding to other entities in the chain. An annotator can discard a chain if she thinks no interesting question can be generated from it. Figure 1(a) shows an example of a chain, and (b) shows the corresponding question formulated by the human annotator. While the question is constructed by considering a single chain, it can have many answers. In the example of Figure 1, `(Pascal,NiklausWirth)` and `(Smalltalk,AlanKay)` are two possible answers among several others.

We additionally ran experiments with two benchmarks from previous work. The first, ESQ, is a set of 485 entity-centric queries compiled by Balog & Neumayer [4]. We remove from this dataset SemSearch ES and INEX LD queries as they do not fit our setting. SemSearch ES contains queries such as *"YMCA Tampa"* and *"nokia e73"*, which refer to a specific entity with no relations at all. INEX LD, is highly keyword centric (e.g. *"allegedly caused World War I"*) with a very noisy gold standard (e.g. `Aerial_bombing_of_cities` is considered a relevant entity for the above query). This leaves us with 255 queries from which we remove 37 involving aggregation (e.g., *"movies with eight or more Academy Awards"*) as these are beyond the capability of all the systems in this experiment, leaving us with 218 queries. Unlike our COMPLEX queries, ESQ queries ask for individual entities rather than tuples, and are usually expressed in the form of a type (e.g., *"EU countries"*) or a type with a description that contains a single relation (e.g., *"movies directed by Francis Ford Coppola"*).

The other benchmark from prior work, which we call ERQ, is constructed by Li et al. [23] and consists of 28 queries. 22 of the queries in this dataset are similar to the ones in ESQ, with 6 only asking for pairs of entities.

System Input Generation: TriniT and each of the three baselines described in Section 5.1 expects a specific form of query as shown in Figure 1(c). In the first step, an information need in the form of a question (Figure 1(b)) is shown to a human annotator who is asked to translate it into a proto-query. In the second step, the proto-query is used to automatically create queries for the four systems using a set of rules described below. The annotator, after being shown four examples of question-to-proto-query translation, is presented with the UI shown in Figure 2. Here, each row provides SPO fields for specifying SPO triple patterns. Each field provides auto-completion functionality for KG entities (S and O) and predicates (P). The annotator is asked to express the given question in SPO form, and is instructed to use the auto-completion suggestions when appropriate, or resort to textual tokens if necessary.

For ES, the type field is filled with the type associated with the first variable in the proto-query, and the text of the question is used in the text field. For ERS and SPOX, variable type-constraints are maintained. ERS cannot deal with KG entities and relations in the query (it only returns entity tuples as results), so they are mapped to their textual form. SPOX can only deal with entities and relations in the S and P components, respectively, but not textual tokens. To accommodate this, we extend an SPO triple pattern in the proto-query with an X component and move the S/O textual component there – note that this is not the


"Programming languages invented by a Turing Award winner."

```
SELECT  ?x ?y   WHERE {
        ?x      type        programming_language
        ?y      type        person
        ?y      "invented"  ?x
        ?y      won         Turing_Award
}
```

Figure 2: Proto-query interface, with input for the question in Figure 1(a).

same as SPOX relaxation described above, which is part of query processing. In real life, we envision a system used by professionals such as journalists and researchers willing to invest some learning effort in exchange for the more expressive querying they get in return from the different types of queries, with support from appropriate UIs.

5.3 Data

We finally describe the data we use for TriniT and the various baselines we consider. We note that we run our own implementations of the baselines on a scale larger than what was previously reported.

We use as our KG Yago2s, whose predicates connect entities from Wikipedia to other entities (e.g. `TomHanks actedIn ForrestGump`), literals (e.g. `TomHanks birthDate 1956-7-9`), or semantic types (e.g. `TomHanks type actor`), with a total of 48M triples (44M class assignments, and 4.4M relations and attributes).

As a text corpus, we use the FACC1 dataset, which annotates text spans in the ClueWeb'09 corpus with entities linked to Wikipedia entities (via Freebase) with a precision and recall estimated to be 80-85% and 70-85%.

For TriniT, we construct the XKG by combining the KG described above with the result of a simple yet effective open information extraction scheme over the annotated ClueWeb corpus as follows. We look for pairs of entity annotations in the same sentence separated by a string of at most 50 characters, and create a triple where the two entities are the subject and object, and the separating string is the relation connecting them. In this way we obtain 392M extractions, resulting in 65M unique triples. For predicate paraphrases, we run the predicate paraphrasing scheme described in Section 3 on top of the XKG to generate 172M pairs of scored predicate paraphrases like those in Table 2.

We use the annotated ClueWeb corpus for ERS as we did for TriniT. In addition, ERS takes type associations from our KG, ensuring ERS has data comparable to that used by TriniT. We attempted to evaluate the three benchmarks on the ERS online demo. However, it contained only a subset of the entities and types needed to answer COMPLEX and ES queries, preventing a fair comparison with other systems. For SPOX, we use our the annotated ClueWeb corpus to associate keywords with entities and subsequently KG triples. To do this we find all words that occur in the same sentence as an entity and keep those with a positive association (using normalized PMI) with the entity. SPOX uses the same KG as TriniT, Yago2s.

In ES we use an entity's Wikipedia page to populate its textual description, in line with [2]. We tried to extend this by adding to this field sentences from the annotated ClueWeb corpus that mention the entity. However, this resulted in worse retrieval effectiveness. We populate the semantic type field for an entity from the types it is associated with in the KG.

	ESQ (218)			
	P@5	MAP	NDCG	R
ES	0.183	0.138	0.211	0.093
ERS	0.182	0.150	0.232	0.119
SPOX	**0.249**	**0.218**	**0.336**$^{\triangle}$	**0.188**
TriniT-Relax	0.158	0.140	0.192	0.093
TriniT+Relax	0.218	0.190	0.287	0.156

	ERQ (28)			
	P@5	MAP	NDCG	R
ES	0.492	0.388	0.489	0.236
ERS	0.408	0.353	0.387	0.177
SPOX	0.577	0.561	0.580	0.248
TriniT-Relax	0.467	0.448	0.502	0.174
TriniT+Relax	**0.637**	**0.614**	**0.692**	**0.267**

	COMPLEX (70)			
	P@5	MAP	NDCG	R
ES	0.132	0.104	0.172	0.115
ERS	0.249	0.243	0.322	0.234
SPOX	0.243	0.237	0.250	0.134
TriniT-Relax	0.370	0.360	0.419	0.258
TriniT+Relax	**0.603**$^{\triangle}$	**0.594**$^{\triangle}$	**0.775**$^{\triangle}$	**0.613**$^{\triangle}$

Table 3: Experimental results.

5.4 Results and Analysis

Table 3 shows our experimental results. Following earlier work, we use precision@5, NDCG (with binary relevance), MAP, and recall as quality measures. For queries with an empty results list, we define all measures to be 0.

For ESQ queries, we use the gold standard provided with the benchmark. For ERQ and COMPLEX no gold standard is given, so we crowdsource relevance judgments and determine the relevance of an answer by majority vote of three judges. Note that since the XKG contains noise, human annotators were instructed to base their judgments on the real world rather than on the XKG. Inter-annotator agreement was measured using Fleiss' kappa to be 0.837 indicating almost perfect agreement. In all cases we use binary relevance.

ERQ and COMPLEX queries do not easily lend themselves to computing the complete set of relevant results, with queries such as *"NBA teams married to actresses, and the teams they play for"* (COMPLEX) or *"people born in Spain"* (ERQ). To compute NDCG and recall, we use pooling from the various systems to create a golden standard. Since ES returns single entities and not tuples, we project all answers on the same dimension as the one used for the ES query and compute the golden standard over that dimension.

$^{\triangle}$Significant improvement (two-tailed paired t-test, $p < 0.01$).

I- "Spouses of actors who graduated from an Ivy League university."

Query:
```
SELECT ?x ?y ?z WHERE { ?x type person .
  ?y type actor .  ?z type university .
  ?y graduatedFrom ?z .  ?x marriedTo ?y .
  ?z "member of" IvyLeague }
```

XKG:
```
ChristopherReeve graduatedFrom JuilliardSchool
ChristopherReeve "went to"   CornellUniversity
```

Relaxations: `(?w graduatedFrom ?z)→(?w "went to" ?z): 0.066`

Relevant answers:
-TriniT-Relax: ϕ
-TriniT+Relax:
`{(DanaReeve, ChristopherReeve, CornellUniversity)}`

II-"Lieutenant governors of the province where Ottawa is located."

Query:
```
SELECT ?x ?y WHERE {?x type province .
  ?y type person .  Ottawa locatedIn ?y .
  ?x "lieutenant governor of" ?y }
```

XKG:
```
Ottawa locatedIn NationalCapitalRegion
NationalCapitalRegion locatedIn Ontario
Ontario "lieutenant governor"   DavidOnley
HilaryWeston "lieutenant governor of"   Ontario
```

Relaxations:
```
(?w locatedIn ?z)→(?w locatedIn ?u .  ?u locatedIn ?z):  1.0
(?w locatedIn ?z) → (?w "part of" ?z):  0.073
(?w "lieutenant governor of" ?z)→
      (?z "lieutenant governor" ?w):  0.259
```

Relevant answers:
-TriniT-Relax: ϕ
-TriniT+Relax:
`{(DavidOnley, Ontario), (HilaryWeston, Ontario)}`

Figure 3: Anecdotal examples of results

Figure 3 gives illustrative examples from our COMPLEX queries set. We discuss the results by benchmark next.

ESQ Queries: Here SPOX outperforms all systems. On these non-relationship centric queries, SPOX boils down to an improved version of ES. If a query can be formulated in a structured manner, then this tends to be reflected in SPOX query formulations, resulting in SPO only queries, without the X components. When this is not possible, most of the query conditions end up in the X component of a type-constraint triple pattern either by formulation or through the improved SPOX relaxation scheme we described above.

TriniT is penalized against SPOX on queries that can be satisfactorily answered with keywords without the need to establish crisp relationships, as TriniT requires. For example, the query *"Nordic authors known for children's literature"* is reduced in the SPOX model to `?x type author[`*'nordic children's literature'*`]`. Here, looking at the co-occurrences of an author and the keywords suffices to return good answers. On the other hand, the SPO query formulation used for TriniT, with P set to *"known for"*, could not find matches in the XKG, even with relaxation. This result demonstrates that keyword based querying is an effective paradigm for a certain class of queries. Once we move to more relationship-centric queries below, we start observing the advantage of TriniT's approach.

We can already observe the advantages of relaxation at this stage. For example, on a simple query asking for *"Italian Nobel winners"*, the TriniT query uses the KG predicate `won`. While this looks reasonable, the KG only lists winners of specific Nobel prizes (e.g., `NobelPrizeInLiterature`). Only by relaxing `won` to the inverse textual predicate, *"winner"*, is the TriniT query able to return the correct answers from triples like `(NobelPrize "winner" EnricoFermi)`.

It is also interesting to observe the relatively close performance of ES and ERS. For most ESQ queries, ERS will reduce to ES, with a type constraint and a set of keywords describing the target entity, but with different scoring schemes.

ERQ Queries vary in how relationship-centric they are. The majority of ERQ queries (22/28) ask about a single entity, not a tuple, requiring no joins. The rest, while they ask for a tuple, can all conceivably be answered from an individual document describing a relevant entity. ERQ queries like *"films starring Robert de Niro, and their directors"* and *"Novels and their Academy Award winning film adaptations"* issued to the ES system in search for movies/novels, respectively, return satisfactory results. TriniT, when answering such questions is able to return tuples with an explanation of the precise relations that hold between the various entities in a tuple (either those part of the user's query, or matched through relaxation).

Seemingly simple ERQ queries which ask for a list of individual entities, not tuples, can be easily mishandled due to a lack of relation awareness. Results from ES and ERS for the query *"football players who were FIFA Player of the Year"* include the entities `DavidBeckham` and `ThierryHenry`, both of whom were runners-up for the award, but never actually received it. TriniT is able to handle this query correctly.

For this class of queries SPOX starts to suffer when the desired relationships connecting two variables are either not sufficiently populated or unavailable in the KG.

COMPLEX Queries are the most interesting: they require combining factual knowledge from multiple sources and establishing the existence of the relation specified in the query. TriniT with relaxations significantly outperforms all other systems on this dataset. Figure 3(I) helps understand why. It would be rare to find documents where keyword matching would correctly answer this query. Even then, we cannot expect to obtain a complete list of results. This query is inherently relationship-centric, best fit for TriniT. The original TriniT query contains one textual XKG predicate *"member of"*, as no KG relation covers it. However, the KG relation `graduatedFrom` lacks sufficient coverage, as the fact `(ChristopherReeve graduatedFrom CornellUniversity)` is missing. The XKG compensates for this through the textual relation *"went to"* (see Table 2), which is exploited by an automatically-invoked relaxation to facilitate the return of the relevant answer shown. For ES, this query, like most others in the benchmark, is too challenging: the evidence needed to answer is multiple hops away from a relevant entity. As in earlier examples, ERS can be brittle when establishing the existence of a relation in the query is critical for answering it. ERS scores entities in an answer based on their proximity to each other and to query terms expressing relations that must hold. This can be detrimental due to the complexities of natural language: e.g., *"private and public universities including Ivy League members, MIT, VanderbiltUniversity, SwarthmoreCollege , CalBerkeley..."* is incorrectly taken as evidence of `SwarthmoreCol-`

lege's membership in the IvyLeague. Figure 3(II) shows an example of a COMPLEX query where multiple relaxations are triggered, including a structural relaxation of the spatial locatedIn predicate, a predicate paraphrase and inverse paraphrase. On this query, SPOX fails as it contains no relation connecting a pair of entities that can serve as correct bindings of ?x and ?y, both of which have to be projected to the user. In this case, SPOX's relaxation scheme is not helpful. Here, ERS turns out to be more effective than SPOX as it relies on a less rigid scheme for answering queries.

5.5 Discussion

On relationship-centric queries, TriniT+Relax outperforms the baselines by a clear margin. This is most pronounced for the COMPLEX benchmark with relationship-centric queries – the very point that paper aims to address. Here the gains are high in all metrics.

It is also important to understand the limitations of TriniT and why it fails on some queries. Simpler entity queries can often be answered satisfactorily using traditional keyword-based approaches, as we have seen for SPOX over ESQ. Here, TriniT's relationship-centric approach is crucial only when keyword matches are misleading and crisp relations must be established. For TriniT+Relax, we observe that losses are mostly due to incorrect XKG facts and semantic drift in the relaxations. Such incorrect facts arise from incorrect entity annotations, or shortcomings in the extraction scheme. Generally, incorrect facts matching a triple pattern are less frequent than correct ones, which also means that the answers obtained through these have smaller weights. This is usually a problem for queries with a small number of expected correct answers, where the correct answers rank highest, and incorrect ones are at the bottom of the list.

The second source of incorrect answers is relaxations that drift from the original query intention. Again, this is mostly an issue in queries where few correct answers are expected compared to the number of answers actually returned, and can often be pruned away by observing a sudden drop in answer scores. This is why the scoring scheme presented in Section 4 uses *max* rather than *sum* for score aggregation.

6. RELATED WORK

Entity Search: Methods for entity search over large text corpora have been greatly advanced; [3] gives an overview. In these models, queries are keywords and return ranked lists of individual entities. Some methods use knowledge bases for feature expansion [9], but stick to the same query-and-answer model. One of the currently best methods is [2], which is based on entity language models and harnesses entity categories (i.e., semantic types) for ranking and for restricting answers to the desired type – so it can, for example, ensure that a query returns only songs, not movies, albums, or singers. However, the model is still limited to computing a list of single entities. So there is no way of returning tuples of entities, such as song-movie pairs, as answers. We included the method of [2] as a baseline in our experiments.

Query Relaxation: In IR, the classic case for query relaxation is query expansion for keyword queries [34] or recommendations for query reformulation [8]. Some work along these lines has explored the use of thesauri or knowledge bases, to generate semantically related terms for a given query. For structured data, generating relaxed queries has been explored for relational data (e.g., [26,39]) and for RDF data [11,12]. For tree-structured XML data, structural relaxation techniques have been developed, such as rewriting an XPath child condition into a descendant condition (e.g., [1]) or using semantic-relatedness-based relaxations for content terms in XPath queries (e.g., [32]). None of these is suitable for the combination of graph-structured data and text corpora.

Search on Knowledge Graphs: There is ample work on querying RDF databases and Linked-Open-Data with SPARQL [16]. However, this is exact-match querying on structured data emphasizing efficiency and scale while disregarding ranking or relaxation. Keyword-based graph search has also been extensively studied for relational databases [37]. These include ranking, but are limited to structured data and do not consider full documents attached to graph nodes. The most notable works on ranking SPARQL query answers are [11, 22], using statistical language models and supporting entity-tuple answers. Our approach is largely inspired by these models. However, this prior work has limited support for keywords attached to triples and does not extend to our XKG setting with text-based triples derived from large corpora. The same limitation holds for work on graph query languages (e.g., [36]). Finally, there are methods for telegraphic text queries over structured KGs (e.g., [28]), but they do not extend to the more demanding case of XKG.

Question Answering: Natural-language question answering (QA) is traditionally based on passage retrieval and statistical learning over unstructured text corpora. Even the ground-breaking IBM Watson system has made only limited use of structured data for special cases [19]. Recently, new approaches to QA have developed methods for translating user questions into structured queries that can be evaluated on KGs and Linked Data [7, 13, 30, 33, 35]. This line of research does not consider the combination of text and triples in an XKG, though.

Querying Entity-Annotated Text: Searching and exploring text corpora that are annotated with entities and/or linked to a KG has been addressed in various projects, most notably the Broccoli system [5,6], ERQ [23], and STICS [18]. Albeit not based on SPARQL, these are very expressive and powerful search engines. However, except for ERQ (see below) they do not provide any non-trivial ranking of results – which is crucial for XKG.

The work of [21,29] addresses telegraphic text queries over XKG-like combinations of text and data. The approach here is to jointly learn the segmentation and entity interpretation of the input query (in text form) and the ranking of candidate results. There is no notion of structured queries, though, and the more advanced queries that our model allows are beyond the scope of that prior work. Most importantly, queries that need to test for multiple relationships and return tuples of entities are not supported.

Closest to our approach is the ERQ system [23]. This work integrated text conditions into a structured query language with typed variables for both entities and entity pairs (i.e., relationships). Albeit primarily addressing richer entity-relationship search over Wikipedia, this method could be applied to XKG. We therefore include it as a baseline in our experimental studies.

7. CONCLUSION

Knowledge workers like journalists or analysts need query functionality that goes beyond the established line of entity search. This paper has addressed these needs by developing a powerful query language and a search engine, TriniT, for *relationship queries*, tapping the combination of knowledge graphs and web corpora to compute answers. For ease of use, TriniT starts with SPO-structured triple patterns that users can easily generate by a form-based UI, but allows also text phrases in each of the SPO components. Then, TriniT automatically and incrementally relaxes the query by rewriting rules for higher recall. Experiments with advanced queries over an extended knowledge graph of web-scale size demonstrate the viability of our approach and its benefits over prior work.

Acknowledgment. Qiuyue Wang is supported by the National Natural Science Foundation of China (Grant no. 61202331, 61532010).

8. REFERENCES

[1] S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava, D. Toman: Structure and Content Scoring for XML. VLDB 2005.

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

[3] K. Balog, Y. Fang, M. de Rijke, P. Serdyukov, L. Si: Expertise Retrieval. Foundations and Trends in Information Retrieval 6(2-3), 2012.

[4] K. Balog, R. Neumayer: A Test Collection for Entity Search in DBpedia. SIGIR 2013.

[5] H. Bast, A. Chitea, F. M. Suchanek, I. Weber: ESTER: Efficient Search on Text, Entities, and Relations. SIGIR 2007.

[6] H. Bast, F. Bäurle, B. Buchhold, E. Haußmann: Semantic Full-text Search with Broccoli. SIGIR 2014.

[7] J. Berant, A. Chou, R. Frostig, P. Liang: Semantic Parsing on Freebase from Question-Answer Pairs. EMNLP 2013.

[8] P. Boldi, F. Bonchi, C. Castillo, S. Vigna: Query Reformulation Mining: Models, Patterns, and Applications. Information Retrieval 14(3), 2010.

[9] J. Dalton, L. Dietz, J. Allan: Entity Query Feature Expansion Using Knowledge Base Links. SIGIR 2014.

[10] B. B. Dalvi, E. Minkov, P. P. Talukdar, W. W. Cohen: Automatic Gloss Finding for a Knowledge Base using Ontological Constraints. WSDM 2015.

[11] S. Elbassuoni, M. Ramanath, R. Schenkel, M. Sydow, G. Weikum: Language-model-based Ranking for Queries on RDF-graphs. CIKM 2009.

[12] S. Elbassuoni, M. Ramanath, G. Weikum: Query Relaxation for Entity-Relationship Search. ESWC 2011.

[13] A. Fader, L. Zettlemoyer, O. Etzioni: Open Question Answering over Curated and Extracted Knowledge Bases. KDD 2014.

[14] L. A. Galárraga, C. Teflioudi, K. Hose, F. M. Suchanek: AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases. WWW 2013.

[15] E. Gabrilovich, C. Markovitch: Computing Semantic Relatedness using Wikipedia-Based Explicit Semantic Analysis. IJCAI 2007.

[16] T. Heath, C. Bizer: Linked Data: Evolving the Web into a Global Data Space. Morgan & Claypool 2011.

[17] J. Hoffart et al.: Robust Disambiguation of Named Entities in Text. EMNLP 2011.

[18] J. Hoffart, D. Milchevski, G. Weikum: STICS: Searching with Strings, Things, and Cats. SIGIR 2014.

[19] IBM Journal of Research and Development 56(3), Special Issue on "This is Watson", 2012.

[20] I. F. Ilyas, G. Beskales, M. A. Soliman: A Survey of Top-k Query Processing Techniques in Relational Database Systems. Computing Surveys 40(4), 2008.

[21] M. Joshi, U. Sawant S. Chakrabarti: Knowledge Graph and Corpus Driven Segmentation and Answer Inference for Telegraphic Entity-seeking Queries. EMNLP 2014.

[22] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, G. Weikum: NAGA: Searching and Ranking Knowledge. ICDE 2008.

[23] X. Li, C. Li, C. Yu: Entity-Relationship Queries over Wikipedia. TIST 3(4), 2012.

[24] Mausam, M. Schmitz, S. Soderland, R. Bart, O. Etzioni: Open Language Learning for Information Extraction. EMNLP-CoNLL 2012.

[25] D. Milne, I.H. Witten: An Effective, Low-Cost Measure of Semantic Relatedness Obtained from Wikipedia Links. WIKIAI 2008.

[26] D. Mottin, A. Marascu, S. Basu Roy, G. Das, T. Palpanas, Y. Velegrakis: A Probabilistic Optimization Framework for the Empty-Answer Problem. PVLDB 6(14), 2013.

[27] Z. Nie, Y. Ma, S. Shi, J. Wen, W. Ma: Web Object Retrieval. WWW 2007.

[28] J. Pound, A. K. Hudek, I. F. Ilyas, G. E. Weddell: Interpreting Keyword Queries over Web Knowledge Bases. CIKM 2012.

[29] U. Sawant, S. Chakrabarti: Learning Joint Query Interpretation and Response Ranking. WWW 2013.

[30] S. Shekarpour, A.-C. N. Ngomo, S. Auer: Question Answering on Interlinked Data. WWW 2013.

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

[32] M. Theobald, H. Bast, D. Majumdar, R. Schenkel, G. Weikum: TopX: Efficient and Versatile Top-k Query Processing for Semistructured Data. VLDB J. 17(1), 2008.

[33] C. Unger et al.: Template-based Question Answering over RDF Data. WWW 2012.

[34] J. Xu, W. B. Croft: Query Expansion Using Local and Global Document Analysis. SIGIR 1996.

[35] M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, G. Weikum: Natural Language Questions for the Web of Data. EMNLP-CoNLL 2012.

[36] S. Yang, Y. Wu, H. Sun, X. Yan: Schemaless and Structureless Graph Querying. PVLDB 7(7), 2014.

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

[38] C. Zhai, J. Lafferty: A Study of Smoothing Methods for Language Models Applied to Information Retrieval. TOIS 22(2), 2004.

[39] X. Zhou, J. Gaugaz, W. Balke, W. Nejdl: Query Relaxation using Malleable Schemas. SIGMOD 2007.