Structured Querying of Annotation-Rich Web Text with Shallow Semantics

Xiaonan Li Chengkai Li Cong Yu

Submitted: March 22, 2010 Updated: April 21, 2010

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 Web search engines, 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 report, we introduce a novel entity-centric structured querying mechanism called Shallow Semantic Query (SSQ) to overcome this limitation. We cover two key technical issues with regard to SSQ, ranking and query processing. Comprehensive experiments show that (1) our ranking model beats state-of-the-art entity ranking methods; (2) the proposed query processing algorithm based on our new Entity-Centric Index is more efficient than a baseline extended from existing entity search systems.

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 [27]. Recently there have been extensive efforts along this general direction [13, 22, 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 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
```

Our goal is to provide a declarative query interface for such tasks and an evaluation mechanism that produces answers directly. To 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 a flat set of keywords. There are three elementary concepts within this SSQ query. First, the query centers on two *entity variables*, x and y. Variable x 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_1) . Third, a *relation predicate* specifies the relation between p and c (p_3).

Developing SSQ presents a significant research challenge and involves several important building pieces. Named entity recognition, disambiguation and categorization are required for properly identifying entities and assigning them to types. Moreover, the noise and spam on Web pages must be addressed in order to reach a quality system. Each of these is an important research problem on its own and has been studied heavily [23, 17, 21, 9]. While it would be rewarding to apply the results in these areas as building blocks in developing SSQ, as an initial attempt, we choose to focus on a special corpus, Wikipedia, which consists of a rich body of community-edited articles annotated with name-entities.

Since its inception in January 2001, Wikipedia has risen to be the largest encyclopedia ever created, containing nearly 3 million articles in English alone as of 2009. In the meantime, Wikipedia articles have amazingly evolved, from mostly plain texts at earlier stage to current ones with substantial structural annotations. Some of the important annotations include *internal links* (links to other Wikipedia articles), *infoboxes* (summary tables of articles) and *categories* (which group articles for navigational convenience). As a result, it is now the primary knowledge source for many users on a wide variety of topics, including people, institutions, geographical locations, events, etc.

The distinguishing characteristics of Wikipedia help ease the aforementioned problems (details in Section 6.1) and thus allow us to focus on the central challenges of SSQ itself, i.e., how to evaluate SSQ queries. Moreover, the high-impact of Wikipedia on our society makes an SSQ system over Wikipedia itself a valuable artifact. It is our hope that the results from this report would lead to the thorough investigation of SSQ over generic Web pages, once the enabling technologies (e.g., Web-scale entity recognition and disambiguation) become available.

Challenges While the structured information in Wikipedia alleviates many peripheral problems such as entity detection, we are still faced with several key challenges in SSQ. *First*, the notion of Shallow Semantic Query and the semantics of query results must be properly defined. *Second*, 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. *Third*, 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 involve multiple entities and their inter-relationships. This report covers all three issues.

2 Related Work

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

The **DB-based** approach explicitly extracts entity-relationship information from text into relational databases. SQL queries can thus be issued over the populated databases. This approach is constrained by the capability of the information extraction (IE) [7, 4, 15, 25, 16, 19, 6, 20] and natural language processing (NLP) [20, 10, 11, 6] 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 the extraction phase, such information is lost and could not be queried.

The **Semantic Web** approach [28, 24, 18, 5] explicitly encodes entities and their relations (and general knowledge) in RDF [1] format, 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. However, the building blocks of Semantic Web, RDF data, must be collected beforehand. Some systems reliably extract RDF from structured/semi-structured semantic data sources [5, 28], 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 [18], but the quality control is much more difficult. Besides, independently developed Semantic Webs face the issue of interoperability [5].

The **IR-based** approach, exemplified by the recently formed entity search and ranking problems in the IR community [12, 26, 2, 3, 30, 29], focuses on retrieving named entities (from free text) relevant to certain contextual constraints. 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 [29, 2], HITS [29] and PageRank [14] are commonly applied with adaptation.

Shallow Semantic Query uniquely takes the DB-IR integrated approach in pursing entity-centric tasks. On the one hand, SSQ queries have explicit structured components (typed entity variables, selection/relation predicates), offering greater expressiveness than pure keyword queries. On the other hand, each individual predicate is a keyword-based constraint, avoiding the strong requirements of explicit schema (as in database) and semantics (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 x:["Stanford" "graduate"] requires a PERSON to co-occur with keywords "Stanford" and "graduate" in the same context. In short, SSQ captures entity properties and relationships through shallow syntax requirements implied by user-specified predicates at query time[1], rather than pre-extracting them at system construction time. Although such syntax clue is by no means rigorous or error-proof, it becomes robust when we take into consideration the repetitive nature of the Web: true facts are more likely to be stated on many different pages. This intuition has been widely used in Web search and mining [7, 4, 14].

The studies most related to SSQ are [12, 14, 31]. [12] learns an optimal scoring function on proximity feature, but it only scores entities by one evidence and makes no attempt to integrate evidences found in multiple documents to improve ranking. Leveraging the redundancy of the Web, [14] aggregates scores of locally evaluated evidences into global scores. However, neither of the two studies

[1]We acknowledge that, the effectiveness of such entity-relationship queries partially relies on the users capability in providing proper keyword constraints, just like in IR queries.

tackles the challenge of improving ranking beyond the first few true answers. Moreover, they only focus on queries comparable to our single-predicate queries and thus do not study structured (multi-predicate) queries. [31] proposes a Content Query Language for querying entities, but essentially is also limited to single-predicate queries. All these works utilize sight variations of the traditional full text index. These variations are exemplified by the Document-Centric Index in Section 5.1. SSQ makes use of our novel Entity-Centric Index (Section 5.2, 5.3) to achieve better efficiency in processing structured queries.

In summary, SSQ is unique in its ability to answer complex structured queries directly over textual corpus. Although currently experimented with Wikipedia, it can be extended to other corpora with assistance of entity identification technology. To promote our vision on structured query over type-annotated corpus, this report provides a full introduction of our current research status on SSQ. The rest of the report is organized as follows:

- Section 3 brings forth Shallow Semantic Query, an entity-centric structured querying facility for querying named entities by their properties and relationships, and formalizes its semantics. Both the ranking problem and the entity retrieval problem are formally defined.
- Section 4 introduces our ranking method based on three position-based features that exploit entity-keyword co-occurrences.
- A novel Entity-Centric Index and a corresponding Entity-Centric Retrieval algorithm for efficient processing of SSQ queries are presented in Section 5.
- Comprehensive experiments are provided in Section 6.

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., x in Query 1) are bound to typed entities and are associated with keyword constraints to form query *predicates* (e.g., x:["Stanford" "graduate"]), which express the semantic criteria in selecting and relating entities. Formally:

Definition 1 (Shallow Semantic Query): 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 of the corresponding $v_i \in V$. Two variables can have 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=$\{x, y\}$, D=$\{$PERSON, COMPANY$\}$, P=$\{p_1, p_2, p_3\}$, where p_1=$\langle\{x\}$, {"Stanford", "graduate"}$\rangle$, p_2=$\langle\{y\}$, {"Silicon Valley"}$\rangle$, and p_3=$\langle\{x, y\}$, {"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 COMPANY y is the only output variable, only one $\langle$Yahoo!$\rangle$ will be in the output. Without loss of generality, we assume U=V throughout our discussion. 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. To concentrate on the SSQ semantics, we omit the formal definition of query syntax in this report 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 explicitly stated in 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. Given a predicate p, if a sentence contains all the phrases in C_p and one entity for each variable in V_p, it is considered an *evidence* for p and these entities in whole are said to satisfy p. Suppose three evidences are found in the corpus as a result of Query 1:

s_1: *Jerry Yang graduated from Stanford University ...*
s_2: *... a senior manager at Yahoo! in Silicon Valley.*
s_3: *Jerry Yang co-founded Yahoo!.*

Jerry Yang satisfies p_1 by evidence s_1; Yahoo! satisfies p_2 by evidence s_2; and they together satisfy p_3 by s_3. Assembling the information together, the entity tuple $\langle$Jerry Yang, Yahoo!$\rangle$ is composed as an answer to the query since it satisfies all the query predicates. In this paper, we assume sentence as the unit of co-occurrence contexts for evidences, while in reality, contexts of coarser granularities, such as paragraphs and documents, are possible.

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 belongs to v_i's type $d_i \in D$.

– Given a predicate p=$\langle V_p, C_p\rangle$, we use t_p to represent the sub-tuple of t such that each entity $e \in t_p$ is instantiated from a corresponding $v \in V_p$. Take p_1 in Query 1 for example. t_{p_1}=$\langle$Jerry Yang$\rangle$ because V_{p_1} has only one variable x and Jerry Yang is instantiated from x. Similarly, t_{p_3}=t.

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

Definition 3 (Evidence Representation): Given an answer tuple t to query q=$\langle V, D, P\rangle$, an evidence of t_p for predicate $p \in P$ is a quadruple $\langle doc, sent, \hat{V}_p, \hat{C}_p\rangle$:

– doc and $sent$ refer to the document ID and the sentence number that together identify a unique sentence in the corpus.

– $\hat{V}_p$ is a set of entity spans. For each $v \in V_p$, there is a pair $\langle f, l\rangle \in \hat{V}_p$, which is the span of entity $e \in t_p$, where e is the instantiation of v. $\langle f, l\rangle$ are the positions of the first and last terms of the phrase representing e in the sentence.

– $\hat{C}$ is a set of phrase positions. For each phrase $c \in C_p$, there is a corresponding $\hat{c} \in \hat{C}_p$, which is the position of the first term of c in the sentence.

Suppose the aforementioned s_1 is the 8th sentence of document 9. It is an evidence of $\langle$Jerry Yang, Yahoo!$\rangle$ for predicate p_1, where Jerry Yang spans from position 0 to 1 and the two phrases ("Stanford" and "graduate") are at positions 4 and 2. This evidence is represented as $\langle 9, 8, \{\langle 0, 1\rangle\}, \{4, 2\}\rangle$. Note that there can be multiple evidences of Jerry Yang for predicate p_1, each being a sentence containing Jerry Yang, "Stanford", and "graduate". We denote all evidences of t_p for predicate p by $\phi_p(t)$ (or equivalently $\phi_p(t_p)$, since entities other than those in t_p are irrelevant to p) and refer to t_p as the *signature* of $\phi_p(t)$. Without loss of generality, we will use sentence and evidence interchangeably unless distinction is needed, since we only consider sentence as the evidence context.

Problem 1 (Position-based Ranking): Denote all answers to query q=$\langle V, D, P\rangle$ by A^q. Our goal is to rank the answers in A^q according to information provided by ϕ^q=$\{\phi_p | \phi_p$=$\bigcup_{t \in A^q} \phi_p(t), p \in P\}$. Since the information that is used for ranking, ϕ^q, is primarily position information (i.e., document IDs, sentence numbers, entity spans and phrase positions), the problem is called position-based ranking problem, and any ranking technique relying on ϕ^q is classified as position-based ranking.

Our **ranking framework** consists of three scoring functions F^S, F^R and F^A, such that for each answer t: (1) its score on a selection predicate $p \in P$ is given by $F^S_p(t)$, or equivalently $F^S_p(t_p)$; (2) its score on a relation predicate $p \in P$ is given by $F^R_p(t)$, or equivalently $F^R_p(t_p)$; and (3) its final score $F^A(t)$ (the answer score) aggregates all predicate scores obtained via F^S and F^R. Under this framework, the scores for different predicates are computed independently from each other. The

intuition can be explained as follows. In Query 1, whether a PERSON is a Stanford graduate (p_1) is independent from whether she founded any COMPANY (p_3) and certainly irrelevant to whether a COMPANY is in Silicon Valley (p_2).

Each answer tuple is scored at three levels. At the entity level, every selection predicate is scored by F^S, to evaluate how well an entity satisfies the constraints on itself. At the relation level, F^R evaluates how well the relations among entities hold true. 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 answers are ranked by their query-level scores F^A. As an example, suppose t=⟨Jerry Yang, Yahoo!⟩ is an answer to Query 1, with $F^S_{p_1}(t)$=0.8 (i.e., the score of Jerry Yang being a Stanford graduate is 0.8), $F^S_{p_2}(t)$=0.7, $F^R_{p_3}(t)$=0.8, then $F^A(t)$=2.3, assuming F_A is summation.

As noted before, let $\phi_p(t)$ be all evidences of an answer t for predicate p. The task of entity retrieval is to retrieve all evidences of every answer tuple for every query predicate.

Problem 2 (Entity Retrieval): Denote A^q as all answer tuples of query q=$\langle V, D, P\rangle$. The task of entity retrieval is to retrieve ϕ^q=$\{\phi_p | p \in P\}$, where ϕ_p=$\cup_{t\in A^q}\phi_p(t)$. Apparently, a system that can solve the Entity Retrieval problem can be used to support any position-based ranking method.

The following two sections address ranking problem and entity retrieval problem respectively.

4 SSQ Ranking

4.1 Position-Based Features

This section studies three position-based features that are derivable from an evidence. These features are the key components in our Cumulative Model (CM) and Bounded Cumulative Model (BCM) that are introduced later.

4.1.1 Proximity

Intuitively, if the entities in t_p and the keyword phrases in C_p are close to each other in an evidence $s \in \phi_p(t)$, they are likely to belong to the same grammatical unit of the corresponding sentence (e.g., a phrase like *Stanford University graduate Jerry Yang*) and thus form a valid evidence. Given predicate p, we define the proximity of t_p in s as

$$prox_p(t, s) = prox_p(t_p, s) = \frac{\sum_{e\in t_p} |token(e, s)| + \sum_{c\in C_p} |c|}{|scope_p(t_p, s)|}$$

where $|token(e, s)|$ is the number of tokens in s representing entity e; $|c|$ is the number of tokens in phrase c; $scope_p(t_p, s)$ is the smallest scope in s covering all the entities in t_p and all the phrases in C_p (a scope is a consecutive sequence of tokens in s); and consequently $|scope_p(t_p, s)|$ is the total number of tokens in the scope. Note that the proximity value is in the range of [0,1] by this definition.

Different representations may be used in various places to refer to the same entity and may have different number of tokens. For example, the entity IBM may be represented by "IBM", "Big Blue", or "International Business Machine". Hence, $|token(\text{IBM}, s)|$ may be 1, 2, or 3 in different s.

Example 3: The following two sentences are both evidences of the underlined entities for predicate p_1 in Query 1. Evidence s_1 is a valid evidence, supporting a true positive, while s_4 is invalid, supporting a false positive.

s_1: *Jerry Yang graduated from Stanford University ...*

s_4: *A professor at Stanford University, Colin Marlow had a relationship with Cristina Yang before she graduated ...*

Predicate p_1 has two phrases, "Stanford" and "graduate", each with one token, hence $\sum_{c\in C_{p_1}} |c|$=2.
In s_1, the PERSON Jerry Yang is represented by two tokens, "Jerry" and "Yang", hence $\sum_{e\in t_{p_1}} |token(e, s_1)|$=2.

The scope covering the entity and the two phrases spans 5 tokens, from "Jerry" to "Stanford", thus $|scope_{p_1}(t_{p_1}, s_1)|$=5. Therefore, the proximity of Jerry Yang in s_1 is $prox_{p_1}(t_{p_1}, s_1)$=$\frac{2+2}{5}$=0.8. Similarly, the proximity of Colin Marlow in s_4 is $\frac{2+2}{13}$=0.31. Based on proximity alone, we say that s_1 is a more valid evidence and therefore, Jerry Yang is more likely to satisfy p_1 than Colin Marlow, given no other evidence.

4.1.2 Ordering Pattern

An ordering pattern refers to the order of entities and phrases in an evidence. Consider predicate p_1=$\langle\{x\}, \{$"Stanford", "graduate"$\}\rangle$ in Query 1. Let c_1 be the first phrase ("Stanford") and c_2 the second ("graduate"). This predicate has six different ordering patterns (xc_1c_2, xc_2c_1, c_1xc_2, c_2xc_1, c_1c_2x and c_2c_1x). Generally, if we denote all possible patterns of a predicate p by O_p, we have $|O_p|$=$(|V_p|+|C_p|)!$. Note that, extra tokens and punctuations between entities and phrases are irrelevant to the patterns, i.e., *Stanford University graduate, Jerry Yang* and *Stanford graduate Jerry Yang* follow the same pattern c_1c_2x.

We observe that some ordering patterns are better indicators of valid evidences than others. For example, to express that somebody is a graduate of Stanford University, valid evidences often follow the pattern xc_2c_1 (e.g., s_1). Those following another pattern, c_1xc_2, are unlikely to be valid (e.g., s_4). To distinguish strong patterns (those that tend to indicate valid evidences) from weak ones, we may assign a different weight to each pattern, so that entities supported by evidences following strong patterns are scored higher. However, it is impossible to pre-determine the weights since the goodness of ordering patterns are predicate-dependent. To illustrate, xc_2c_1 is a strong pattern for predicate p_1 in Query 1, but may not be equally strong for another predicate p_1'=$\langle\{x$:NOVEL$\}, \{$"by", "Jane Austen"$\}\rangle$, because it is less common to see an evidence such as

... Pride and Prejudice ... Jane Austen ... by ...

In our approach, we assign different weights to different patterns, such that evidences following strong patterns are weighted higher. The weights of ordering patterns for a predicate p are dynamically derived from ϕ_p, the set of all evidences for p. Denoting $\phi_p(o)$ as the subset of evidences following pattern o, we define the weight of o for predicate p as its frequency in ϕ_p,

$$f_p(o) = |\phi_p(o)|/|\phi_p|$$

This definition assumes that strong patterns appear more often than weak ones. Although in theory it may happen that many invalid evidences follow the same pattern, making a weak pattern more common, we do not observe such cases in our experiments.

Another possible direction is leveraging Machine Learning techniques to predict which patterns lead to better results. While we are also exploring this direction as future work, we note here that one significant challenge of the Machine Learning approach is the need to obtain training data, which can be costly in terms of human effort.

4.1.3 Mutual Exclusion

Given a predicate p, multiple evidences in ϕ_p may have the same $\langle doc, sent\rangle$ value (i.e., come from the same sentence). They are evidences of different entities and may follow different ordering patterns. The co-existence of different patterns in one sentence is called *collision* and the patterns are referred to as *colliding patterns*. The mutual exclusion rule dictates that, when collision happens, at most one colliding pattern is effective and the sentence is only considered evidences following that pattern.

Example 4: The following sentence illustrates mutual exclusion rule for p_1 in Query 1. The sentence appears as three evidences, one for each underlined entity. Ric Weiland follows the pattern o_1=xc_2c_1. Paul Allan and Bill Gates follow o_2=c_2c_1x. Semantically, the former pattern is the effective pattern and the sentence is an evidence of Ric Weiland.

After Ric Weiland graduated from Stanford University, Paul Allen and Bill Gates hired him ...

Without understanding the semantics, it is difficult to decide which colliding pattern is absolutely effective. Therefore, we relax the rule with a *credit* mechanism, where every colliding pattern is considered partially effective, and patterns with higher credits are more likely to be effective than those with lower credits. We assume each sentence s (that is an evidence of at least one sub-tuple t_p for predicate p) has a total credit of 1, meaning that there is only one effective pattern. Given a predicate p, denote the colliding patterns in s by $O_p(s)$. Each $o \in O_p(s)$ gets a credit $credit_p(o, s)$, and $\sum_{o \in O_p(s)} credit_p(o, s)$=1.

To allocate credits to the colliding patterns $O_p(s)$, we adopt the intuition that patterns followed by more prominent entities are more likely to be effective. Specifically, let $T_p(o, s)$ be all sub-tuples on p following pattern o in s. For each $o \in O_p(s)$, we choose a representative from $T_p(o, s)$, denoted by $T_p^*(o, s)$, which is the one with the highest proximity value, i.e., $T_p^*(o, s)$=$\arg\max_{t_p \in T_p(o,s)} prox_p(t_p, s)$. We compare the representatives (and thus the patterns that they follow), by how prominent they are, i.e., by their overall numbers of evidences in ϕ_p. The credit of o in sentence s is

$$credit_p(o, s) = \frac{|\phi_p(T_p^*(o, s))|}{\sum_{o' \in O_p(s)} |\phi_p(T_p^*(o', s))|}$$

where $\phi_p(T_p^*(o, s))$ is the set of evidences of $T_p^*(o, s)$ for predicate p. Note that we choose the most proximate sub-tuple as the representative of a colliding pattern and allocate credits based on representatives only. The intuition is that the most proximate sub-tuple is most likely to form a grammatical unit with phrases in C_p, and hence the most reliable one for allocating credits.

In Example 4, t^1=$T_{p_1}^*(o_1, s)$=Ric Weiland (i.e., the representative of patterns o_1 is Ric Weiland) since he is the only PERSON in s following pattern o_1. t^2=$T_{p_1}^*(o_2, s)$=Paul Allen because he has higher proximity (0.67) than Bill Gates (0.44), though both follow o_2. Suppose Ric Weiland is found in 4 evidences ($|\phi_{p_1}(t^1)|$=4) and Paul Allen in 2 ($|\phi_{p_1}(t^2)|$=2). Then, $credit_{p_1}(o_1, s)$=$\frac{4}{4+2}$=0.67 and $credit_{p_1}(o_2, s)$=0.33.

Note that the pattern credit here is different from the weight of pattern in Section 4.1.2. The weight of pattern o is a global measure (aggregating over ϕ_p) of how frequent, and thus how reliable, pattern o is. The credit of o, on the contrary, is a local measure particular to each sentence s, indicating how likely o is the effective pattern in s.

4.2 Single-Predicate Scoring

So far, we have introduced all the features for evaluating the validity of an individual evidence. Integrating these features together, this section presents Cumulative Model (CM) for scoring an answer on a single predicate. We assume that F^S is the same as F^R (i.e., the same function is used for scoring all predicates), hence for brevity, we use $F_p(t)$ instead of $F_p^S(t)$ and $F_p^R(t)$.

Let $\phi_p(t, o) \subseteq \phi_p(t)$ be all evidences of t for predicate p that follow pattern $o \in O_p$. Our Cumulative Model (**CM**) is

$$F_p(t) = \sum_{o \in O_p} (f_p(o) \sum_{s \in \phi_p(t,o)} prox_p(t, s) credit_p(o, s))$$

where $f_p(o)$ is the weight of pattern o; $prox_p(t, s)$ is t_p's proximity in evidence s; $credit_p(o, s)$ is the credit of o in s.

The model divides $\phi_p(t)$, t's evidences for p, into $|O_p|$ groups, $\{\phi_p(t, o) | o \in O_p\}$, so that evidences in each group follow the same pattern. For each group $\phi_p(t, o)$, the model computes a *group score* (the inner summation). The group scores are linearly combined using weights $f_p(o)$ (the outer summation), such that the group scores of strong patterns account more in $F_p(t)$. The kernel of the function, $prox_p(t, s)\ credit_p(o, s)$, evaluates the validity of s being an evidence of t for predicate p. It is monotonic to both the proximity of t_p and the credit of t_p's pattern o. Answers supported by evidences having higher proximities and pattern credits will accumulate higher scores and thus ranked higher.

It is interesting to note that CM can be customized easily by switching on and off its component features, so that we can evaluate the effectiveness of individual features. While detailed evaluations are presented in Section 6, below we list three important customizations.

Table 1: Example Answers

	x	y	p_1	p_2	p_3	Π	Σ
t_1	Jerry Yang	Yahoo!	0.8	0.7	0.8	0.448	2.3
t_2	Larry Page	Google	0.6	0.5	0.6	0.18	1.7
t_3	Scott McNealy	Cisco	0.9	0.8	0.2	0.144	1.9
t_4	Bill Gates	IKEA	0.3	0.1	0.2	0.006	0.6

COUNT is the straightforward baseline method that scores a tuple t by its number of supporting evidences for predicate p, i.e., $F_p(t)$=$|\phi_p(t)|$. It can be reduced from the CM model by turning off all the features, i.e., by setting $prox_p(t,s) \equiv 1$, $credit_p(o,s) \equiv 1$, and $f_p(o) \equiv 1$:

$$F_p(t) = \sum_{o \in O_p} (1 \sum_{s \in \phi_p(t,o)} 1) = \sum_{o \in O_p} |\phi_p(t,o)| = |\phi_p(t)|$$

PROX applies only the proximity feature (Section 4.1.1) and is reduced from CM by $credit_p(o,s) \equiv 1$ and $f_p(o) \equiv 1$:

$$F_p(t) = \sum_{o \in O_p} \sum_{s \in \phi_p(t,o)} prox_p(t,s) = \sum_{s \in \phi_p(t)} prox_p(t,s)$$

MEX applies only the mutual exclusion rule (Section 4.1.3). The representative of a colliding pattern in a sentence is randomly chosen from the tuples following that pattern in the sentence, given that we are not using proximity. This is derived from CM by setting $prox_p(t,s) \equiv 1$ and $f_p(o) \equiv 1$:

$$F_p(t) = \sum_{o \in O_p} \sum_{s \in \phi_p(t,o)} credit_p(o,s) = \sum_{s \in \phi_p(t)} credit_p(o,s)$$

4.3 Multi-Predicate Scoring

We extend our single-predicate scoring model to handle multi-predicate queries. Given a query answer, CM computes a score on each predicate. However, it remains unclear how to derive the final score, $F^A(t)$, from predicate scores.

With CM, predicate scores are unbounded, i.e., the more evidences the higher scores. When multiple predicate scores are aggregated, some could be so high that they dominate the aggregate score, which is called *predicate dominance*. To alleviate this problem, we propose Bounded Cumulative Model (**BCM**) as an alternative for scoring predicates:

$$F_p(t) = \sum_{o \in O_p} (f_p(o)[1 - \prod_{s \in \phi_p(t,o)} (1 - prox_p(t,s) credit_p(o,s))])$$

BCM uses the same three features as CM does, but differs from CM in the computation of group scores, each of which is computed from a set of evidences $\phi_p(t,o)$. Basically, BCM bounds all group scores in the range [0,1], and consequently it bounds the predicate scores within [0,1]. Note that $\sum_{o \in O_p} f_p(o)$=1 according to Section 4.1.2.

Given an answer t to query q=$\langle V, D, P \rangle$, t's final score, $F^A(t)$, is computed as the product of its scores on all predicates,

$$F^A(t) = \prod_{p \in P} F_p(t)$$

where $F_p(t)$ can be either BCM or CM. For our problem, product is a more reasonable aggregate function than summation, another common aggregate function, because it favors answers with balanced predicate scores over those with polarized ones. To illustrate why balanced scores should be

Table 2: Example Signatures in $\Phi'_{p_1}, \Phi'_{p_2}, \Phi'_{p_3}$

p_1	x	p_2	y	p_3	x	y
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
Step 1: $\Phi_{1,3}=\Phi'_{p_1} \ltimes^x \Phi'_{p_3}$={(a1 c2), (a2 c3), (a3 c5), (a4 c4)}						
Step 2: $\Phi=\Phi_{1,2,3}=\Phi_{1,3} \ltimes^y \Phi'_{p_2}$={(a1 b3 c2), (a3 b2 c5), (a4 b3 c4)}						
Step 3: $\Phi_{p_1}=\pi_x\Phi$={a1,a3,a4}, $\Phi_{p_2}=\pi_y\Phi$={b2,b4}, $\Phi_{p_3}=\pi_{(x,y)}\Phi$={c2,c4,c5}						

favored, consider the case in Table 1. The table shows four answers to the query of Query 1. For each answer, it lists all three predicate scores (by BCM), as well as the final scores using product and summation, respectively. The two aggregates agree on the ranking of t_1 and t_4, which get unanimously (i.e., balanced) high and low predicate scores, but disagree on t_2 and t_3. The true positive, t_2, gets modest and balanced scores on all the predicates. It is correctly ranked higher than t_3, a false positive, by product, but loses the comparison by summation. Answer t_3 gains high scores on p_1 and p_2 (Both are indeed satisfied by t_3.), but low score on p_3 (In reality, it does not satisfy p_3.). However, the final score of t_3 by summation is dominated by the high scoring predicates and thus t_3 is mistakenly ranked above t_2.

5 Processing SSQ Queries

In this section, we focus on how to process SSQ queries, i.e. how to retrieve evidences for all query answers (Problem 2). In practice, we solve the Entity Retrieval problem in a slight variation. Let $\Phi_p=\{\phi_p(t)|t \in A^q\}$, where A^q is the answer set. Hence, each element in Φ_p is a group of evidences. Instead of retrieving ϕ^q, we retrieve $\Phi^q=\{\Phi_p|p \in P\}$. The set Φ^q can be trivially converted to ϕ^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 for q into a series of independent entity retrieval for single-predicate query p, plus additional processing to integrate their results. By Definition 2, independent entity retrieval for "query" p is to find $\Phi^p=\{\Phi'_p\}$, where $\Phi'_p=\{\phi_p(t)|t \in A^p\}$. It can be easily derived that $\Phi_p \subseteq \Phi'_p$. If a system can process any predicate p as a single-predicate query (i.e., retrieve Φ'_p), then Φ^q can be obtained by integrating all Φ'_p following the procedure below.

Table 2 shows a toy example of Φ'_p for all predicates of Query 1. Each element in Φ'_p, which is an evidence group $\phi_p(t)$, is represented by its signature t_p. Φ'_{p_1} has five signatures, a1 to a5; Φ'_{p_2} has four, b1 to b4; and Φ'_{p_3}, c1 to c5.

Step 1 calculates $\Phi_{1,3}=\Phi'_{p_1} \ltimes^x \Phi'_{p_3}$, the join of Φ'_{p_1} and Φ'_{p_3} on x, where x is the common variable of V_{p_1}, V_{p_3}. Dick Price is a Stanford graduate (a5) but he does not found any company (no signature in Φ'_{p_3} contains him). Hence, a5 is not joinable with any element in Φ'_{p_3}. Steve Jobs founded Apple (c1) but he is not a Stanford graduate (not in Φ'_{p_1}). Hence, c1 is not joinable with any element in Φ'_{p_1}. All other PERSONs appear in both Φ'_{p_1} and Φ'_{p_3}. In the end, $\Phi_{1,3}$ contains four tuples, d1=(a1 c2), d2=(a2 c3), d3=(a3 c5), and d4=(a4 c4).

Step 2 calculates $\Phi=\Phi_{1,2,3}=\Phi_{1,3} \ltimes^y \Phi'_{p_2}$, the join of $\Phi_{1,3}$ and Φ'_{p_2} on y. b1 is not joinable because eBay does not appear in $\Phi_{1,3}$. d2 is not joinable because Google is not in Φ'_{p_2}. Eventually, $\Phi_{1,2,3}$ contains three tuples, (a1 b3 c2), (a3 b2 c5), and (a4 b3 c4).

In general, if a subset of predicates $P'=\{p_k|k=1..K\} \subseteq P$ have common variables $V_{P'}=\bigcap_{p\in P'} V_p$, the Φ'_{p_k}'s shall be joined on $V_{P'}$,

$$\ltimes^{V_{P'}}_{p\in P'} \Phi'_p = \Phi'_{p_1} \ltimes^{V_{P'}} \ldots \ltimes^{V_{P'}} \Phi'_{p_K}$$

where $V_{P'}$ is the join attribute(s) and Φ'_{p_k}'s are join inputs. A similar shortcut syntax will be used in our algorithms later. 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

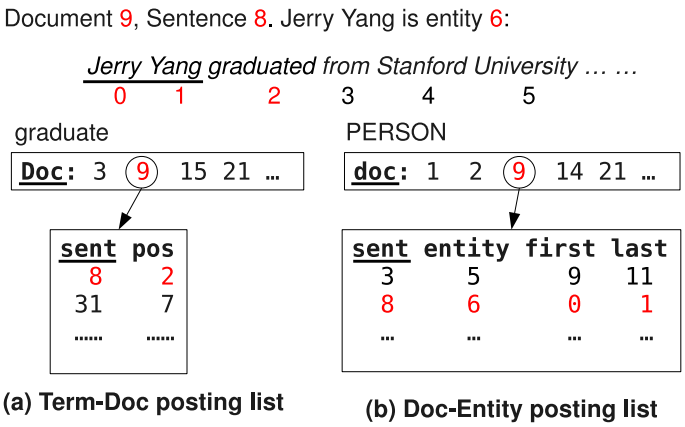


Figure 1: Document Centric Index

be performed. For brevity, we call the whole join procedure involving all such P' as *graph join* (on V), denoted by $\Phi = \otimes^V_{p \in P} \Phi'_p$.

Step 3 calculates Φ_{p_1}, Φ_{p_2}, and Φ_{p_3} by projecting (π) Φ on V_p of each p. For example, x is the only variable of V_{p_1}. Projecting Φ on x produces Φ_{p_1}=$\pi_x\Phi$={a1, a3, a4}. Similarly, Φ_{p_2}=$\pi_y\Phi$={b2, b4}, and Φ_{p_3}=$\pi_{(x,y)}\Phi$={c2, c4, c5}. Generally, for a query with predicates P, the series of projections on individual V_p's are briefly denoted by $\pi_P\Phi$.

The result of step 3 is Φ^q, the evidences of all query answers. To sum up, we present the following proposition.

Proposition 1: An SSQ query q=$\langle V, D, P\rangle$ can be evaluated in three phases: (1) process each $p \in P$ independently as a single-predicate query, obtaining Φ'_p; (2) graph join all Φ'_p on V, obtaining Φ. (3) project Φ on individual predicates. In short,

$$\Phi^q = \pi_P \otimes^V_{p \in P} \Phi'_p$$

The rest of this section studies how to process queries, particularly, the evaluation of individual predicates. As a baseline, Section 5.1 reviews entity retrieval algorithm using Document-Centric Index (DCI). Section 5.2 introduces our novel Entity-Centric Index (ECI) as an alternative. Based on ECI, Section 5.3 proposes Entity-Centric Retrieval algorithm for efficiently processing SSQ queries.

5.1 Baseline: Document-Centric Retrieval

Document-Centric Index (DCI) (or slight differently versions) is used by existing entity search systems [12, 14, 31]. 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 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 their positions.

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

Algorithm 1: Document-Centric Retrieval

Input: Query $q = \langle V, D, P \rangle$
Output: Φ^q

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

Algorithm 1 follows exactly the 3-phase processing depicted in Proposition 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 is potentially inefficient as it retrieves Φ'_p rather than Φ_p. Processing power is potentially wasted on retrieving evidences belonging to $\Phi'_p - \Phi_p$.

Example 5: Consider a query Q with two selection predicates, $p_1 = \langle \{x\}, \{\text{"Stanford"}, \text{"graduate"}\} \rangle$ and p_2=$\langle \{x\}, \{\text{"Russian"}\} \rangle$, where x 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 phase 2 (10 per person per predicate). Other evidences (10,800 in total) are trash evidences to be discarded, a huge waste 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 5.3 will show how Entity-Centric Index allows to break this limitation.

5.2 Retrieval with Entity-Centric Index

To overcome the drawback of DCI, we present Entity-Centric Index (ECI) (Figure 2), a novel index organization of the $\{term, doc, entity\}$ three-dimension space. ECI has the same number of posting

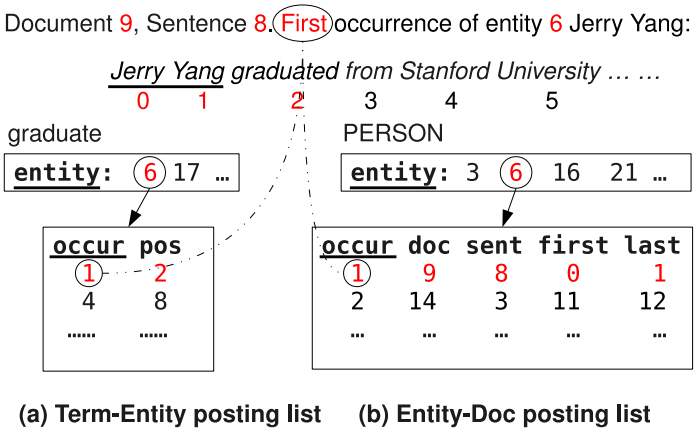


Figure 2: Entity-Centric Index

lists as DCI, 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 the 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 ($occur$=1) and position 8 of his 4th occurring sentence. 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), entities 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 multiple 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 Φ'_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 Query 1, 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 accesses 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^x=$\langle\{x\}, \{$"found"$\}\rangle$ and p_3^y=$\langle\{y\}, \{$"found"$\}\rangle$, which are evaluated as two irrelevant predicates to retrieve all their evidences, denoted as $R(p_3^x)$ and $R(p_3^y)$. 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...*

$R(p_3^x)$ will contain evidence $\langle 10, 7, \{\langle 6, 0, 1\rangle\}, \langle 3\rangle\rangle$ and $R(p_3^y)$ 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, $SP(p)$=$\{p^v|v\in V_p\}$, where p^v=$\langle\{v\}, C_p\rangle$ is a *split predicate* of p. The evidences of p, $R(p)$, can be evaluated as $R(p)$=$\bowtie_{v\in V_p}^{\langle doc,sent\rangle} R(p^v)$, where $R(p^v)$ is the evidence set for p^v.

$R(p)$ is then sorted and grouped by V_p to get Φ'_p. We trivially consider a selection predicate as a split predicate of itself. Since each Φ_{p^v} is a posting list merge join and outputs are ordered by $\langle entity, occur \rangle$, the joins $\bowtie^{ds}$ are still inefficient, which is the major drawback of ECI. Section 5.3 will show how this drawback can be relieved in case of multi-predicate queries.

So far, we have known how to evaluate arbitrary predicate individually. By Proposition 1, we can evaluate any SSQ query 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 multi-predicate queries).

Algorithm 2: Basic Entity-Centric Retrieval

Input: Query $q = \langle V, D, P \rangle$
Output: Φ^q

1 **foreach** $p = \langle V_p, C_p \rangle \in P$ **do**
2 $SP(p) \leftarrow \{p^v | v \in V_p\}$; `// Relation splitting`
3 $R(p^v) \leftarrow \emptyset, \forall p^v \in SP(p)$;
4 **foreach** $p^v \in SP(p)$ **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 $\Phi^q \leftarrow \pi_P \otimes^V_{p \in P} \Phi'_p$; `// Proposition 1`

Consider again Example 5 in Section 5.1. For p_1, suppose there are 200 persons appearing in the posting lists of both "Stanford" and "graduate", i.e., $|R^1(p^x_1)|$=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.

5.3 Entity-Centric Retrieval with Pruning

Let's first re-examine Example 5. 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 $|R^1(p^x_1)|$=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.

Algorithm 3: Entity-Centric Retrieval

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

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

In ECR, the loop over V (line 5) processes split predicates batch by batch. For $SP(v)$, EDPL of v's type is merge joined (on $entity$) with all TEPLs from all split predicates in $SP(v)$ (line 8). For each entity r^1 returned by this join, the inner loop (line 10) retrieves evidences for each split predicates $p^v \in SP(v)$ respectively. Recall that in our discussion on Example 5 at the beginning of this subsection, $|R^1(v)|$=30. For each entity in $R^1(v)$, line 11 retrieves 10 evidences for p^v_1=p_1 and 10 for p^v_2=p_2. Hence, $|R(p^v_1)|$=$|R(p^v_2)|$=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 Φ''_p. It is important to note that $\Phi_p \subseteq \Phi''_p \subseteq \Phi'_p$, due to the pruning of trash evidences.

Pruning Analysis: 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 of the two algorithms, the posting list merge join on $occur$, $\ltimes^{occur}$. This refers to line 9 of bECR and line 11 of ECR. In both algorithms, the join $\ltimes^{occur}$ should be performed for every split predicate $p^v \in SP$. The parameter that actually makes difference between the two algorithms is how many times $\ltimes^{occur}$ is executed. For a split predicate group $SP(v)$ in ECR, $R^1(v)$ is the intersection of entities in EDPL of v and TEPLs from all $p^v \in SP(v)$. Denote x^1 as the set of entities in the posting list of x,

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

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

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

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

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

$$N_{bECR} = \sum_{p^v \in SP} |R^1(p^v)|$$

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

6 Empirical Results

Our initial attempt of SSQ is a prototype system over Wikipedia. In this section, we provide experimental results on (1) ranking effectiveness of CM and BCM in comparison with other entity ranking approaches and (2) efficiency of ECR algorithm in comparison with the baseline DCR algorithm.

6.1 Prototype and Data Set

Corpus Our system building and experiments were carried out on the 2008-07-24 snapshot of Wikipedia[2]. We removed all the category pages and administrative pages, obtaining 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, they are significantly different from the main body of the article in both format and data characteristics, thus they should be treated separately by other techniques such as Information Extraction, as discussed in Section 1.

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 [3]. 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 our system. An entity can fall into multiple types. For instance, 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.

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

[3]Most Wikipedia articles belong to one or more categories that are listed at the bottom of each article.

Table 4: Compare SSQ and TextRunner(TR)

Query	1	2	3	4	5	6	7	8	9	10	11
SSQ	27	11	31	33	14	25	24	23	4	4	9
TR	13	17	0	14	7	16	2	12	2	1	6

Entity Annotations To identify occurrences of entities in the corpus, we exploit *internal links* in Wikipedia articles. An internal link is a hyperlink in some Wikipedia article to another Wikipedia article. Example 6 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). We interpret this internal link as an occurrence of the entity Cisco Systems and that the sentence uses one token, "Cisco", to reference it. Nearly 100 million annotations are identified in this way for the 0.75 million entities.

Example 6 (Internal Link): Cisco Career Certifications are IT professional certifications for [[Cisco Systems|Cisco]] products.

Query Set We use two query sets for experiments, INEX17 and OWN28. The INEX17 is adapted from topics used in Entity Ranking track of INEX 2009 [2]. There are 60 topics available in INEX. We only adapted topics about entities belonging to our predefined 10 types. A total of 17 queries are obtained, including 11 single-predicate queries and 6 multi-predicate queries (without relation predicates). OWN28 contains 28 manually designed queries, including 16 single predicate queries, 5 multi-predicate queries without relation and 7 multi-predicate queries with relation. For testing the efficiency of query processing algorithms, we draw a subset of topics from INEX17 and OWN28, extend them into more complicated queries (see Section 6.5 for detail).

6.2 SSQ vs. DB-based System

To help better understand the difference between SSQ and DB-based approach, we compare our prototype system with the state-of-the-art Open IE system, TextRunner[4]. TextRunner contains facts extracted from 500 million high-quality Web pages, which is much larger than our corpus. For the comparison, we took the 11 single-predicate queries from INEX17, converted them into TextRunner-friendly queries, and submitted those queries to TextRunner through their keyword search interface. The conversion is done to maximize recall from TextRunner. For example, if we are looking for novels by Neil Gaiman, the SSQ predicates, $\langle\{v\}, \{$"by", "Neil Gaiman"$\}\rangle$ are shortened to "Neil" "Gaiman" for TextRunner (current TextRunner does not support phrases). Table 4 compares the recall of SSQ and TextRunner on the 11 queries, showing the numbers of correct answers returned by each system.

Surprisingly, TextRunner provides much less correct answers than SSQ for most of the queries, though TextRunner extracts from a much larger corpus. However this does not mean SSQ is "better" than TextRunner. They are different approaches and have different focuses: (1)TextRunner focuses on the extraction of relations themselves, thus cannot query facts that are not extracted. SSQ rely on the users to form appropriate query predicates to "extract" at query time; (2) SSQ supports multi-predicate queries and aims at better precision at relatively large ranks instead of only top-few answers, which is not the focus of TextRunner; (3) The two systems use different corpora. Given its *extraction-based* nature, TextRunner relies on various part-of-speech patterns, noun-verb-noun patterns in particular, to extract facts. However, a large number of facts are not expressed in such patterns and thus cannot be extracted by TextRunner. For example, "American Gods, a novel by Neil Gaiman", "US Open champion Roger Federer". Meanwhile, our SSQ system avoids the pattern recognition problem by focusing on co-occurrences only.

[4]http://www.cs.washington.edu/research/textrunner/

Table 5: MAP and nDCG on INEX17/OWN28

Query	COUNT	MEX	PROX	CM	BCM	ER
nDCG on INEX17						
Single-11	0.889	0.911	0.920	0.920	0.920	0.904
Multi-6	0.880	0.918	0.932	0.954	0.958	0.927
All-17	0.886	0.913	0.924	0.932	0.933	0.912
MAP on INEX17						
Single-11	0.756	0.812	0.843	0.844	0.842	0.779
Multi-6	0.772	0.820	0.852	0.885	0.894	0.809
All-17	0.762	0.815	0.846	0.859	0.860	0.790
nDCG on OWN28						
Single-16	0.917	0.943	0.947	0.953	0.954	0.923
Multi-12	0.800	0.812	0.836	0.844	0.878	0.781
ALL-28	0.867	0.887	0.899	0.906	0.922	0.862
MAP on OWN28						
Single-16	0.758	0.825	0.838	0.858	0.853	0.760
Multi-12	0.579	0.620	0.660	0.684	0.748	0.521
ALL-28	0.681	0.738	0.762	0.783	0.808	0.658

6.3 Analyzing Alternative Ranking Methods

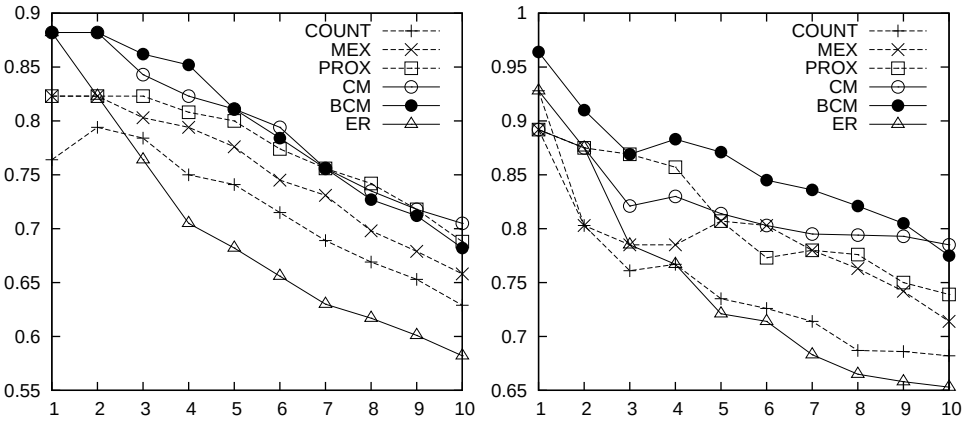
In this section, we compare and analyze the multiple ranking methods discussed earlier, namely COUNT, PROX, MEX, CM and BCM. All the methods differ in how they compute predicate scores, i.e., $F_p(t)$. For multi-predicate queries, the same aggregate function, product, is used to compute answer scores, $F^A(t)$. We compare these ranking methods using three popular measures: *nDCG*, *MAP*, and *Precision-at-*k.

nDCG (Normalized Discounted Cumulative Gain): The first block in Table 5 shows the average nDCG on single-predicate queries (Single-11), multi-predicate queries (Multi-6), and all queries (All-17) from INEX17. Both MEX and PROX improve over COUNT, by 0.02-0.05 across all three cases. PROX appears to be more effective than MEX. CM and BCM are comparable to PROX on Single-11, but further improve by more than 0.02 on Multi-6. We only observe minor difference between CM and BCM.

MAP (Mean Average Precision): The second block of Table 5 shows the MAP on INEX17. The observations are mostly similar to those from the nDCG analysis. Note that a larger distinction between CM and BCM is observed on Multi-6, with BCM about 0.01 better than CM.

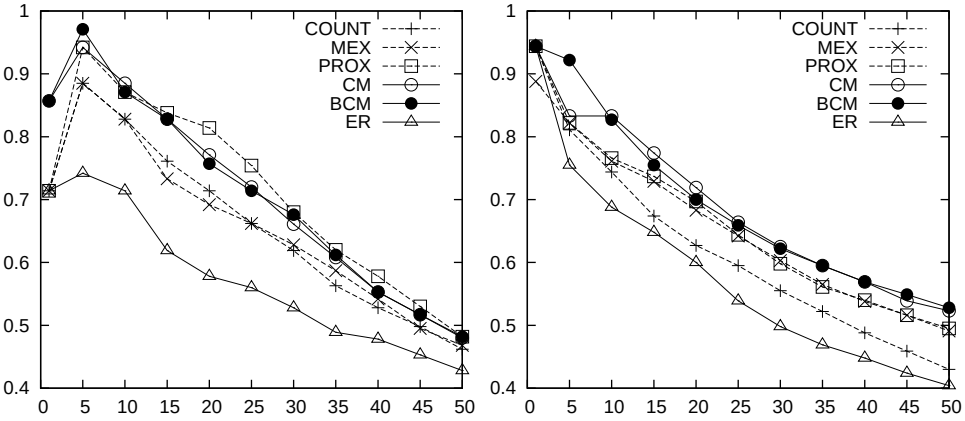
For further investigation, we repeat the above experiments on OWN28 and provide the results in the bottom half of Table 5. Most results are consistent with INEX17. However, on multi-predicate queries in OWN28 (Multi-12), BCM shows clear advantage over CM in terms of both nDCG (by 0.034) and MAP (by 0.064). The different observations on INEX17 and OWN28 is because, we believe, OWN28 has more multi-predicate queries than INEX17 and the advantage of BCM is more stably observed on OWN28.

Precision-at-k: According to the best reported MRR (Mean Reciprocal Rank) of existing entity search systems [12, 14], the first true answer is typically ranked at top 1-2. To further analyze how different methods perform in detail, especially beyond the top-few answers, we plot precision-at-k curves. Figure 3(a,b) show the results for k=10. COUNT has the worst performance. PROX is consistently better than MEX across all ranks, but worse than CM and BCM, agreeing with the conclusion drawn from nDCG and MAP analysis. BCM is consistently the best among all, while CM has inconsistent performance on INEX17 and OWN28. Figure 3(c,d) show the results for k=50. The curve for each method shows the average precision of the method at rank position k for queries that returned 50 or more answers, including 7 queries in INEX17 and 18 in OWN28. In Figure 3(c), CM and BCM excel before k=10 and BCM is slightly better. PROX is the best after k=10 but is significantly worse than BCM at top ranks. In Figure 3(d), BCM is clearly the best among all,



(a) $k = 10$ on INEX17

(b) $k = 10$ on OWN28



(c) $k = 50$ on INEX17

(d) $k = 50$ on OWN28

Figure 3: Precision-at-k on INEX17/OWN28

although a little worse than CM between 10 and 25.

In summary, the individual features are effective for entity ranking and they work best in concert when they are integrated into CM and BCM. BCM rivals CM on single-predicate queries, but excels on multi-predicate queries because BCM alleviates the predicate dominance problem. Besides, it achieves good precisions consistently across top-50.

6.4 BCM vs. Other Entity Ranking Methods

This section compares BCM with three state-of-the-art entity ranking methods: *EntityRank (ER)*, *INEX* and *INRIA*. All of these systems used Wikipedia as corpus and entity repository, though INEX and INRIA used different snapshots than ours.

EntityRank (ER) [14] focuses on single-predicate queries. It outperforms another closely related method [12] by a large margin, in term of MRR. We re-implemented ER as a plugin for scoring individual predicates in our ranking framework. The same aggregate function, product, is used to compute answer scores (F^A) for multi-predicate queries.

The detailed performance of ER is shown in Table 5 and Figure 3. In Table 5, both CM and

BCM outperform ER by large margins. The peak margin (0.22) in terms of MAP is observed on Multi-12 from OWN28, between BCM and ER. In Figure 3, ER rivals PROX, CM, and BCM at top-2, verifying the high MRR reported in [14]. However, it deteriorates very fast when $k > 2$, dropping below 0.7 around k=5, while BCM remains above 0.7 even at k=10.

INEX Entity Ranking track [2] focuses on a different problem setting. INEX queries are specified as narrative descriptions on the desired entities. Participating systems can use any techniques to answer the queries, but need to understand the query descriptions, which itself is challenging, thus their MAPs may tend to be low. The MAP achieved by the best system participating in the 2009 track is 0.517. To avoid the overhead of assessing participating systems, INEX used a sampling strategy to estimate their MAPs.

INRIA [29] works on the same problem as INEX. Unlike INEX participants, it is not based on co-occurrence of entities and query inputs. Rather, it ranks entities by link analysis and TF-IDF weighting. It achieves MAP of 0.390 on 18 topics adapted from INEX 2006 ad hoc track.

In comparison with INEX and INRIA, the MAP achieved by BCM on INEX17 is 0.860. We acknowledge that this comparison is not strictly fair. First, the results are based on different query sets (INEX17 is a subset of INEX Entity Ranking topics) and snapshots of Wikipedia. Second, they focus on different query styles (structured query vs. narrative description). However, our argument is that the high MAP of BCM at least indicates that the structured entity-relationship queries can be highly effective in reality.

In summary, our extensive analysis indicates that the proposed ranking model is very effective for ranking entities. Given that SSQ is capable of handling more complex queries with structures (which is absent from all other systems), it is a promising approach to answer entity related queries.

6.5 Efficiency of DCR and ECR

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. Our query set 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.

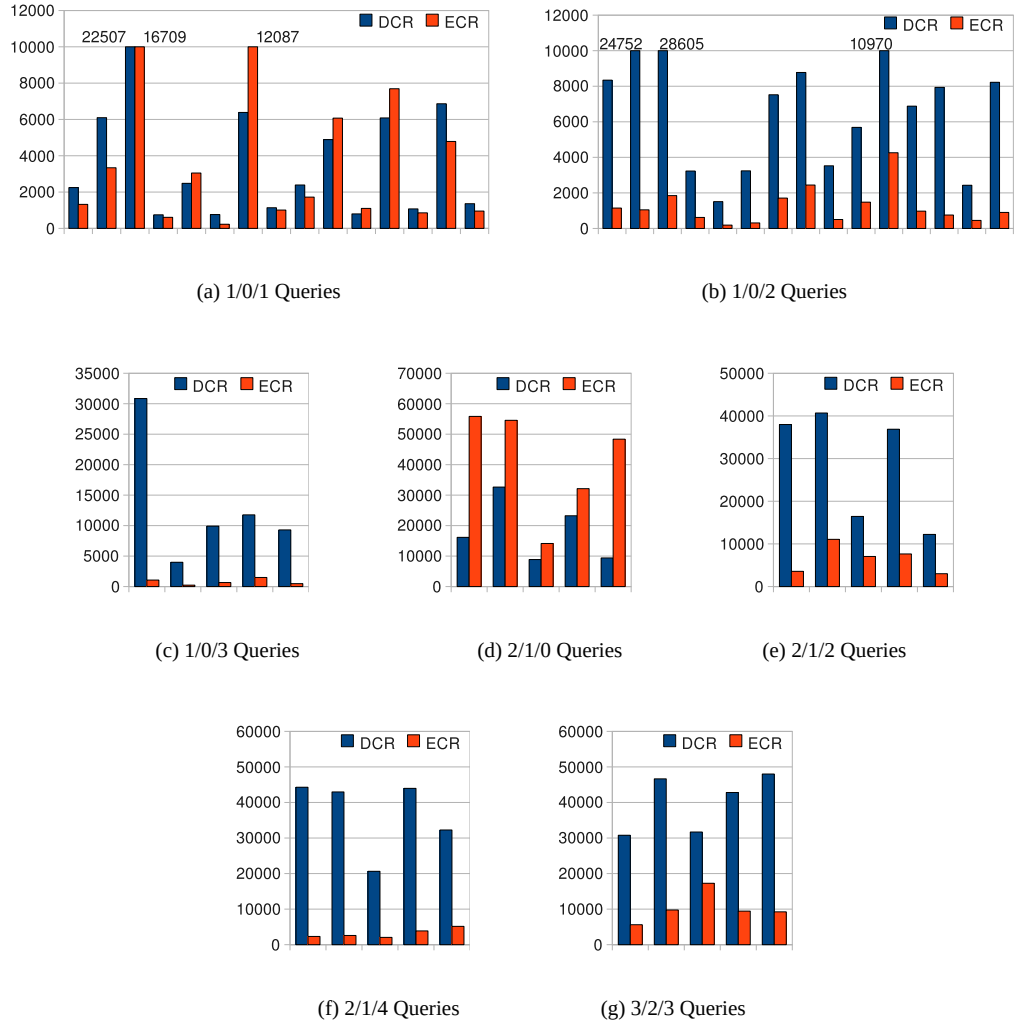
Query Group 1 (G1) contains fifteen 1/0/1 queries, fifteen 1/0/2 queries and five 1/0/3 queries, designed in the following procedure: 1) design a 1/0/3 query Q; 2) create three 1/0/2 queries by trimming one predicate off Q; 3) create three 1/0/1 queries by trimming two predicates off Q; 4) repeat steps 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 queries.

Figure 4(a)-(c) shows comparison results on G1, whose queries involving only one variable. The y-axis shows the disk I/O counts incurred by processing the queries. 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 be even reduced. For example in Figure 4(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



(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 4: Disk I/O Comparison between DCR and ECR on G1,G2 and G3

predicates' keywords. As more keywords are introduced in by additional predicates, this intersection becomes smaller and smaller, hence less entities need to retrieve evidences.

Figure 4(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 other side, DCR 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 4(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.

6.6 Efficiency on Pre-Joined Posting List

Very recently, [31] 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 [10] and used as contextual posting list in [31]). However, completely building all advanced posting lists are too huge to afford. Hence, [31] 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 report takes a simple approach while leaving a 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. The new algorithms are referred to as aDCR and aECR respectively.

The comparison result (Figure 5) 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 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 bare chance to show 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.

7 Conclusion

In this report, we introduced a novel querying mechanism, Shallow Semantic Query, which enables users to issue structured entity-centric queries over textual content and obtain direct answers. We thoroughly discussed two key issues in developing a quality SSQ system, ranking and query processing. Although our current study on SSQ is still at its early age, experiments already indicate that it is a competing approach towards a general solution to entity-centric information needs. We look forward to more in-depth studies in future.

References

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

[2] INEX 2009 entity-ranking track. `http://www.l3s.de/~demartini/XER09/`.

[3] TREC 2009 entity track: Searching for entities and properties of entities. `http://ilps.science.uva.nl/trec-entity/guidelines/`.

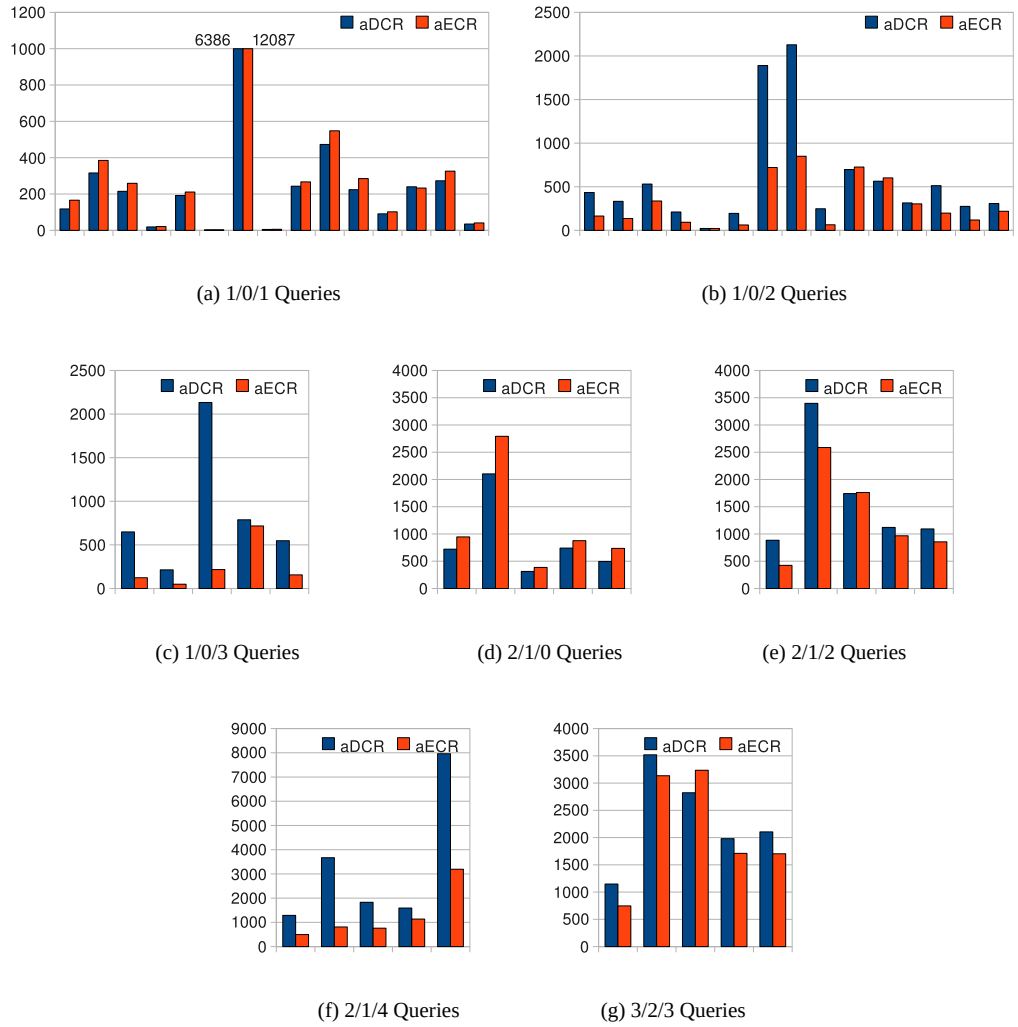


Figure 5: Disk I/O Comparison between aDCR and aECR on G1,G2 and G3

[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] R. C. Bunescu and M. Pasca. Using encyclopedic knowledge for named entity disambiguation. In *EACL*. The Association for Computer Linguistics, 2006.

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

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

[12] 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.

[13] 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.

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

[15] 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.

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

[17] S. Cucerzan. Large-Scale Named Entity Disambiguation Based on Wikipedia Data. *EMNLP*, 2007.

[18] 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.

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

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

[21] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Unsupervised named-entity extraction from the web: an experimental study. *Artif. Intell.*, 165(1):91–134, 2005.

[22] 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.

[23] H. Garcia-Molina. Entity resolution: Overview and challenges. pages 1–2. 2004.

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

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

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

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

[28] 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.

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

[30] 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.

[31] 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.