Structured Querying of Annotation-Rich Web Text with Shallow Semantics

Xiaonan Li
Department of Computer Science and Engineering
University of Texas at Arlington
xiaonan.li@mavs.uta.edu

Chengkai Li
Department of Computer Science and Engineering
University of Texas at Arlington
cli@uta.edu

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

ABSTRACT

Information discovery on the Web has so far been dominated by keyword-based document search. However, recent years have witnessed arising needs from Web users to search for named entities, e.g., finding all Silicon Valley companies. With existing search engine, users have to digest returned Web pages by themselves to find the answers. Entity search has been introduced as a solution to this problem. However, existing entity search systems are limited in their capability to address complex information needs that involve multiple entities and their inter-relationships. In this paper, we propose a novel entity-centric structured querying mechanism called Shallow Semantic Query (SSQ) to overcome this limitation. To support efficient processing of SSQ queries, we present our novel Entity Centric Index (ECI) and the Entity Centric Retrieval (ECR) algorithm based on the new index. Experiments show that for processing complex SSQ queries, ECR is significantly faster than the baseline algorithm extended from existing entity search systems. Our initial attempt of SSQ is a prototype system over Wikipedia and is accessible at **http://idir.uta.edu/ssq**.

1. INTRODUCTION

With the continuous evolution of the Web, structured data is proliferating on more and more Web pages. Such data provides us a view of the Web as a repository of "entities" (material or virtual) and their relationships. For discovering and exploring the entities that fascinate them, Web users are in need of structured querying facilities, coupled with text retrieval capabilities, that explicitly deal with the entities, their properties and relationships. In a recent report on self-assessment of the database field by a group of researchers and practitioners, it is pointed out that the database community is at a turning point in its history, partly due to the explosion of structured data on the Web. One of the major directions that database research is expanding toward is the interplay between structure and text [26]. Recently there have been extensive efforts along this general direction [12, 20, 8].

Despite the increasing popularity of structured information on the Web, the prevalent manner in which Web users access such information is still keyword-based document search. Although keyword search has been quite effective in finding specific Web pages matching the keywords, there clearly exists a mismatch between its *page-centric text-focused* view and the aforementioned *entity-centric structure-focused* view of the Web. User information needs often cannot be clearly expressed with a set of keywords, and processing the search results may require substantial user efforts.

Example 1 (Motivating Examples): Consider a business analyst investigating the development of Silicon Valley. Particularly, she is interested in the following tasks:

Task 1: Find companies located in Silicon Valley.

Task 2: Find companies and their founders, where the companies are in Silicon Valley and the founders are Stanford graduates.

There are two major mismatches making keyword queries unsuitable for resolving such tasks. First, our tasks focus on *typed* entities such as PERSON and COMPANY and their relations. Second, our tasks often involve synthesizing information scattered across different places. Hence, a simple list of articles returned by one keyword search is not sufficient. For instance, one article 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 have to be found in other articles.

While conceptually simple, with only keyword search, the tasks described above require substantial user efforts to assemble information from a potentially large number of articles. To accomplish Task 2, our analyst may start with a search on "Silicon Valley company" and scan through the potentially long list of result articles to, hopefully, fetch a list of companies that are likely to be in Silicon Valley. She then similarly issues another search on "Stanford graduate" to find a list of people graduated from Stanford University. She then manually combine entities in these two lists and, by multiple additional searches, check if a company was founded by a person, for each pair of person and company. Alternatively, she can also go through the list of companies and, for each company, find its founders and check if Stanford is their alma mater by multiple search queries. Both are painful options and require the user to break down the task into a time-consuming, error-prone iterative procedure of searching, reading and re-searching.

Query 1 (SSQ Query For Task 2):

```
SELECT p,c
FROM   PERSON AS p, COMPANY AS c
WHERE  p:["Stanford" "graduate"] AND
       c:["Silicon Valley"] AND
       (p,c):["found"]
```

Our goal is to provide a declarative query interface for such tasks and an evaluation mechanism that produces answers directly. To

VLDB '10, September 13-17, 2010, Singapore

accomplish this goal, we propose a structured querying mechanism called *Shallow Semantic Query* (SSQ). Query 1 illustrates the SSQ query for Task 2. The query syntax is modeled after SQL, allowing information needs to be specified in a structured manner instead of only a set of keywords. There are three elementary concepts within this SSQ query. First, the query centers on two *entity variables*, p and c. Variable p is bound to all entities belonging to type PERSON and c to all entities belonging to type COMPANY. Second, for each variable, the query specifies a *selection predicate* as the criterion on the selection of entities. For example, a desired person p should be a Stanford graduate (p : ["Stanford" "graduate"]). Third, a *relation predicate* specifies the relation between p and c (c was founded by p). To illustrate the concepts of SSQ and help issue SSQ queries, we developed a prototype (**http://idir.uta.edu/ssq**) based on techniques introduced in this paper.

Challenges: Developing SSQ brings forward several research challenges. *First*, the notion of Shallow Semantic Query and the semantics of query results must be clearly defined. Ideally, it should meet two requirements. On one hand, it should be flexible enough to express complex inter-relationships among entities. On the other, query specification should be as user-friendly as possible. *Second*, as a search system involving user interaction, an efficient query processing algorithm is needed. This is particularly challenging for SSQ since SSQ queries are structured and may potentially involve multiple entities and their inter-relationships. *Third*, an effective ranking mechanism has to be established. Ranking models that are typical in document retrieval systems (e.g., PageRank and Vector Space Model) do not directly apply to ranking SSQ search results. In this paper, we focus on the first two challenges and refer readers to our technical report [23] for a full discussion on all three topics.

Contributions: This paper makes the following contributions:

- We introduce Shallow Semantic Query, an entity-centric structured mechanism for querying entities and their relationships, and formalize its semantics (Section 3).
- We formalize the problem of Entity Retrieval for SSQ (Section 3.1) which is a generalization of the entity search tasks in existing systems. Our definition of Entity Retrieval can be used to support a wide class of entity ranking techniques.
- We design a novel Entity Centric Index and propose Entity Centric Retrieval algorithm for efficiently processing SSQ queries in conformity to our Entity Retrieval definition (Section 4).
- We implemented a prototype SSQ system (http://idir.uta.edu/ssq) and conducted comprehensive experiments to demonstrate the efficiency of our query processing algorithm (Section 5).

2. RELATED WORK

Shallow Semantic Query is not the only approach to enable entity-centric queries. A large body of research works from different areas have been published towards the general goal. This section provides a review of some of the most important related works and clarify their differences from SSQ.

The **DB-oriented** approach explicitly extracts entity-relation tuples from text into relational databases. SQL queries and keyword queries can thus be issued over the populated database [7, 4, 14, 19, 9, 10, 21, 16]. Systems taking this approach heavily rely on the extraction capability. Facts that are not pre-extracted cannot be queried. Although Natual Language Processing (NLP) promises the ideal extraction precision and recall [19, 9, 10, 6], the huge computational cost of current NLP techniques render themselves unsuitable for processing large corpus and thus encouraged a cheaper alternative, Information Extraction (IE). Typical IE techniques [24, 15, 18] define or learn application-specific extraction patterns, thus could not be applied for arbitrary relations. Open IE [6, 19] breaks the limitation by exploiting Part-of-Speech patterns, in particular, extracting verb-connected facts. However, many relations are often stated without verbs. For example, it is much more common to see the expression *US President Bill Clinton* than *Bill Clinton is a US President*.

Semantic Web approach [27, 22, 17, 5] explicitly encodes entities and their relations (and knowledges in general) in RDF [1], the W3C recommendation of data model for Semantic Web. It exploits the full featured structured query language, SPARQL [1] to support sophisticated entity-relationship queries, coupled with reasoning power. Despite its tempting promise, the building blocks of Semantic Web, RDF data must be collected beforehand. Hence, it suffers similar problem to DB-oriented approach. Some systems reliably extract RDF from structured/semi-structured semantic data sources [5, 27], like Infoboxes in Wikipedia and WordNet. However, such data sources are still quite limited in scope. Others apply IE techniques over Web pages to bootstrap RDF extraction [17], but the quality control is much more difficult. The requirement on explicit semantics in Semantic Web rules out existing Open IE techniques, since Open IE only exploits semantic independent language patterns. Besides, independently developed Semantic Webs suffers the issue of interoperability. [5]

The **IR-oriented** approach, exemplified by the recently formed entity search and ranking problems in the IR community [11, 25, 2, 3, 29, 28], focuses on retrieving named entities relevant to certain contextual constraints from free text. The problem is often presented as a natural language description of the preferred entities plus a type constraint on the entities. To rank the answers, typical IR techniques like TF-IDF [28, 2], HITS [28] and PageRank [13] are commonly applied with adaptation. However, there is a surprising lack of studies on how to efficiently search entities. A slight variation of full text index has been used and loosely discussed in some existing work [11, 13]. Not until very recently, a careful study on efficiency problem [30] is proposed, which selectively builds additional posting lists as a trade-off between space and efficiency. However, all existing systems focus searching entities by single contextual constraint, which limits their capability in addressing complex user information needs. Although Content Query Language [30] brings more flavors into the query syntax, like type definition, context pattern and query-customizable ranking, it is inherently limited to single contextual constraint per query.

Shallow Semantic Query uniquely takes the DB+IR integrated approach in pursing entity-centric tasks. On one hand, SSQ queries have explicit structured components (typed entity variables, selection/relation predicates), offering greater expressiveness than pure keyword queries and other entity query language; on the other, each individual predicate is specified by keyword-based constraints, avoiding the strong requirements of explicit schema (as in database) and semantic (as in Semantic Web). The SSQ system finds entities satisfying predicates by a simple and intuitive requirement - entities should co-occur with keywords in predicates in some contexts (e.g., a sentence). For example, predicate p : ["Stanford" "graduate"] requires every entity appearing in the query answer co-occurs with keyword "Stanford" and "graduate". In other words, SSQ aims to capture entity properties and relationships through shallow syntax requirements implied by user defined predicates, rather than pre-extracting them at system construction time.

In summary, SSQ is unique in its ability to answer complex structured query over textual corpus. Although currently experimented with Wikipedia , it can be extended to other corpus with assistance of entity identification technology.

3. SHALLOW SEMANTIC QUERY

In this section, we formally introduce the concept of Shallow Semantic Query (SSQ). An SSQ query consists of *entity variables* and *predicates*. Entity variables (e.g.,, p in Query 1) are bound to typed entities and are associated with keyword constraints to form query *predicates* (e.g.,, p : ["Stanford" "graduate"]), which express the semantic criteria in selecting and relating entities. Formally:

Definition 1 (Shallow Semantic Query): Given a set of entity types $\mathbb{D}$, a shallow semantic query is a quadruple $\langle V, D, P, U\rangle$:

– V is a set of entity variables $\{v_1, \ldots, v_n\}$.

– D is a multi-set of entity types $\{d_1, \ldots, d_n\}$, where d_i is the type for the corresponding $v_i \in V$. Two variables can be of the same type, i.e.,, d_i=d_j, thus D is a multi-set.

– P is a set of conjunctive predicates. Each $p \in P$ is a pair $\langle V_p, C_p\rangle$, where $V_p \subseteq V$ and C_p is a keyword-based constraint associated with V_p. The constraint C_p is a set of phrases, where each phrase is made up of one or more keywords. The predicate p is a *selection predicate* if $|V_p|$=1 and *relation predicate* otherwise.

– $U \subseteq V$ is the set of variables constituting the output tuple.

Example 2: By the above definition, Query 1 can be formulated as q=$\langle V, D, P, U\rangle$, V=U=$\{v_1, v_2\}$, D=$\{$PERSON, COMPANY$\}$, $P = \{p_1, p_2, p_3\}$, where $p_1 = \langle\{v_1\}, \{$"Stanford","graduate"$\}\rangle$, $p_2 = \langle\{v_2\}, \{$"Silicon Valley"$\}\rangle$, and p_3=$\langle\{v_1, v_2\}, \{$"found"$\}\rangle$. p_1 and p_2 are selection predicates; p_3 is a relation predicate.

Note that U is a subset of V, resembling the notion of projection in relational algebra. For example, suppose $\langle$Jerry Yang, Yahoo!$\rangle$ and $\langle$David Filo, Yahoo!$\rangle$ are both answers to Query 1. If c is the only output variable, only one $\langle$Yahoo!$\rangle$ will be in the output. Without loss of generality, we assume U=V throughout this paper. Hence for short, an SSQ query can be written as q=$\langle V, D, P\rangle$.

We use an SQL-like syntax to express SSQ queries (Query 1), where the SELECT, FROM and WHERE clauses specify output variables, entity types and predicates, respectively. Due to space limitations, we omit the formal definition of query syntax and explain the queries in plain English when needed.

As noted before, SSQ is intended to work on textual data, therefore, it only recognizes information that is explicitly stated in the text and retrieves entities co-stated with predicate phrases within certain contexts. In other words, SSQ system searches for query answers supported by textual evidences.

Definition 2 (SSQ Answer Tuple): Given a query q=$\langle V, D, P\rangle$, an answer tuple t is defined as follows:

– t=$\langle e_1, e_2, \ldots, e_{|V|}\rangle$ is a tuple of entities, where each e_i is an entity instantiated from variable $v_i \in V$ and has v_i's type $d_i \in D$. [1]

– $\forall p$=$\langle V_p$,$C_p\rangle \in P$, an *evidence* of t with regard to p is a textual context containing (1) all the entities in $t.V_p$ (the projection of t containing only entities instantiated from variables in V_p) and (2) all the phrases in C_p. The context is a unit piece of textual content.

– t is an SSQ answer to q if and only if, for each $p \in P$, there exists at least one evidence of t with regard to p.

The unit of textual context for evidences can be sentences, paragraphs, documents, windows of words, etc. Different contexts have different retrieval capabilities and thus result in different answer tuples. While a comparative study of different contexts is left for future investigation, this paper choose *sentence* as the unit of context for the following reasons: (1) it is the logical unit that people convey information and most facts are embedded in sentences; (2) it is one of the simplest textual context to implement. In the following discussion, we use "evidence" and "sentence" interchangeably unless a distinction is needed.

[1]We discuss how to assign types to entities in Appendix C.

3.1 Entity Retrieval

An evidence of tuple t with regard to predicate $p = \langle V_p, C_p\rangle$ is represented as a quadruple $\langle d, s, \widehat{t.V_p}, \widehat{C_p}\rangle$, where:

– d and s are document ID and sentence number. They uniquely identify sentence s of document d in the whole corpus.

– Each $\hat{e}_i \in \widehat{t.V_p}$ is a triple $\hat{e}_i$=$\langle e_i, f_i, l_i\rangle$. e_i is the binding entity of variable $v_i \in t.V_p$; f_i and l_i encode the span of e_i, i.e., the position of the first and last words representing the entity;

– Each $\hat{c}_i \in \widehat{C_p}$ is the position of keyword constraint $c_i \in C_p$. If c_i is a phrase, $\hat{c}_i$ is the position of the first term of c_i.

As an example, if the sentence 8 of document 9 reads,

Jerry Yang graduates from Stanford University ...

SSQ should retrieve it as an evidence of Jerry Yang with regard to p_1 of Example 2. The evidence, $\langle 9, 8, \{\langle 6, 0, 1\rangle\}, \langle 4, 2\rangle\rangle$, means that in sentence 8 of document 9, entity 6 (Jerry Yang) co-occurs with "Stanford" and "graduate". The entity name spans from position 0 to 1, "Stanford" at position 4 and "graduate" at 2.

Given an answer tuple t, denote $\phi(t.V_p)$ as all evidences of t with regard to predicate p. The task of entity retrieval is to retrieve all evidences of every answer tuple with regard to every predicate.

Definition 3 (Entity Retrieval): Denote $A(q)$ as all answer tuples of query q=$\langle V, D, P\rangle$. The task of entity retrieval is to evaluate

$$\Phi(q) = \cup_{p\in P}\phi(p) = \cup_{p\in P}\{\phi(t.V_p)|t \in A(q)\}$$

Note that each element in $\Phi(q)$ or $\phi(p)$ is a group of evidences, $\phi(t.V_p)$. Such a group contains all evidences of $t.V_p$ with regard to p. We refer $t.V_p$ as *signature* of the group $\phi(t.V_p)$.

Our evidence representation encodes typical information required by several state-of-the-art entity ranking methods [13, 25, 11]. Actually, it can be used to support a wide class of entity ranking methods that employ positional information of keywords and entities. We identify all such ranking functions as *Position-based Entity Ranking*. As part of SSQ concept, we have also defined a ranking framework for Position-based Entity Ranking. But due to space limit, we omit full discussion in this respect. More details can be found in Appendix A and [23].

4. ANSWERING SSQ QUERIES

In this section, we focus on how to process SSQ queries, i.e. how to evaluate $\Phi(q)$ (Defintion 3) for arbitrary query q.

Given an SSQ query q=$\langle V, D, P\rangle$, if each predicate $p \in P$ is treated as a single predicate query, we can decompose entity retrieval on q into a series of independent entity retrieval on single predicate query p, plus additional processing to integrate their results. By Definition 3, independent entity retrieval on p is to find $\Phi(p) = \{\phi(t)|t \in A(p)\}$. It can be shown that $\phi(p) \subseteq \Phi(p)$.

If a system can independently process any predicate p, i.e., evaluate $\Phi(p)$, then any single predicate query can be trivially evaluated as $\Phi(q)$=$\Phi(p)$. In high level, current entity search systems can be viewed as single predicate SSQ query system.

We now discuss how to integrate all $\Phi(p)$'s into $\Phi(q)$, where q has multiple-predicates.

Table 1 shows an example case after independent evaluation of all three predicates in Example 2. We use the signatures (i.e., entity tuples) to represent evidence groups in each $\Phi(p_i)$. $\Phi(p_1)$ has five signatures, a1 to a5; $\Phi(p_2)$ has three, b1 to b3; $\Phi(p_3)$, c1 to c5.

Let's first examine v_1. Dick Price is a Stanford graduate (a5) but he does not found any company (no signature in $\Phi(p_3)$ has him). Hence, a5 should not be part of any answer tuple. a5 is ruled out. Steve Jobs founded Apple (c1) but he is not a Stanford graduate

Table 1: Example Signatures in $\Phi(p_1), \Phi(p_2), \Phi(p_3)$

p_1	v_1	p_2	v_2	p_3	v_1	v_2
a1	Jerry Yang	b1	eBay	c1	Steve Jobs	Apple
a2	Larry Page	b2	IKEA	c2	Jerry Yang	Yahoo!
a3	Bill Gates	b3	Yahoo!	c3	Larry Page	Google
a4	David Filo	b4	Apple	c4	David Filo	Yahoo!
a5	Dick Price			c5	Bill Gates	IKEA
Joinable Signatures: (a1 b3 c2), (a3 b2 c5), (a4 b3 c4)						

(not in $\Phi(p_1)$). Hence, c1 is removed too. All other persons appear in both $\Phi(p_1)$ and $\Phi(p_3)$.

Now we switch to v_2. b1 is ruled out because eBay does not appear in $\Phi(p_3)$; c3 is ruled out because Google is not in $\Phi(p_2)$. As a result, Larry Page is longer in $\Phi(p_3)$, and consequently, a2 is ruled out. Sine c1 has been ruled out when we were examining v_1, Apple is no longer in $\Phi(p_3)$ and hence b4 should also be removed.

Eventually, after we have examined both variables, $\Phi(p_1)$ has a1, a3, a4 left; $\Phi(p_2)$ has b2, b3 and $\Phi(p_3)$ has c2, c4, c5. (a1 a3 c2) can be joined to form an answer tuple, t_1=⟨Jerry Yang,Yahoo!⟩, (a3 b2 c5) to another tuple t_2=⟨Bill Gats,IKEA⟩ and (a4 b3 c4) the third one t_3=⟨David Filo,Yahoo!⟩. By Definition 3, $\phi(p_1)$={a1,a3,a4}, $\phi(p_2)$={b2,b3}, $\phi(p_3)$={c2,c4,c5} and $\Phi(q)$=$\phi(p_1)\cup\phi(p_2)\cup\phi(p_3)$.

The above procedure can be formalized as a series of joins on each variable followed by a series of projections on each predicate. First, $\Phi_{1,3}$=$\Phi(p_1) \bowtie^{v_1} \Phi(p_3)$; then Φ=$\Phi_{1,3} \bowtie^{v_2} \Phi(p_2)$. To obtain $\Phi(q)$, Φ is projected on v_1 of p_1, v_2 of p_2 and $\langle v_1, v_2\rangle$ of p_3 respectively to get $\phi(p_i)$'s, which are then unioned together.

Generally, if a predicate subset P'=$\{p_{k_i}|i$=$1..n\}\subseteq P$ have common variables $V_{P'}\subseteq\bigcap_{p\in P'} V_p$, their $\Phi(p)$'s shall be joined on $V_{P'}$,

$$\bowtie^{V_{P'}}_{p\in P'} \Phi(p) = \Phi(p_{k_1}) \bowtie^{V_{P'}} \ldots \bowtie^{V_{P'}} \Phi(p_{k_n})$$

where $V_{P'}$ is the join attribute(s) and $\Phi(p)$'s are join inputs. A similar shortcut syntax will be used in our algorithms. For each $P'\subseteq P$, whose $V_{P'}\neq\emptyset$ and $\nexists P''\subset P', V_{P''}=V_{P'}$, the joins on $V_{P'}$ shall be performed. For brevity, we call the whole join procedure involving all such P' as *graph join* (on V), denoted as $\otimes^V_{p\in P}\Phi(p)$, which shall then be projected (π) on V_p of each p, to obtain $\phi(p)$ and eventually, $\Phi(q)$. We denote the whole processing of independently evaluated $\Phi(p)$'s as $\pi_P \otimes^V_{p\in P} \Phi(p)$.

Theorem 1: An SSQ query q=$\langle V, D, P\rangle$ can be evaluated in three phrases. Phrase 1, evaluate each $p \in P$ independently. Phrase 2, graph join all $\Phi(p)$'s on V. Phase 3, project the result of graph join on individual predicates and union them. In short,

$$\Phi(q) = \pi_P \otimes^V_{p\in P} \Phi(p)$$

The rest of this section study physical evaluation of $\Phi(q)$. As a baseline, Section 4.1 reviews entity retrieval algorithm using Document Centric Index (DCI). Section 4.2 introduces our novel Entity Centric Index (ECI) as an alternative index. Based on ECI, Section 4.3 proposes Entity Centric Retrieval algorithm for efficiently processing SSQ queries.

4.1 Baseline: Document Centric Retrieval

Document Centric Index (DCI) (or slight different versions) is used by existing entity search systems [11, 13]. It is a variant of full text index. As Figure 1 shows, DCI consists of two kinds of posting lists, term-document posting list (TDPL) and document-entity posting list (DEPL).

A TDPL is created for each unique term in corpus, listing all documents where it appears in ascending order of document ID. Each

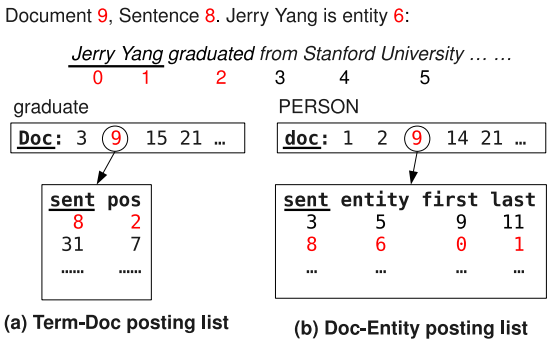


Figure 1: Document Centric Index

document in a TDPL is associated with a list of entries recording exact term locations in that document. Each entry has two attributes, $sent$ (sentence where the term occurs) and pos (position of term within the sentence). In Figure 1(a), term "graduate" appears in documents 3, 9, 15, 21 and so on. In document 9, it can be located as the second term (position 2) of sentence 8 and the seventh (position 7) term of sentence 31. As can be seen, TDPL is almost the same as the posting list used in tradition full text index, except that the "position" attribute in traditional full text index becomes $\langle sent, pos\rangle$ in TDPL.

DEPL is structurally similar to TDPL. A DEPL is created for every entity type to be supported. It lists all documents containing entities of that type in ascending order of document ID. Each document in DEPL is associated with a list of entries recording occurrences of entities. In Figure 1(b), documents 1, 2, 9,... contain PERSONs. In document 9, entity 6 (Jerry Yang) appears in sentence 8, spanning from position 0 to position 1.

DCI follows the $term \rightarrow doc \rightarrow entity$ information flow in the three-dimension space of $\{term, doc, entity\}$. TDPL bridges term to document; while DEPL bridges document to entity.

With the posting lists in Figure 1, we can easily find all sentences where a PERSON co-occurs with "graduate" as follows. We scan the two document lists with a merge join (major-join) on doc. We first find that doc 9 is joinable, as it appears in both. So, we temporarily pause the major-join and starts another merge join (sub-join) of the two entry lists associated with doc 9. The sub-join is on attribute $sent$, during which sentence 8 is firstly joined and the corresponding positions and entity ID are retrieved. Thus, we retrieved one co-occurrence of a PERSON (Jerry Yang) with "graduate" in document 9 sentence 8. The sub-join continues until either entry list is exhausted, at which time the major-join resumes and proceeds to the next joinable document, doc 21. When the major-join completes, we would have retrieved all sentences where a PERSON co-occurs with "graduate", together with all the positions.

In general, given any predicate p, $\Phi(p)$ can be evaluated by merge joining all posting lists of $|V_p| \cup |C_p|$ on $\langle doc, sent\rangle$. By Theorem 1, an arbitrary SSQ query can thus be answered by Document Centric Retrieval (DCR) algorithm (Algorithm 1).

Algorithm 1 follows exactly the 3-phrase processing depicted in Theorem 1. As we noted before, existing entity search systems use DCI to handle a special class of SSQ queries, single predicate query. Their processing algorithms are essentially one iteration of the outer-loop in DCR algorithm.

DCR algorithm is potentially inefficient as it retrieves $\Phi(p)$ rather than $\phi(p)$ (Definition 3). Processing power is potentially wasted on retrieving evidences belonging to $\Phi(p) - \phi(p)$.

Example 3: Consider a query Q with two selection predicates, $p_1 = \langle\{v\}, \{$"Stanford","graduate"$\}\rangle$ and p_2=$\langle\{v\}, \{$"Russian"$\}\rangle$,

Algorithm 1: Document Centric Retrieval

Input: Query $q = \langle V, D, P \rangle$
Output: $\Phi(q)$

1 **foreach** $p = \langle V_p, C_p \rangle \in P$ **do**
2 $X \leftarrow V_p \cup C_p$;
3 $R(p) \leftarrow \emptyset$;
4 $x^1 \leftarrow$ documents in posting list of $x, \forall x \in X$;
5 **foreach** $r^1 \in R^1(p) = \bowtie^{doc}_{x \in X} x^1$ **do**
6 $x^2 \leftarrow$ entries in x associated with document $r^1, \forall x \in X$;
7 $R^2(p) \leftarrow \bowtie^{sent}_{x \in X} x^2$;
8 $R(p) \leftarrow R(p) \cup (\{r^1\} \times R^2(p))$
9 $\Phi(p) \leftarrow$sort and group $R(p)$ by V_p;
10 $\Phi(q) \leftarrow \pi_P \otimes^V_{p \in P} \Phi(p)$; // Theorem 1

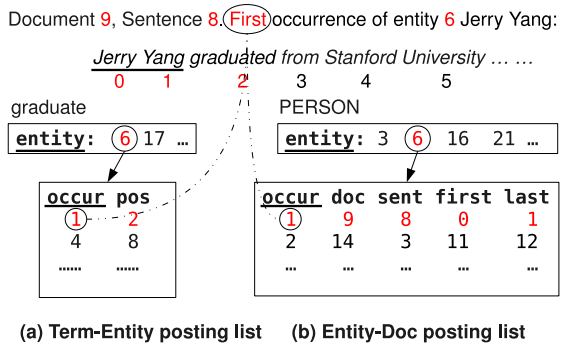


Figure 2: Entity Centric Index

where v is a PERSON. Suppose 100 persons satisfies p_1 with 1,000 evidences (10 evidences per person) and 1,000 persons satisfies p_2 with 10,000 evidences (10 per person). A total of 11,000 evidences are retrieved. However, if 10 persons actually satisfy both predicates, only 200 evidences survive the graph join in phrase 2 (10 per person per predicate). Other evidences (10,800 in total) are trash evidences to be discarded, a huge wast of processing power.

In summary, ordering posting list entries by $\langle doc, sent \rangle$ makes DCI a convenient structure to retrieve evidences for arbitrary SSQ predicate. However, for multi-predicate queries, independent predicate evaluation may waste processing power on retrieving large quantities of trash evidences. It is unknown how to (and probably not able to) prune trash evidences using the basic DCI. Section 4.3 will show how Entity Centric Retrieval breaks this limitation.

4.2 Retrieval with Entity Centric Index

To overcome DCI's drawback, we present Entity Centric Index (Figure 2), a novel index organization of the $\{term, doc, entity\}$ three-dimension space. ECI has the same number of posting lists as DCI, with one for each term and one for each type. However, these posting lists are ordered by entity ID rather than document ID.

A term-entity posting list (TEPL) for term w enlists, in ascending order, all entities co-occurring with w in some sentence. Each such entity is associated with a list of entries recording co-occurrence information with two attributes, $occur$ (entity occurrence identifier) and pos (w's position). In Figure 2(a), "graduate" co-occurs with entity 6, 17, etc. It occurs at position 2 of the sentence where entity 6, Jerry Yang, appears for the first time and position 8 of his 4th occurring sentence. Here, we assume that an entity's all N occurrences throughout the corpus are numbered 1 to N by certain order, say the natural order of $\langle doc, sent \rangle$ pairs of the occurrences.

An entity-document posting list (EDPL) for type T enlists all entities of type T in ascending order and associates a complete list of occurrence information with each entity. In Figure 2(b), entity 3, 6 and 16 all belong to PERSON. The first occurrence of entity 6 is in document 9 sentence 8, spanning from position 0 to 1.

ECI follows the $term \rightarrow entity \rightarrow doc$ information flow, with TEPL bridging the first arrow and EDPL bridging the second. By merge joining the two posting lists in Figure 2 on $\langle entity, occur \rangle$ (in a similar fashion as the major/sub-join on $\langle doc, sent \rangle$ in DCI), we can retrieve the evidence that entity 6 co-occurs with "graduate" in document 9 sentence 8.

In general, a selection predicate can be evaluated with ECI as conveniently as with DCI, by merge joining muliple TEPLs with one EDPL on $\langle entity, occur \rangle$. Since posting lists are primarily ordered by entity ID, the resulting evidences are naturally ordered by entity ID and can be grouped by V_p to form $\Phi(p)$ effortlessly.

However, evaluating relation predicate is quite different because it does not require all posting lists to be joined on $entity$. In p_3 of Example 2, the EDPLs of PERSON and COMPANY have completely distinct set of entities. p_3 requires a PERSON and a COMPANY to appear in the same sentence, i.e., they must be joined on $\langle doc, sent \rangle$, which are only subsidiary attributes in EDPL. Naively, a costly nested-loop join (on $\langle doc, sent \rangle$) of the two posting lists can solve the problem, which access both posting lists entirely. But with the presence of relation keyword "found", we may do better.

We split p_3 into two selection predicates, $p_3^{v_1}$=$\langle \{v_1\}, \{\text{"found"}\} \rangle$ and $p_3^{v_2}$=$\langle \{v_2\}, \{\text{"found"}\} \rangle$, which are evaluated independently to retrieve all their evidences, denoted as $R(p_3^{v_1})$ and $R(p_3^{v_1})$. The two sets are then joined on $\langle doc, sent \rangle$ to form evidences for p_3. To illustrate, suppose sentence 7 of document 10 reads

Jerry Yang (entity 6) *co founded Yahoo!* (entity 17) *in 1995...*

$R(p_3^{v_1})$ will contain evidence $\langle 10, 7, \{\langle 6, 0, 1 \rangle\}, \langle 3 \rangle \rangle$ and $R(p_3^{v_2})$ will contain $\langle 10, 7, \{\langle 17, 4, 4 \rangle\}, \langle 3 \rangle \rangle$. The two will be joined to form an evidence of p_3, $\langle 10, 7, \{\langle 6, 0, 1 \rangle \langle 17, 4, 4 \rangle\}, \langle 3 \rangle \rangle$.

We refer to this processing technique as *relation splitting*. It is potentially less costly than the naive nested-loop join, as it only retrieves evidences for entities in PERSON$\cap found$ and entities in COMPANY$\cap found$.

Generally, a relation predicate p=$\langle V_p, C_p \rangle$ can be split into $|V_p|$ selection predicates, $S(p)$=$\{p^v | v \in V_p\}$, where p^v=$\langle \{v\}, C_p \rangle$ is a *split predicate* of p. The evidences of p, $R(p)$, can be evaluated as $R(p)$=$\bowtie^{\langle doc, sent \rangle}_{v \in V_p} R(p^v)$, where $R(p^v)$ is the evidence set for p^v. $R(p)$ is then sorted and grouped by V_p to get $\Phi(p)$. We trivially consider a selection predicate as a split predicate of itself.

Thus, we have known how to evaluate any single predicate. By Theorem 1, we can evaluate arbitrary SSQ queries with Basic Entity Centric Retrieval (bECR) algorithm (Algorithm 2). bECR also evaluates predicates independently (the outer-loop over P, line 1). Therefore, it retrieves the same evidences as DCR does, including trash evidences (in case of a multi-predicate query).

Consider again Example 3 in Section 4.1. For p_1, suppose there are 200 persons appearing in both "Stanford"'s and "graduate"'s posting lists, i.e., $|R^1(p_1^v)|$=200 (line 7). Hence for these 200 persons, the inner loop is executed. It will find, according to the example setting, 100 persons satisfying p_1 with 1,000 evidences. Also for p_2, 1,000 persons will be retrieved with 10,000 evidences. In sum, 1,200 persons need to execute the inner loop, and a total of 11,000 evidences will be retrieved.

However, with ECI, a subset of trash evidences can be prevented from being retrieved.

Algorithm 2: Basic Entity Centric Retrieval

Input: Query $q = \langle V, D, P\rangle$
Output: $\Phi(q)$

1 **foreach** $p = \langle V_p, C_p\rangle \in P$ **do**
2 $S(p) \leftarrow \{p^v | v \in V_p\}$; `// Relation splitting`
3 $R(p^v) \leftarrow \emptyset, \forall p^v \in S(p)$;
4 **foreach** $p^v \in S(p)$ **do**
5 $X \leftarrow \{v\} \cup C_p$;
6 $x^1 \leftarrow$ entities in posting list of $x, \forall x \in X$;
7 **foreach** $r^1 \in R^1(p^v) = \bowtie^{entity}_{x \in X} x^1$ **do**
8 $x^2 \leftarrow$ entries in x associated with entity $r^1, \forall x \in X$;
9 $R^2(p^v) \leftarrow \bowtie^{occur}_{x \in X} x^2$;
10 $R(p^v) \leftarrow R(p^v) \cup (\{r^1\} \times R^2(p^v))$;
11 **if** $|V_p| > 1$ **then**
12 $R(p) \leftarrow \bowtie^{\langle doc, sent\rangle}_{v \in V_p} R(p^v)$;
13 $R(p) \leftarrow$sort $R(p)$ by V_p;
14 **else**
15 $R(p) \leftarrow R(p^v)$; `// naturally sorted by` V_p
16 $\Phi(p) \leftarrow$group $R(p)$ by V_p;
17 $\Phi(q) \leftarrow \pi_P \otimes^V_{p \in P} \Phi(p)$; `// Theorem 1`

Algorithm 3: Entity Centric Retrieval

Input: Query $q = \langle V, D, P\rangle$
Output: $\Phi(q)$

1 $S(p) \leftarrow \{p^v | v \in V_p\}, \forall p \in P$; `// Relation splitting`
2 $S \leftarrow \bigcup_{p \in P} S(p)$;
3 $S(v) \leftarrow \{p^v | p^v \in S\}, \forall v \in V$; `// Group` S `by` v
4 $R(p^v) \leftarrow \emptyset, \forall p^v \in S$; `// Store evidences for` p^v
5 **foreach** $v \in V$ **do**
6 $X \leftarrow \{v\} \cup \bigcup_{p^v \in S(v)} C_p$;
7 $x^1 \leftarrow$entities in posting list of $x, \forall x \in X$;
8 **foreach** $r^1 \in R^1(v) = \bowtie^{entity}_{x \in X} x^1$ **do**
9 $x^2 \leftarrow$entries in x associated entity $r^1, \forall x \in X$;
10 **foreach** $p^v \in S(v)$ **do**
11 $R^2(p^v) \leftarrow \bowtie^{occur}_{x \in \{v\} \cup C_p} x^2$;
12 $R(p^v) \leftarrow R(p^v) \cup (\{r^1\} \times R^2(p^v))$
13 **foreach** $p \in P$ **do**
14 **if** $|V_p| > 1$ **then** `// Relation predicate`
15 $R(p) \leftarrow \bowtie^{\langle doc, sent\rangle}_{v \in V_p} R(p^v)$;
16 $R(p) \leftarrow$ sort $R(p)$ by V_p;
17 **else**
18 $R(p) \leftarrow R(p^v)$ `// Selection predicate`
19 $\Phi'(p) \leftarrow$group $R(p)$ by V_p;
20 $\Phi(q) \leftarrow \pi_P \otimes^V_{p \in P} \Phi'(p)$

4.3 Entity Centric Retrieval with Pruning

Let's first re-examine Example 3. p_1 requires a PERSON to co-occur with "Stanford" and "graduate"; p_2 requires the same person to co-occur with "Russian". If a person does not co-occur with all the three keywords, it is guaranteed not an answer to query Q. Suppose for p_1, among the $|R^1(p_1^v)|$=200 persons appearing in both "Stanford"'s and "graduate"'s posting lists, 30 persons also appear in "Russian"'s. Then, only for these 30 persons, evidences for p_1 and p_2 need to be retrieved. Following the example setting, around 600 evidences will be retrieved (10 per person per predicate), a huge cut-down from 11,000 evidences.

Ordering posting lists by entity ID provides ECI an opportunity to accomplish such pruning capability. Based on the intuition described above we propose Entity Centric Retrieval (ECR) algorithm (Algorithm 3). ECR does not evaluate predicates independently, instead it applies relation split to all predicates and processes split predicates sharing the same variable in batch.

In ECR, the loop over V (line 5), processes split predicates batch by batch. For $S(v)$, EDPL of v's type is merge joined (on $entity$) with all TEPLs from all split predicates in $S(v)$ (line 8). For each entity r^1 returned by this join, the inner loop (line 10) retrieves evidences for each split predicates $p^v \in S(v)$ respectively. Recall our discussion on Example 3 at the beginning of this subsection. $|R^1(v)|$=30. For each entity in $R^1(v)$, line 11 retrieves 10 evidences for p_1^v=p_1 and 10 for p_2^v=p_2. Hence, $|R(p_1^v)|$=$|R(p_2^v)|$=300.

Then, during the loop over P (line 13), evidences for split predicates of the same relation predicate p are joined on $\langle doc, sent\rangle$ to produce evidences for p (line 15). For each $p \in P$, its evidences, $R(p)$, are grouped by V_p (line 18), producing $\Phi'(p)$. It is important to note that $\phi(p) \subseteq \Phi'(p) \subseteq \Phi(p)$, due to potential pruning of trash evidences. An analytical study on the pruning capability of ECR algorithm over bECR algorithm is given in Appendix B.

5. EMPIRICAL RESULTS

To demonstrate the effectiveness of SSQ and evaluate the efficiency of ECR algorithm We developed a prototype SSQ system (**http://idir.uta.edu/ssq**) over a 2008 snapshot of Wikipedia [2]. The Wikipedia articles (2.4 millions) serve as both the text corpus for finding query answers and the repository of named entities. We manually define ten entity types and assigned 0.75 million articles into these types using simple rules. These articles become our entity set[3]. Further, we leverage *internal links* in Wikipedia articles as entity occurrences, collecting nearly 100 millions annotations of entities in our entity set. For more detail about our data set, please see Appendix C.

This section reports empirical comparison between ECR and DCR based on our prototype. We use the *de facto* standard, count of disk block I/O, as the measure of query processing cost. Basically, for each test query, we compare the disk block reads incurred by both algorithms. The block size is 1 KB.

5.1 Efficiency of DCR and ECR

To understand cost of processing complex SSQ queries, our query set (Appendix D) is systematically designed in groups with growing complexity (number of predicates). Each query is labeled as an x/y/z query, with x the number of entity variables, y the number relation predicates and z the number of selection predicates.

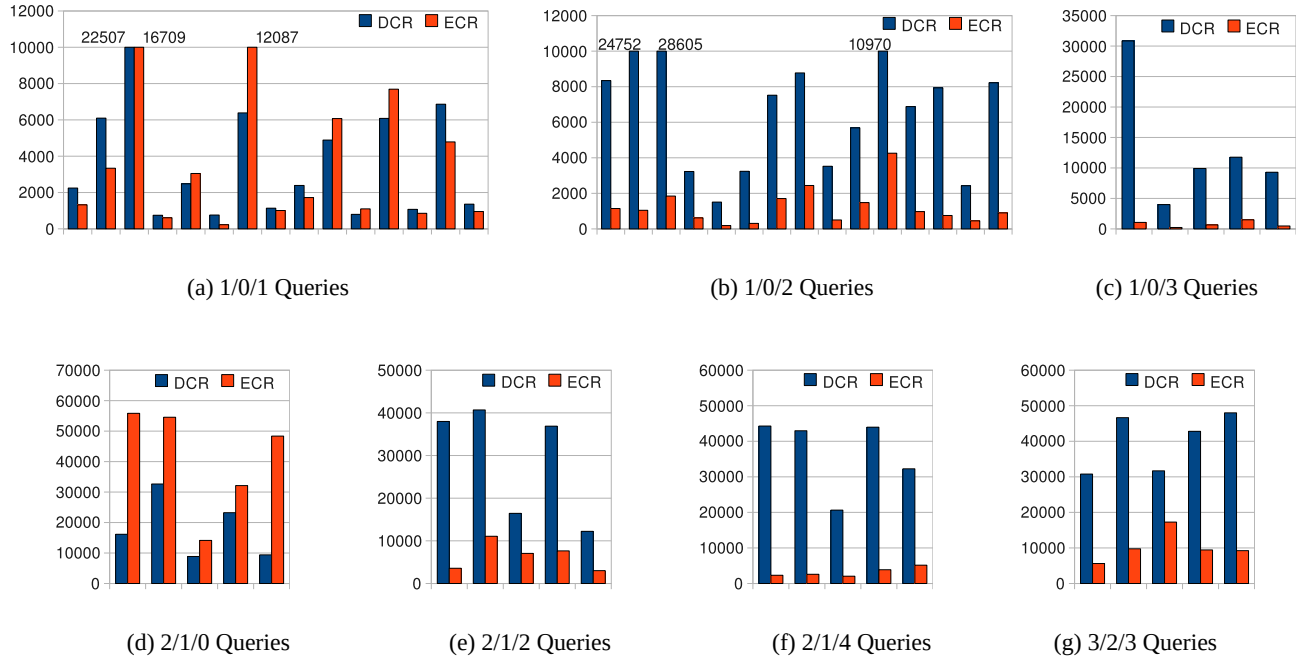
Query Group 1 (G1) contains fifteen 1/0/1 queries, fifteen 1/0/2 queries and five 1/0/3 queries, designed in the following procedure: 1) Design an 1/0/3 query Q; 2) Create three 1/0/2 queries by trimming one predicate off Q; 3) Create three 1/0/1 queries by trimming two predicates off Q; 4) Repeat steps 1-3 for five different Q's.

Query Group 2 (G2) contains five 2/1/0 queries, five 2/1/2 queries and five 2/1/4 queries, designed in the following procedure: 1) Design a 2/1/4 query Q, each variable with two selection predicates; 2) Create one 2/1/2 query by trimming one selection predicate off each variable; 3) Create one 2/1/0 query by trimming off all selection predicates; 4) Repeat steps 1-3 for five different Q's.

Query Group 3 (G3) is created from the five 2/1/2 queries in G2. For each 2/1/2 query, a new variable v is added in, with a selection predicate on v and a relation predicate between v and one of the existing variable. Thus, G3 has five 3/2/3 predicates.

[2]http://download.wikimedia.org

[3]These articles are still used as part of corpus to search answers.



(a) 1/0/1 Queries (b) 1/0/2 Queries (c) 1/0/3 Queries

(d) 2/1/0 Queries (e) 2/1/2 Queries (f) 2/1/4 Queries (g) 3/2/3 Queries

Figure 3: Disk I/O Comparison between DCR and ECR on G1,G2 and G3

Figure 3(a)-(c) shows comparison results on G1, whose queries involving only one variable. To fit the figures for better visibility, we cut tall bars at the level of 10,000 and attach the actual disk I/O counts beside the cut bars. It can be seen that, there is no clear difference between DCR and ECR on 1/0/1 queries. However, on 1/0/2 queries, ECR incurs significantly less disk I/Os than DCR. And when it comes to 1/0/3 queries, ECR can be orders of magnitude faster than DCR.

It is important to note that as more selection predicates are added to the queries, processing cost incurred by ECR could even reduces. For example in Figure 3(a), many red bars are high above 2000, while in (b), most are below or close to 2000. The reason is that, when there are multiple predicates involving the same variable, ECR will find the intersection of entities in TEPLs of all predicates' keywords. As more keywords are introduced in by additional predicates, this intersected set becomes smaller and smaller. Hence less entities needs to retrieve evidences. Therefore, even though ECR is processing more predicates, incur less disk I/Os.

Figure 3(d)-(f) compares DCR and ECR on G2. Queries in this group involve two variables. On 2/1/0 queries, ECR appears to be noticeably worse, costing 1.5 to 5 times the disk I/O of DCR. The reason is that 2/1/0 queries are single relation predicate queries. ECR applies relation splitting to the predicate and evaluates two split predicates separately. While on the side, DCR is able process single relation predicate as conveniently as single selection predicate. However, as we add more predicates on each variables, we observe that ECR has significant drops of disk I/O on 2/1/2 and 2/1/4 queries. The reason is the same as discussed before for G1.

Finally 3(g) shows that ECR still scales well on G3, when there are three variables and two relation predicates in a query, costing in general 1/2 to 1/5 disk I/Os of DCR. We stopped the experiments at 3/2/3 queries due to limit of time. However, based on empirical results on G1 and G2, it is reasonable to believe that for 3/2/N queries, where N>3 ECR is likely to show more significant advantage over DCR.

In summary, ECR is not as good as DCR at processing single relation predicate queries. For single selection predicate queries, it is query-dependent about which one is better. But their difference (in terms of disk I/O) is not significant. When each variable in a query is involved in multiple predicates, ECR begins to show its pruning power and unequivocally beats the performance of DCR. To conclude, between DCR and ECR, ECR (together with the enabling DCI index) is a clear choice for processing complex SSQ queries.

5.2 Efficiency on Pre-Joined Posting List

Very recently, [30] proposed several advanced posting lists to speedup entity search with DCI, including (1) pre-joined posting list between a TDPL and a DEPL, (2) pre-joined posting list between two TDPLs and (3) neighborhood posting list (first introduced in [9] and used as contextual posting list in [30]). However, completely building all advanced posting lists are too hugh to afford. Hence, [30] studied how to selectively build them as trade-off between space and efficiency.

For ECI, it is possible to build counterparts of all these advanced posting lists. However, this paper take a simple approach and leave the comprehensive study, including index selection, as future work.

Regardless of space consumption, we blindly build pre-joined posting list between each pair of TDPL and DEPL for DCI. Since all basic TDPLs and DEPLs are covered by pre-joined posting lists, there is no need to retain them. Hence, they are removed from index. The index consisting of purely pre-joined posting lists is referred to as JDCI. Similarly, we pre-joined every pair of TEPL and EDPL in ECI and remove the basic ones in ECI. The new index is referred to as JECI. The DCR and ECR algorithms are also slightly modified to accommodate changes in posting list structures. They new algorithm is referred to as aDCR and aECR respectively.

The comparison result is summarized as follows.Disk I/O cost of aDCR is mostly comparable to aECR on 1/0/1, 1/0/2, 2/1/0, 2/1/2 and 3/2/3 queries. This is because pre-joined posting lists are usually much shorter than basic posting lists. On one hand, there is a

Table 2: Rankin Quality of SSQ

	INEX17		OWN28	
	nDCG	**MAP**	**nDCG**	**MAP**
BCM	0.933	0.860	0.922	0.808
ER	0.912	0.790	0.862	0.658

lower bound of evidences that must be retrieved, the ground truth set; on the other, the posting lists are shortened a lot due to pre-joining, ruling out many trash evidences in advance. Hence, aDCR does not waste too much on retrieving trash evidences and aECR has a hard time showing its pruning power.

We also observe that when there are three or more predicates on each variables (1/0/3, 2/1/4 queries), aECR seems better than aDCR, although not very significant. This means that, aECR could still be more efficient than aDCR when processing very complex queries. The inherent reason behind this phenomenon is that aDCR still have to evaluate each predicates independently while aECR can leverage more predicates to prune trash evidences, the more predicates the better pruning power.

Overall, it is difficult to claim either algorithm to be clearly better at this moment. However, for large Web corpus, it is not affordable to build fully pre-joined indexes. Hence, we look forward to a more comprehensive study on the two retrieval approaches over large corpus, with investigation on index selection.

5.3 Quality of SSQ Answer

As part of our research on SSQ, we also studied the ranking problem. However, due to space limit, we only report our major results on ranking and refer readers to [23] for technical details.

Our experiments on ranking is carried out on two test set, INEX17 and OWN28. INEX17 contains 17 queries adapted from topics used in Entity Ranking track of INEX 2009 [2]. OWN28 contains 28 manually designed queries. We compared our proposed bounded cumulative ranking model (BCM) with EntityRank [13] model (ER), over both query set. The comparison is shown in Table 2. The measure we used are nDCG (Normalized Discounted Cumulative Gain) and MAP (Mean Average Precision), both being commonly adopted ranking evaluation measures. In Table 2, BCM is consistently better than ER. Further evaluation on precision-at-k reveals that, ER is as good as BCM at ranking top-1 answer, but its precision drops quickly below 0.7 around k=4 or 5, while BCM still maintains precision around 0.7 at k=10.

6. CONCLUSION

In this paper, we proposed a novel querying mechanism, Shallow Semantic Query, which enables users to issue structured entity-centric queries over textual content and obtain direct answers. We proposed an efficient Entity Centric Retrieval algorithm based on our newly designed Entity Centric Index. Experiments on a prototype showed that our algorithm is significantly faster than a baseline algorithm extended from existing entity search systems.

7. REFERENCES

[1] http://www.w3.org/tr/rdf-sparql-query.

[2] INEX 2009 entity-ranking track.

[3] TREC 2009 entity track: Searching for entities and properties of entities.

[4] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *DL*, 2000.

[5] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, , and Z. Ives. DBpedia: A nucleus for a Web of open data. In *6th Int.l Semantic Web Conf.*, 2007.

[6] M. Banko, M. J. Cafarella, S. Soderl, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *In IJCAI*, pages 2670–2676, 2007.

[7] S. Brin. Extracting patterns and relations from the world wide web. In *WebDB*, 1998.

[8] W. Bruce Croft and H.-J. Schek. Introduction to the special issue on database and information retrieval integration. *The VLDB Journal*, 17(1):1–3, 2008.

[9] M. J. Cafarella and O. Etzioni. A search engine for natural language applications. In *WWW*, pages 442–452, 2005.

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

[11] S. Chakrabarti, K. Puniyani, and S. Das. Optimizing scoring functions and indexes for proximity search in type-annotated corpora. In *WWW*, pages 717–726, 2006.

[12] S. Chaudhuri, R. Ramakrishnan, and G. Weikum. Integrating DB and IR technologies: What is the sound of one hand clapping? In *CIDR '05*, pages 1–12, 2005.

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

[14] E. Chu, A. Baid, T. Chen, A. Doan, and J. Naughton. A relational approach to incrementally extracting and querying structure in unstructured data. In *VLDB*, 2007.

[15] W. W. Cohen. Information extraction and integration: an overview. 2004.

[16] P. DeRose, W. Shen, F. Chen, A. Doan, and R. Ramakrishnan. Building structured Web community portals: a top-down, compositional, and incremental approach. In *VLDB '07*, pages 399–410, 2007.

[17] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. A. Tomlin, and J. Y. Zien. SemTag and seeker: bootstrapping the semantic Web via automated semantic annotation. In *WWW*, 2003.

[18] A. Doan, R. Ramakrishnan, and S. Vaithyanathan. Managing information extraction: state of the art and research directions. In *SIGMOD '06*, pages 799–800, 2006.

[19] O. Etzioni, M. Banko, S. Soderland, and D. S. Weld. Open information extraction from the Web. *Commun. ACM*, 51(12):68–74, 2008.

[20] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *ACM SIGMOD Record*, 34(4):27–33, 2005.

[21] E. Kandogan, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar semantic search: a database approach to information retrieval. In *SIGMOD*, pages 790–792, 2006.

[22] G. Kasneci, F. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. NAGA: Searching and ranking knowledge. In *ICDE*, pages 953–962, 2008.

[23] X. Li, C. LI, and C. Yu. Structured querying of annotation-rich web text with shallow semantics. Technical report, University of Texas at Arlington, 2010.

[24] A. McCallum. Information extraction: Distilling structured data from unstructured text. *Queue*, 3(9):48–57, 2005.

[25] D. Petkova and W. B. Croft. Proximity-based document representation for named entity retrieval. In *CIKM*, 2007.

[26] Rakesh Agrawal et al. The claremont report on database research. *ACM SIGMOD Record*, 37(3):9–19, 2008.

[27] F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO: a core of semantic knowledge unifying WordNet and Wikipedia. In *WWW '07*, pages 697–706, 2007.

[28] A.-M. Vercoustre, J. A. Thom, and J. Pehcevski. Entity ranking in wikipedia. In *SAC*, 2008.

[29] H. Zaragoza, H. Rode, P. Mika, J. Atserias, M. Ciaramita, and G. Attardi. Ranking very many typed entities on Wikipedia. In *CIKM '07*, pages 1015–1018, 2007.

[30] M. Zhou, T. Cheng, and K. C.-C. Chang. Data-oriented content query system: searching for data into text on the web. In *WSDM '10: Proceedings of the third ACM international conference on Web search and data mining*, pages 121–130, New York, NY, USA, 2010. ACM.

APPENDIX

A. SSQ RANKING FRAMEWORK

Definition 4 (SSQ Ranked Result): Given a query q=$\langle V, D, P\rangle$, we introduce three scoring functions F_S, F_R and F_A, such that the ranked result of q is a ranked list of $\langle$tuple, score$\rangle$ pairs $\langle t, s\rangle$, satisfying the following conditions:

– t is an answer tuple;

– Each selection predicate p=$\langle V_p$=$\{v_k\}$, $C_p\rangle \in P$ has a computed score $p^{F_S}(t) > 0$ with regard to t;

– Each relation predicate p=$\langle V_p$=$\{v_{k_1}, \ldots, v_{k_i}\}$, $C_p\rangle \in P$ similarly has a computed score $p^{F_R}(t) > 0$.

– The tuple score s is greater than 0. It is a scalar value measuring the likelihood of tuple t being a correct answer to q. It is computed as $F_A(\{p(t)|p \in P\})$, where F_A aggregates the scores of all the predicates obtained via F_S and F_R.

By the above definition, each answer tuple is scored at three levels. First at the entity level, every selection predicate is scored by F_S, to evaluate how well an entity satisfies the constraints on itself. Then at the relation level, F_R evaluates each relation predicate with regard to how well the relation among entities holds true. Finally at the query level, F_A evaluates how well a tuple of entities satisfy the predicates altogether, based on the scores of individual predicates. The tuples are ranked by their query level scores s.

As an example, suppose t_1=$\langle$Jerry Yang, Yahoo!$\rangle$ is an answer to Query 1, with $p_1(t_1)$=0.8 (i.e., the score of Jerry Yang being a Stanford graduate is 0.8), $p_2(t_1)$=0.7, $p_3(t_2)$=0.8, then r = 2.3, assuming F_A=SUM. On the contrary, t_2=$\langle$Jerry Yang, Google$\rangle$ is less likely to be an answer since $p_3(t_2)$ could be close to 0.

We shall make several important notes on Definition 4:

– Our definition of SSQ ranking result is more a generic ranking framework than a particular ranking model. The definition does not dictate any specific ranking function or features for scoring, allowing customization.

– The framework relies on an important assumption, *predicate independency*, meaning that every predicate is scored independently, even if two predicates involve the same set of entities.

– We focus on conjunctive predicates, thus the AND clause in the SQL-like skeleton. Therefore a query answer t should satisfy all the query predicates.

– A tuple t is said to "satisfy" a predicate p if $p(t)$>0. A result tuple should thus have positive scores on all the predicates.

B. PRUNING CAPABILITY OF ECR

To better understand the pruning capability of ECR, we compare the executions of inner-most loop in both bECR and ECR, because the inner-most loop contains the most costly step in the two algorithms, the posting list merge join on *occur*, $\bowtie^{occur}$. This refers to line 9 of bECR and line 11 of ECR.

First of all, in either algorithm, the join $\bowtie^{occur}$ should be performed for every split predicate $p^v \in S$. The parameter that actually makes difference between the two algorithms is how many times $\bowtie^{occur}$ is executed.

For a split predicate group $S(v)$ in ECR, $R^1(v)$ is the intersection of entities in EDPL of v and all TEPLs of all $p^v \in S(v)$. Denote x^1 as the set of entities in the posting list of x,

$$R^1(v) = v^1 \cap \Big(\bigcap_{p^v \in S(v)} \bigcap_{w \in C_p} w^1 \Big)$$

Governed by the loop of line 8, $\bowtie^{occur}$ is executed $|R^1(v)|$ times, for each $p^v \in S(v)$. Total executions of line 11 given query q is

$$N_{ECR} = \sum_{v \in V} \sum_{p^v \in S(v)} |R^1(v)| = \sum_{p^v \in S} |R^1(v)|$$

bECR computes a different R^1 for each p^v, denoted as $R^1(p^v)$. By the join $\bowtie^{entity}$ in line 7, $R^1(p^v)$=$v^1 \cap \bigcap_{w \in C_p} w^1$. Controlled by the loop of line 7 $\bowtie^{occur}$ is executed $|R^1(p^v)|$ times. Eventually, bECR executes line 9 by N_{A2} times, where $N_{bECR} = \sum_{p^v \in S} |R^1(p^v)|$

Obviously, $\forall v$, $R^1(v) \subseteq R^1(p^v)$ and hence $|R^1(v)| \leq |R^1(p^v)|$. and hence we have $N_{ECR} \leq N_{bECR}$.

Table 3: Ten Types from Wikipedia

Type	(E)ntities	(O)ccurrences	O/E
AWARD	1,045	626,340	600
CITY	70,893	28,261,278	389
CLUB	15,688	5,263,865	335
COMPANY	24,191	9,911,372	409
FILM	41,344	3,047,576	74
NOVEL	16,729	1,036,596	63
PERSON	427,974	38,228,272	89
PLAYER	95,347	2,398,959	25
SONG	29,934	732,175	24
UNIVERSITY	19,717	6,141,840	311
TOTAL	742,862	95,648,273	129

C. DATA SET

Corpus - Our system building and experiments were carried out on the 2008-07-24 snapshot of Wikipedia . We removed all the category pages and administrative pages and obtained about 2.4 million articles as our corpus. For each article, we removed all its section titles, tables, infoboxes, and references. Although tables and infoboxes also present valuable information for structured query, we feel they are significantly different from the main body of the article in both format and data characteristics, thus should be treated separately by other techniques such as the IE-based methods discussed in Section 2.

Entity Set - The Wikipedia articles serve as both the text corpus for finding query answers and the repository of named entities. Each article represents a unique entity named by the article title. We manually define ten entity types (see Table 3) and use simple regular expressions to assign entities (articles) to these types based on their categories. For example, if an article belongs to a category whose name ends with "novels", e.g.,, "British novels", we treat the article as an entity of type NOVEL. About 0.75 million out of the 2.4 million articles were assigned to the 10 types in the system. An entity can fall into multiple types, e.g., David Beckham belongs to PLAYER, and the more general category, PERSON. This simple method turns out to be quite accurate and sufficient for demonstrating the effectiveness of the SSQ system.

Entity Annotations - To identify occurrences of entities in the corpus, we exploited *internal links* in Wikipedia articles. An internal link is a hyperlink in some Wikipedia article to another Wikipedia article. Example 4 shows a sentence with one internal link, in which the anchor text "Cisco" (right to the vertical bar in double brackets) links to an article titled "Cisco Systems" (left to the vertical bar). SSQ interprets this internal link as an occurrence of the entity Cisco Systems and that the sentence uses one token, "Cisco", to reference it.

Example 4 (Internal Link): Cisco Career Certifications are IT professional certifications for [[Cisco Systems|Cisco]] products.

D. EXAMPLE QUERIES

We show some example queries from each query group and list the top 3 answers (if there are 3) returned by our prototype system.

```
Q1: 1/0/1
SELECT x FROM PERSON AS x
WHERE x:["Stanford" "graduate"]

William_Reddington_Hewlett
William_Forsyth_Sharpe
Jerry_Yang


Q2: 1/0/2
SELECT x FROM PERSON AS x
WHERE x:["Stanford" "graduate"]
  AND x:["computer" "science"]

Bill_Gates
John_McCarthy
Alan_Kay


Q3: 1/0/3
SELECT x FROM PERSON AS x
WHERE x:["Stanford" "graduate"]
  AND x:["computer" "science"]
  AND x:["Russian"]

Bill_Gates
Sergey_Brin


Q4: 1/0/1
SELECT x FROM FILM AS x
WHERE x:["star" "Tom Hanks"]

Cast_Away
Saving_Private_Ryan
Big


Q5: 1/0/2
SELECT x FROM FILM AS x
WHERE x:["star" "Tom Hanks"]
  AND x:["World War II"]

Saving_Private_Ryan
The_Pacific_(miniseries)
Independent_film


Q6: 1/0/3
SELECT x FROM FILM AS x
WHERE x:["star" "Tom Hanks"]
  AND x:["World War II"]
  AND x:["Steven Spielberg"]

Saving_Private_Ryan
The_Pacific_(miniseries)
Independent_film


Q7: 2/1/2
SELECT x, y FROM PERSON AS x, COMPANY AS y
WHERE (x,y):["found"]
  AND x:["Stanford" "graduate"]
  AND y:["Silicon Valley"]

Jerry_Yang  Yahoo!
Scott_McNealy  Sun_Microsystems
David_Packard  Hewlett-Packard_Company


Q8: 2/1/4
SELECT x, y FROM PERSON AS x, COMPANY AS y
WHERE (x,y):["found"]
  AND x:["Stanford" "graduate"]
  AND x:["Russian"]
  AND y:["Silicon Valley"]
  AND y:["search engine"]

Sergey_Brin  Google
Bill_Gates  Microsoft


Q9: 3/2/3
SELECT x,y,z
FROM PERSON AS x, COMPANY AS y CITY AS z
WHERE (x,y):["found"]
  AND (y,z):["headquarter"]
  AND x:["Stanford" "graduate"]
  AND y:["Silicon Valley"]
  AND z:["Apple Computer"]

Scott_McNealy  Sun_Microsystems
                    Cupertino,_California
Vinod_Khosla  Sun_Microsystems
                    Cupertino,_California
Andy_Bechtolsheim  Sun_Microsystems
                    Cupertino,_California


Q10: 2/1/2
SELECT x, y
FROM PERSON AS x, FILM AS y
WHERE (x,y):["star"]
  AND x:["Academy Award"]
  AND y:["Australian" "actor"]

Brokeback_Mountain  Heath_Ledger
Gladiator_(2000_film)  Russell_Crowe
A_Beautiful_Mind_(film)  Russell_Crowe


Q11: 2/1/4
SELECT x,y FROM PERSON AS x, CLUB AS y
WHERE (x,y):["led"]
  AND x:[final MVP]
  AND x:["Rookie of the Year"]
  AND y:["NBA" "champion"]
  AND y:["Eastern Conference"]

Michael_Jordan  Chicago_Bulls
Rick_Barry  Golden_State_Warriors
Larry_Bird  Boston_Celtics
```