EntityEngine: Answering Entity-Relationship Queries using Shallow Semantics

Xiaonan Li
University of Texas at Arlington
xiaonan.li@mavs.uta.edu

Chengkai Li
University of Texas at Arlington
cli@uta.edu

Cong Yu
Yahoo! Research New York
congyu@yahoo-inc.com

ABSTRACT

We introduce EntityEngine, a system for answering entity-relationship queries over text. Such queries combine SQL-like structures with IR-style keyword constraints and therefore, can be expressive and flexible in querying about entities and their relationships. EntityEngine consists of various offline and online components, including a position-based ranking model for accurate ranking of query answers and a novel entity-centric index for efficient query evaluation.

Categories and Subject Descriptors: H.3.3 Information Storage and Retrieval: Information Search and Retrieval

General Terms: Algorithms, Design, Experimentation

Keywords: entity search, entity ranking, structured entity query, Wikipedia

1. INTRODUCTION

The Web is now an information repository full of *entities*. In discovering fascinating entities, Web users are in need of structured querying facilities that explicitly deal with the entities, their properties, and relationships.

Example 1 (Motivating Example): Consider a business analyst interested in this task: Find the list of *companies* and their *founders*, where the companies are in Silicon Valley and the founders are Stanford graduates.

Two major mismatches make keyword search unsuitable for resolving such tasks. First, our tasks focus on *typed* entities such as PERSON and COMPANY and, in database terminology, their "join" relationships. Second, our tasks often require synthesizing information scattered across different places. For instance, one page may tell the analyst that Jerry Yang is a founder of Yahoo!, but whether Yahoo! is a Silicon Valley company and whether Jerry Yang is a Stanford graduate may only be found in other pages. Therefore, with only keyword search, the user has to manually assemble information from many result pages of multiple searches.

In this paper, we demonstrate EntityEngine (http://idir.uta.edu/erq), a system for answering entity-relationship query (ERQ), for solving tasks like the above one. An SQL-like declarative query mechanism, ERQ produces entities directly instead of documents. Below is the query for Example 1.

Query 1 (Entity-Relationship Query for Example 1):

```
SELECT x, y
FROM   PERSON x, COMPANY y
WHERE  x:["Stanford", "graduate"]  // Predicate p1
  AND  y:["Silicon Valley"]        // Predicate p2
  AND  x,y:["found"]               // Predicate p3
```


CIKM'10, October 26–30, 2010, Toronto, Ontario, Canada.
ACM 978-1-4503-0099-5/10/10.

We take a DB-IR integration approach in proposing entity-relationship queries. On the one hand, ERQs have explicit structured components: *typed entity variables* (e.g., x, bound to entities of type PERSON), *selection predicates* (e.g., p_1), and *relation predicates* (e.g., p_3 for the requirement that x founded y). In general, an ERQ may have an arbitrary number of variables and predicates. On the other hand, the predicates are specified by keyword-based constraints. The query semantics dictates that entities satisfy a predicate by a simple and intuitive requirement: the entities co-occur with the keywords in some contexts. A context can be a sentence, a window of text, etc. For simplicity, we use sentence as context in this demo. For example, predicate p_1 requires every x in the query answer to co-occur with "Stanford" and "graduate" in some sentence[1]. In other words, we aim to capture entity properties and relationships through shallow syntax requirements implied by users at query-time[2]. Although such syntax clue is by no means rigorous or error-proof (e.g., false evidence such as "X's partner is a Stanford graduate"), it becomes robust when we take into account the repetitive nature of the Web: true facts are more likely to be stated on many different pages.

Our approach is different from previous work in two important aspects. (1) We support complex queries with multiple selection and relation predicates. Studies on entity search [2, 3, 6] focus on single-predicate queries. The INEX and TREC entity tracks do not consider entity relationships, i.e., relation predicates. (2) The DB-based approach of structured querying of the Web [1] explicitly extracts information into databases (or similarly into RDF [5]), which will then be queried. This approach is constrained by the capability of the information extraction (IE) and natural language processing (NLP) techniques. Particularly, it requires explicit identification of the "names" of entity relationships. For example, if a "found" relation between Jerry Yang and Yahoo! was not detected during extraction phase, the information is lost and could not be queried. By contrast, ERQ, as an alternative to the DB-based approach, relies on users in forming keyword constraints. It is particularly attractive for tasks where entity properties and relationships cannot be satisfactorily extracted given the capacity of IE techniques.

2. SYSTEM ARCHITECTURE

Entity Catalog and Corpus: The 2008-07-24 snapshot of Wikipedia (about 2.5 million English articles) is used as

[1]Query expansion can be applied to increase recall, e.g., allowing a context to match "alumni" if not "graduate".

[2]The effectiveness of ERQ partially relies on users' capability in choosing proper keywords, as in IR queries.

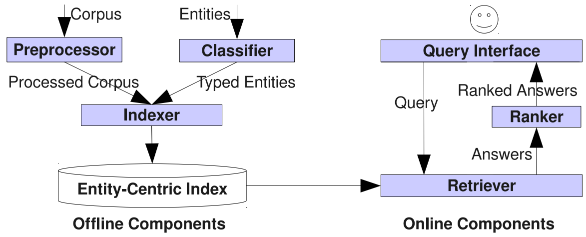

Figure 1: System Architecture

the entity catalog, i.e., each article describes one entity and its title is the entity name. The same snapshot is also the corpus. In an article, the hyperlinks to other Wikipedia articles are occurrences of the linked entities. About 0.75 million entities are assigned into ten predefined types based on simple hand-crafted rules. Nearly 100 million occurrences of these entities are collected from the corpus. We are also using Wikify [3] to detect entity occurrences in generic Web pages to enable ERQs on Web corpus.

EntityEngine is implemented upon Apache Lucene. It has several offline and online components, as shown in Figure 1.

Offline Components: For each article, the *Preprocessor* removes section titles, tables, etc., retaining only the main text, which is segmented into sentences. It then removes punctuation marks and stems all words. The *Classifier* assigns entities into predefined types based on simple rules. The *Indexer* takes preprocessed corpus and typed entities as input and constructs a novel Entity-Centric Index (ECI) [4] that records the occurrences of entities in articles. Unlike conventional document-centric inverted index in IR systems, ECI orders postings in each inverted list by entity ID instead of document ID. This design enables more efficient evaluation of ERQs than the document-centric index.

Online Components: The *Retriever* leverages ECI to retrieve entities satisfying query predicates. By the semantics of ERQ, a query can yield many false positive answers due to abundant accidental co-occurrences of entities and keywords. Therefore, in addition to ranking the answers according to co-occurrence frequency, it is also critical to tell true co-occurrence evidence apart from false evidence. The *Ranker* is thus based on a Bounded Cumulative Model (BCM) that integrates three position-based features– proximity, ordering pattern, and mutual exclusion. BCM ranks co-occurrence contexts by their likelihood of being true evidence. Comprehensive experiments on both INEX benchmark queries and our hand-crafted queries verified its superiority over state-of-the-art techniques adapted for ERQ [4]. Finally, the *Query Interface* is an Ajax-based Web client that assists users in composing queries and displays ranked answers with supporting evidence.

3. DEMONSTRATION PLAN

During the demo session, users can either form arbitrary queries or choose from 45 prepared queries, at our demo site http://idir.uta.edu/erq. Below we use Query 1 to illustrate.

Query Formulation (Figure 2):

(1) The user specifies entity variable x by selecting type PERSON from the first combo box. Variable x has one selection predicate, p_1, specified by entering keywords "Stanford" and "graduate" in the input field. The user similarly specifies variable y and predicate p_2.

(2) The user can add more selection predicates to x (and y) by clicking the "Add" button beside its predicate and a blank input field appears below it. The user can also remove additional predicates by clicking the "X" mark beside them.

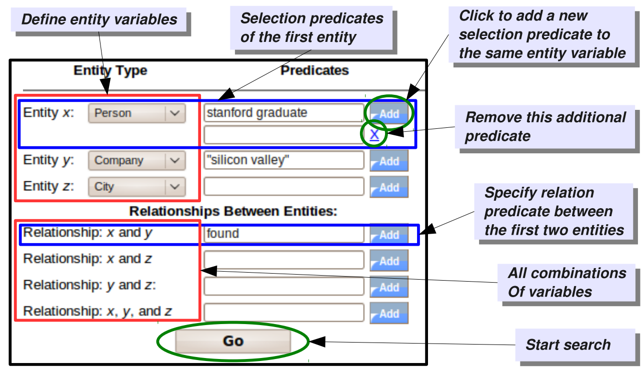

Figure 2: Query Interface

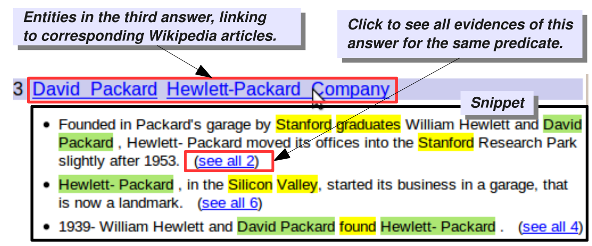

Figure 3: Result Presentation

(3) To further explore the interface, the user temporarily defines a third variable CITY z. The user observes four lines in the bottom part of the interface, corresponding to different combinations of the variables– xy, xz, yz, and xyz. Relation predicates on the combinations can be specified in the corresponding input fields. The user now removes variable z by de-selecting its type CITY. Then all the combinations containing z disappear.

(4) The user enters "found" in the input field next to combination xy, specifying the relation predicate p_3. The user clicks the "GO" button and the constructed query is sent to EntityEngine for evaluation.

Result Interpretation (Figure 3): 25 answers are returned for Query 1. (Figure 3 shows only one of them.) A snippet is provided under each answer, e.g., the black box in Figure 3. The snippet shows one context (sentence) for each predicate, serving as evidence of the answer satisfying the predicate. In the sentences, entities and query keywords are highlighted with different colors. The user can browse more evidence sentences for each predicate (if any) by clicking the "see all" link at the end of each sentence.

4. REFERENCES

[1] M. J. Cafarella, C. Ré, D. Suciu, O. Etzioni, and M. Banko. Structured querying of Web text. In *CIDR*, 2007.

[2] S. Chakrabarti, K. Puniyani, and S. Das. Optimizing scoring functions and indexes for proximity search in type-annotated corpora. In *WWW*, 2006.

[3] T. Cheng, X. Yan, and K. C.-C. Chang. Entityrank: searching entities directly and holistically. In *VLDB*, 2007.

[4] X. Li, C. LI, and C. Yu. Structured querying of annotation-rich web text with shallow semantics. Technical report, CSE Department, UT-Arlington, 2010.

[5] F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO: a core of semantic knowledge unifying WordNet and Wikipedia. In *WWW*, 2007.

[6] M. Zhou, T. Cheng, and K. C.-C. Chang. Data-oriented content query system: searching for data into text on the web. In *WSDM*, pages 121–130, 2010.

[3] `http://wdm.cs.waikato.ac.nz:8080/`