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 the paradigm of keyword search because of its flexibility. Such a paradigm is limited in two main aspects. First, keyword queries are suitable for document retrieval rather than entity retrieval, i.e., search results are presented as pointers to Web pages. Users have to digest those pages to extract desired information themselves. Second, keyword searches are limited to simple information needs. Adding too many keywords in a query often results in non-relevant pages or no page at all. Being able to allow the users to issue complex structural queries and retrieve entities without reading any pages can add significant value to the search system. In this paper, we propose a novel entity-centric structure-focused search mechanism called Shallow Semantic Query (SSQ). We propose a ranking model for SSQ queries based on a set of shallow syntax features. We also develop a novel entity-centric index and design efficient query processing strategies for evaluating SSQ queries. Our initial attempt of SSQ is a prototype system over `Wikipedia` and is accessible at http://idir.uta.edu/ssq. Our comparison with TextRunner and other entity ranking models indicates that SSQ is a promising approach for entity-centric queries and that our ranking model is effective, achieving nDCG of 0.933 and MAP of 0.860 for INEX queries. We also conducted experiments to evaluate the efficiency of the proposed indexing and query processing approach and demonstrate its advantages over several other alternatives.

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. In discovering and exploring the entities that fascinate them, the 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, and one of the major directions that database research is expanding toward is the interplay between structure and text [28]. Recently there have been extensive efforts along this general direction [12, 22, 7].

Despite the increasing popularity of structured information on the Web, the prevalent manner in which the users access such information is still keyword 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. Users' 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 the list of companies located in Silicon Valley.

Task 2: Find the list of 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, in database terminology, their "join" relationships. Second, our tasks often involve synthesizing information scattered across different places, therefore a simple list of articles 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 various 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 "graduate Stanford" to find a list of people graduated from Stanford University. She then manually joins these two lists and, by multiple additional searches, check if a company was founded by a person, for each joined 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 time-consuming and error-prone iterative steps of searching, reading and re-searching.

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 only a set of keywords. There are three basic concepts within an 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 is bound 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 *join predicate* specifies the relation between p and c (c was founded by p).

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 prototype SSQ system is accessible at **http://idir.uta.edu/ssq**. We take a DB-IR integration approach in pursing SSQ. On the one hand, SSQ queries have explicit structured components– typed entity variables and selection/join predicates, offering greater expressive power than pure keyword queries by the entity-centric and structure-focused view; On the other hand, the individual predicate is specified by keyword-based constraints, resembling keyword search. In fact, the SSQ system finds entities satisfying a predicate by a simple and intuitive requirement– the entities co-occur with the keywords in some contexts, sharing the same intuition adopted in traditional search queries. For example, predicate p : ["Stanford" "graduate"] requires every p appearing in the query answer co-occurs with "Stanford" and "graduate". In other words, SSQ aims to capture entity properties and relationships through shallow syntax requirements implied by users at query-time, instead of explicitly extracting and reasoning about complex semantic information in text before query-time. Thus the name *shallow semantic queries*.

Previous works on structured querying of the Web mainly fall into three categories. (1) The *DB-oriented* approach explicitly extracts structured information into relational databases, thus supporting SQL-style queries [6, 4, 14, 20, 9, 10, 24, 17]. This approach can be limited because the extracted information must conform to prescribed schemata that are often application-specific [6, 4, 14, 24, 17]. It is also constrained by the capability of the information extraction (IE) and natural language processing (NLP) techniques in extracting attributes and relationships of entities [20, 9, 10]. Typical IE techniques [26, 15, 19] define or learn application-specific extraction patterns, thus could not be applied for arbitrary relations. Open IE [20] breaks the limitation by exploiting Part-of-Speech patterns. In particular, it extracts 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*. (2) The *Semantic Web-oriented* approach [29, 25, 18, 5] represents data in RDF [1], the W3C recommendation of data model for Semantic Web, and exploits the data by RDF search and query methods. Some publish and export various open data repositories as RDF and interlinking them [5]. These RDF exports only capture the explicit structured information (such as "infoboxes" and internal links in Wikipedia), leaving out the implicit structures in text. Some extract RDF from Web pages to bootstrap the Semantic Web [18, 29]. However, similar to the DB-oriented approach, such extraction requires the identification of the "names" for entity attributes and relationships, to be represented as the "property" in the triple form "<subject, property, object>" of RDF statements. The satisfaction of this requirement relies on the domain-specific knowledge during extraction and/or the generality of the IE and NLP methods. (3) The *IR-oriented* approach, exemplified by the recently formed entity search and ranking problems in the IR community [11, 27, 2, 3, 31, 30], focus on finding named entities relevant to certain contextual information in natural language text or related to a given entity. They often come with type constraints on the entities, similar to SSQ queries. However, they do not have the concept of "predicates" and do not deal with multiple predicates or join relationships between entities.

Developing SSQ presents a significant research challenge and involves several important building pieces. Named entity resolution, disambiguation, recognition, and categorization are required for properly identifying the 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 (e.g., [23, 16, 21, 8]). 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, namely Wikipedia.

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 to ease the aforementioned problems (details in Section 5.1) and thus allow us to focus on the central challenges of SSQ itself and not to be distracted. Moreover, the high-impact that Wikipedia has on our society makes a SSQ system over Wikipedia itself a valuable artifact. It is also our hope that the results from this paper would lead to the thorough investigation of SSQ over generic Web pages.

Challenges: With the structured annotations in Wikipedia alleviating the peripheral problems such as entity detection, we address several key challenges in SSQ. *First*, the notion of Shallow Semantic Query and the semantics of query results must be clearly defined. Ideally, it should meet two requirements. On the one hand, it should be flexible enough to address complex inter-relationships among entities. On the other hand, the query should be as easy to specify as possible. *Second*, an effective ranking mechanism has to be established. Ranking models that are typical in document retrieval systems, for example PageRank and Vector Space Model, do not directly apply to ranking SSQ search results. The relevance of an entity to an SSQ query has little to do with the relevance of the document from which the entity is found. An article on programming language may well mention that some company is in Silicon Valley, which is an answer to Task 1. *Third*, as a search system involving user interaction, how to efficiently retrieve entities and identify their relations is an important issue to be addressed.

Contributions: This paper makes the following contributions:

- We introduce Shallow Semantic Query, a entity-centric structure-focused mechanism for querying entity properties and relationships, and formally define its semantics (Section 2).
- We present a framework for ranking SSQ query results (Section 2). Following this framework, we propose a ranking method based on a group of shallow syntax features that center on entity-keyword co-occurrence (Section 3).
- We propose a novel entity-centric index and design efficient query processing strategies for evaluating SSQ queries (Section 4).
- We implemented a prototype SSQ system (**http://idir.uta.edu/ssq**) and conducted comprehensive experiments to demonstrate the effectiveness of our ranking method and the efficiency of the indexing and query evaluation approach (Section 5).

2. 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 sematic criteria in selecting and joining entities. Formally:

Definition 1 (Shallow Semantic Query): Given a set of entity types $\mathcal{D}$, 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 $|V_p|=1$, otherwise a *join predicate*.

– $U \subseteq V$ represents the set of entity variables that constitute the output tuple. ■

By the above definition, in Query 1 $\langle V_q, D_q, P_q, U_q\rangle$, $V_q = U_q = \{$p, c$\}$, $D_q = \{$PERSON, COMPANY$\}$, $P_q = \{p_1, p_2, p_3\}$, where $p_1 = \langle\{$p$\}$, $\{$"Stanford", "graduate"$\}\rangle$, $p_2 = \langle\{$c$\}$, $\{$"Silicon Valley"$\}\rangle$, and $p_3 = \langle\{$p, c$\}$, $\{$"found"$\}\rangle$. Among the predicates, p_1 and p_2 are selection predicates, and p_3 is a join predicate.

Note that U is a subset of V, resembling the notion of projection in relational algebra. For example, suppose $\langle$<u>Jerry Yang</u>, <u>Yahoo!</u>$\rangle$ and $\langle$<u>David Filo</u>, <u>Yahoo!</u>$\rangle$ are both answers to Query 1. If c is the only output variable, only one $\langle$<u>Yahoo!</u>$\rangle$ 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 a SQL-like syntax to express SSQ queries, where the `SELECT`, `FROM` and `WHERE` clauses correspond to 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 evidence.

Definition 2 (SSQ Answer Tuple): Given a query $q = \langle V, D, P\rangle$, the query 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]

– For any $\langle V_p, C_p\rangle \in P$, an *evidence* of t with regard to p is a textual context s containing (1) all the entities in $t.V_p$ (defined as the projection of t containing only those entities that are instantiations of variables in V_p) and (2) all the phrases in C_p. The context s is a unit piece of textual content and can be custom defined.

– 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 the evidences can be sentences, paragraphs, documents, windows of words, etc. Different textual contexts have different retrieval capabilities and thus result in different answer tuples. While a comparative study of the different contexts is left for future investigation, in this paper, we 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 Section 5.1.

Definition 3 (SSQ Ranking Result): Given a query $q = \langle V, D, P\rangle$, we introduce three scoring functions F_S, F_J and F_A, such that the ranking result of q is a ranked list of $\langle$tuple, score$\rangle$ pairs $\langle t, r\rangle$, 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 join predicate $p = \langle V_p = \{v_{k_1}, \ldots, v_{k_i}\}, C_p\rangle \in P$ similarly has a computed score $p^{F_J}(t) > 0$.

– The tuple score r 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_J. ■

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_J evaluates each join predicate with regard to how well the relation among entities holds true. Finally at the query level, F_A evaluates how much a tuple of entities satisfy the predicates altogether, based on the scores of individual predicates. The tuples are ranked by their query level scores r.

As an example, suppose $t_1 = \langle$<u>Jerry Yang</u>, <u>Yahoo!</u>$\rangle$ 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, $t_2 = \langle$<u>Jerry Yang</u>, <u>Google</u>$\rangle$ 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 3:

– 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. Section 3 presents a specific ranking model under this framework.

– 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.

– The score on a predicate reflects the degree of satisfaction and is dependent on the particular ranking model.

3. SSQ RANKING

Coupled with structural query specification, SSQ at its core queries by keyword-based constraints over unstructured text content. Its flexibility in specifying keyword-based predicates brings two main challenges in ranking. *First*, the keyword-implied query semantics must be captured in ranking. Since SSQ does not maintain any semantic knowledge encoded for machine understanding (except for entities and their types), there is a mismatch between the semantic input/output and the underlying textual data. How to rank semantically based on textual data needs to be resolved. *Second*, by nature SSQ should maintain a high precision over long range of its search results. Most existing works on Web search and entity retrieval focus on *factoid* queries, where users look for a few true answers. Examples are searching for the home page of US government and finding out who founded Google. In such queries, it is sufficient to achieve high precision at top ranks. Shallow Semantic Query, however, extends the scope to *list* queries where users may expect a large volume of true answers. For example, a query for all US Open champions would expect more than 200 true answers. While it could be relatively easy to achieve high precision at top ranks, maintaining good precision over long range is more difficult.

The rest of this section is divided into three parts. First, we introduce the shallow syntax features that are building blocks of our ranking method. Then, we present our predicate scoring model by integrating these features. Finally, we discuss how to aggregate predicate scores to produce the query-level score.

3.1 Shallow Syntax Features

With regard to a single predicate, a straightforward way of scoring tuples is to count their supporting evidences. Intuitively, a tuple with more evidences is more likely to satisfy the predicate. However, during our investigation, we observe that for most predicates, the evidence counts follow a "long-tail" distribution, which makes it harder to distinguish true answers from false positives. That is, while some tuples have distinctively more evidences than others, many more tuples that are true answers have small numbers of evidences. In the meantime, there are tuples that happen to appear in some false evidences. Hence, purely using the number of evidences to score tuples is only effective in finding answers that are popular in the corpus, but cannot easily tell apart true positives and false positives with small numbers of evidences. Therefore, the core task of our scoring method is to evaluate the validity of an evidence s of an entity tuple t with regard to a predicate p. Towards this goal, we exploit several shallow features of evidences. Note that we only consider sentences as evidences, as explained in Section 2, therefore these features are particular to sentences.

3.1.1 Proximity

Intuitively, if the entities in $t.V_p$ and the predicate phrases in C_p are close to each other, there is a good chance that they belong to the same grammatical structure and thus form a valid evidence. Given an evidence (sentence) s of entity tuple t with regard to predicate $p = \langle V_p, C_p \rangle$, the proximity of t is defined as

$$prox(t,p,s) = \frac{\sum_{e \in t.V_p} |token(e,s)| + \sum_{c \in C_p} |c|}{|scope(t,p,s)|}$$

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

Different representations may be used in various places to refer to the same entity. These representations thus may have different number of tokens. For example, the company IBM may be represented by "IBM", "Big Blue", or "International Business Machine", with 1, 2, and 3 tokens, respectively. In the above formula, we use $token(e,s)$ to refer to the tokens in s representing entity e. (See Section 5.1 for more details about entity identification.)

Example 2: The following two sentences are both evidences of the underlined entities with regard to predicate p : ["Stanford" "graduate"] in Query 1. Evidence 1 is a valid evidence, supporting a true positive, while Evidence 2 is invalid, supporting a false positive.

Evidence 1: *In 1960, while in residence at the Sri Aurobindo Ashram Cultural Integration Fellowship in San Francisco, he met a fellow Stanford University graduate, Dick Price.*

Evidence 2: *Two years after he graduated from Stanford University, he married Barbra Ann Briggs, whose father was Stephen Foster Briggs of Briggs and Stratton.*

The predicate p has two phrases, "Stanford" and "graduate", each with one token, hence $\sum_{c \in C_p} |c|$=2. In Evidence 1, the projection of tuple t over p, $t.V_p$, has one entity which is represented by two tokens "Dick" and "Price", hence $\sum_{e \in t.V_p} |token(e,s)|$=2. The scope covering all entities and phrases spans 5 tokens from "Stanford" to "Price", thus $|scope(t,p,s)|$=5. Therefore, the proximity of Dick Price in Evidence 1 is $prox(t,p,s)$=$\frac{2+2}{5}$=0.8, and the proximity of Barbra Ann Briggs in Evidence 2 is $\frac{2+3}{9}$=0.56. Based on proximity alone, we say that Evidence 1 is a more valid evidence. Consequently, Dick Price is more likely to be a Stanford graduate than Barbra Ann Briggs is, given no other evidence. ■

3.1.2 Ordering Pattern

An ordering pattern refers to a particular order of entities and phrases appearing in an evidence. Take as an example the predicate p : ["Stanford" "graduate"]. Let c_1 stand for "Stanford", c_2 for "graduate", and e for any entity instantiated from p, i.e., of type PERSON. This predicate has six different ordering patterns, namely ec_1c_2, ec_2c_1, c_1ec_2, c_2ec_1, c_1c_2e, and c_2c_1e. In general, for a predicate p=$\langle V_p, C_p \rangle$, there are ($|V_p|$+$|C_p|$)! possible ordering patterns in total. Note that, ordering pattern only refers to the appearing order of entities and phrases. There could be other tokens between them and punctuation marks are ignored. Therefore, *Stanford University graduate, Dick Price* and *Stanford graduate Dick Price* follow the same ordering pattern c_1c_2e.

We observe that some ordering patterns are better indicators of valid evidences than others. For example, the ordering pattern in Evidence 1 (c_1c_2e, i.e., *Stanford graduate somebody*), can be commonly used to express that somebody is a graduate of Stanford University, while the pattern in Evidence 2 (c_2c_1e, i.e., *graduate Stanford somebody*) may not be as common in expressing such knowledge. Yet another pattern, c_1ec_2, is seldom used.

To distinguish the stronger ordering patterns (those that tend to indicate true evidences) from the weaker ones, we may assign a different weight to each pattern, so that tuples with evidences in strong patterns are scored higher. However, it is impossible to pre-determine the weights, since the goodness of an ordering pattern is predicate-dependent. For example, c_1c_2e is a strong pattern for predicate p: ["Stanford" "graduate"], but may not be equally good for n: ["by" "Jane Austen"] (n is of type NOVEL) because it is uncommon to see an evidence such as *written by Jane Austen, Pride and Prejudice*. Note that stop words (e.g., "by") are not ignored from SSQ query predicates.

The SSQ system automatically derives the weights of ordering patterns for each predicate, given the above-mentioned predicate-dependency nature of the patterns. In particular, given predicate p, we use the frequency $f(o,p)$ of each pattern o as its weight:

$$f(o,p) = |S(o,p)|/|S(p)|$$

where $S(p)$ is the set of all evidences retrieved for p and $S(o,p) \subseteq S(p)$ is the set of evidences in which pattern o exists. This definition assumes that the strong patterns appear more often than the weak ones. Although in theory there might be such a case that many invalid evidences happen to follow the same pattern, making a bad pattern more common, in practice we did not observe any violation of the assumption throughout our experiments.

3.1.3 Mutual Exclusion

Given a predicate p=$\langle V_p, C_p \rangle$, a sentence may contain the phrases in the constraint C_p and multiple entity tuples instantiated from the variables in V_p. Therefore the same sentence may potentially be considered multiple evidences with regard to p, each for a different tuple. These evidences may exhibit different ordering patterns of tuples and phrases. The co-existence of different ordering patterns is called a *collision* and the patterns are called *colliding patterns*. The mutual exclusion rule dictates that, when collision happens, at most one colliding pattern is effective and the sentence is only considered evidence for tuples following that effective pattern.

Example 3: Evidence 3 demonstrates the mutual exclusion rule for predicate p : ["Stanford" "graduate"]. This sentence is an evidence for three entities. Ric Weiland follows the pattern o_1=ec_2c_1. Paul Allan and Bill Gates follow o_2=c_2c_1e. In this sentence, only the former pattern is the effective pattern, thus it is only a valid evidence for Ric Weiland.

Evidence 3: *After Ric Weiland graduated from Stanford University, Paul Allen and Bill Gates hired him in 1975, the same year they founded Microsoft in Albuquerque.* ■

In general, it is difficult to decide which colliding pattern is absolutely effective. Therefore, we soften 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 the ones with lower credits. We assume each sentence s (that is an evidence for at least one tuple) has a total credit of 1, meaning that there is only one effective pattern. Given a predicate p, for a sentence s with colliding patterns $O(p, s)$, each colliding pattern $o{\in}O(p, s)$ gets a credit $credit(o, p, s)$, and the credits of all the colliding patterns sum up to the total credit of s, i.e., $\sum_{o\in O(p,s)} credit(o, p, s)$=1. For a sentence without collision (i.e., only one ordering pattern), the single pattern gets the total credit 1.

To allocate credits to the colliding patterns in a sentence, we adopt the intuition that patterns followed by more prominent entities are more likely to be effective. Specifically, given an evidence s for predicate p, $T(o, p, s)$ denotes the set of tuples appearing in s that follow pattern o. (Note that the tuples are projected on V_p). For each $o{\in}O(p, s)$, we choose one representative tuple from $T(o, p, s)$, denoted by $t(o, p, s)$, which is the one with the highest proximity value, i.e., $t(o, p, s)$=arg $\max_{t\in T(o,p,s)} prox(t, p, s)$. We compare these representatives, thus the patterns that they follow, by how prominent they are, i.e., by their overall numbers of evidences in the corpus. The credit of o in sentence s is

$$credit(o, p, s) = \frac{|S(t(o, p, s))|}{\sum_{o'\in O(p,s)} |S(t(o', p, s))|}$$

where $S(t(o, p, s))$ is the set of evidences from which $t(o, p, s)$ is found. Note that we choose the most proximate tuple as the representative of a colliding pattern and allocate credits based on representatives only. The intuition is that the most proximate tuple is most likely to form a grammatical structure with the phrases in the predicate, hence is the most reliable for allocating credits.

In Evidence 3, $t(o_1, p, s)$=Ric Weiland, i.e., Ric Weiland is the representative of patterns o_1, since it is the only entity following o_1 in the sentence. $t(o_2, p, s)$=Paul Allen because it has higher proximity (0.67) than Bill Gates (0.44), although both follow pattern o_2. Suppose Ric Weiland is found in 4 evidences and Paul Allen in 2 evidences (both the 4 evidences and the 2 evidences include Evidence 3), i.e., $|S(t(o_1, p, s))|$=4 and $|S(t(o_2, p, s))|$=2. Then $credit(o_1, p, s)$=4/(4+2)=0.67 and $credit(o_2, p, s)$ = 2/(4+2) = 0.33.

Note that the pattern credit here is different from the weight of pattern introduced in Section 3.1.2. The weight of pattern o is a global measure of how often, and thus how reliable, pattern o is, aggregating over all the evidences. The credit of o, on the contrary, is a local measure particular to each evidence s, indicating how likely o is the effective pattern in s.

3.2 Predicate Scoring

So far, we have introduced all the features for evaluating the validity of evidences. It remains a challenge to synthesize them in an integral scoring model, i.e., to define F_S and F_J in Definition 3. Here we address the issue.

Given a predicate p=$\langle V_p, C_p\rangle$, the set of all possible ordering patterns is O, which contains $(|V_p|$+$|C_p|)!$ patterns as mentioned in Section 3.1.2. Let $S(t, o, p)$ be the group of evidences for tuple t with regard to p, in which $t.V_p$ and C_p follow pattern o. Our *cumulative* predicate scoring model (denoted as CM) for both F_S and F_J is uniformly defined as

$$p(t) = \sum_{o\in O}(f(o, p) \sum_{s\in S(t,o,p)} prox(t, p, s)credit(o, p, s))$$

where $f(o, p)$ is the weight of pattern o; $prox(t, p, s)$ is proximity of t in evidence s; and $credit(o, p, s)$ is the credit of o in s.

The model divides $S(t, p)$, t's evidences with regard to p, into $|O|$ groups, $\{S(t, o, p)|o \in O\}$, so that evidences in one group follow the same pattern. For each $o{\in}O$, the model computes a *group score* (the inner summation) for the evidence group $S(t, o, p)$. Group scores are then combined linearly using $f(o, p)$ as the weights (the outer summation), such that the group scores of strong patterns account more in the predicate score $p(t)$. The kernel of the scoring function, $prox(t, p, s)credit(o, p, s)$, evaluates the validity of s being an evidence of t. It is monotonic to both the proximity of t and the credit of t's pattern o. Tuples with evidences that have higher proximities and pattern credits will thus accumulate higher scores on the predicate p.

It is interesting to note that the aforementioned cumulative scoring model (CM) can be customized easily by switching on and off the shallow syntax features in various combinations, so that we can evaluate the effectiveness of individual features. While the detailed evaluation is presented in Section 5, below we briefly describe several customizations. Note that all the methods differ only in how they compute predicate level scores, and they use the same aggregate function to compute query level score.

• COUNT: This is the straightforward baseline method that scores a tuple t by its number of supporting evidences with regard to p, i.e., $p(t)$=$|S(t, p)|$. It can be reduced from the CM model by turning off all the features, i.e., by setting $prox(t, p, s) \equiv 1$, $credit(o, p, s) \equiv 1$, and $f(o, p) \equiv 1$:
$p(t) = \sum_{o\in O} (1 \sum_{s\in S(t,o,p)} 1) = \sum_{o\in O} |S(t, o, p)| = |S(t, p)|$

• PROX: It applies only the proximity feature (Section 3.1.1) and is reduced from CM by $credit(o, p, s) \equiv 1$ and $f(o, p) \equiv 1$:
$p(t) = \sum_{o\in O} \sum_{s\in S(t,o,p)} prox(t, p, s) = \sum_{s\in S(t,p)} prox(t, p, s)$

• MEX: It applies only the mutual exclusion rule (Section 3.1.3). 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 $prox(t, p, s) \equiv 1$ and $f(o, p) \equiv 1$:
$p(t) = \sum_{o\in O} \sum_{s\in S(t,o,p)} credit(o, p, s)$

By the CM model, a tuple's score on a predicate is not bounded. When multiple predicate scores are aggregated at the query level, the score on one predicate could be high enough to dominate the query-level score. To alleviate this problem, we propose an alternative predicate scoring model, *bounded cumulative* model (denoted as BCM), as follows:

$$p(t) = \sum_{o\in O}(f(o, p)[1- \prod_{s\in S(t,o,p)} (1 - prox(t, p, s)credit(o, p, s))])$$

It differs from CM in the computation of group scores. Basically,

Table 1: Example List of Answers

	PERSON p	COMPANY c	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 Inc.	0.6	0.5	0.6	0.18	1.7
t_3	Scott McNealy	Cisco Systems	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

it bounds a group score within [0,1], and consequently predicate score within [0,1]. (Note that all the $f(o,p)$ will sum up to 1, according to Section 3.1.2.)

3.3 Query-Level Ranking

After an entity tuple t has been evaluated on all the predicates, a final query-level judgment of t is made by combining the predicate scores (F_A in Definition 3). In this paper, we assume every predicate, either a selection or a join predicate, is equally important and use product to aggregate the predicate scores.

$$F_A(\{p(t)|p \in P\}) = \prod_{p \in P} p(t)$$

Product as the query-level aggregate function F_A makes better sense for our problem than summation, which is another commonly used aggregate scoring function, because it favors answers with balanced predicate scores over those with polarized scores. To illustrate why balanced scores should be favored, consider the example in Table 1. The table shows four answer tuples to Query 1. For each tuple, it lists the scores on all three predicates, as well as the query-level 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. Answer t_2, a true positive gets modest and balanced scores on all the predicates. It is correctly ranked higher than t_3, a false positive, by product, but loses its position by summation. t_3 gains high scores on p_1 and p_2 (Both predicates are indeed satisfied by t_3.), but low score on p_3 (It in fact does not satisfy p_3 in reality.). However, the query-level score of t_3 by summation is dominated by the high scoring predicates and thus mistakenly ranks it above t_2.

4. SSQ QUERY PROCESSING

While the importance of retrieving high quality results is important, it is equally important to do it efficiently. As part of this work, we examined various index structures and query processing strategies for entity retrieval. And based on those analyses, we propose a novel entity-centric index for the efficient retrieval of entities.

4.1 Entity Retrieval

In SSQ, the entity retrieval module provides the ranking module with the entities along with the supporting evidences of these entities. The ranking module relies on these evidences to make ranking decision as discussed in Section 3. Given an SSQ query q=$\langle V, D, P\rangle$, an evidence of tuple t with regard to predicate $p \in P$ is a quadruple $\phi(t,p)$=$\langle \widehat{t.V_p}, doc, sent, \widehat{C_p}\rangle$, where:

– doc and $sent$ refer to the document ID and sentence number uniquely identifying a sentence ϕ in corpus,

– Each $\hat{e}_i \in \widehat{t.V_p}$ is a triple, $\hat{e}_i$=$\langle e_i, f_i, l_i\rangle$, where 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.

The evidence representation above encodes the minimally required information to support SSQ ranking. The task of entity retrieval is to retrieve all evidences of every SSQ answer tuple (Definition 2) with regard to every predicate. Formally,

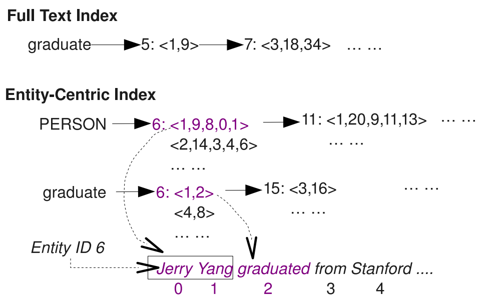

Figure 1: Entity-Centric Index

Definition 4 (Entity Retrieval): Given an SSQ query q=$\langle V, D, P\rangle$, the task of entity retrieval is to find the set of evidences Φ=$\{\phi(t,p)|t \in A(q), p \in P\}$, where $A(q)$ is the set of SSQ answer tuples. ∎

4.2 Entity-Centric Index

Traditional full text index creates one posting list for every unique term, encoding all the documents and positions where the term appears. In the example full text index in Figure 1, "graduate" occurs 2 times in document 5, at positions 1 and 9 respectively. This index organization maps keywords to documents and is therefore particularly suited for document retrieval. However, it is ill-suited for the task of entity retrieval, where the outputs are entities and their evidences instead of documents.

Our entity-centric index is analogous to full text index. Figure 1 shows the conceptual structure of entity-centric index. It consists of two components, type-entity and word-entity. The **type-entity (TE)** component is a set of posting lists, one for each entity type. The entries of a TE posting list are sequence of entity IDs (in ascending order) of that type. For every entity, it further stores all its occurrences. For example, in Figure 1, entity 6, Jerry Yang, is a PERSON, its first occurrence is from position 0 to 1 of sentence 8 in document 9 and second occurrence spans position 4 through 6 of sentence 3 in document 14. The **word-entity (WE)** component is a set of posting lists created for every unique word in corpus. Again, the entries of a WE posting list is an ascending sequence of entity IDs. An entity e is present in the posting list of word w if and only if there is at least one sentence in which they co-occur. A WE posting list further stores the co-occurrence information for every entry. In Figure 1, Jerry Yang co-occurs with "graduate" in its first occurrence (sentence 8 in document 9) and "graduate" is at position 2 of that sentence.

In summary, TE allows easy access to entities and their occurrences, while WE indexes co-occurrences between words and entities. Together, they form our entity-centric index.

4.3 Query Processing

We describe how the entity-centric index can be used for query processing in Algorithm 1. Given a selection predicate p=$\langle\{v\}, C_p\rangle$, the algorithm takes the TE posting list of v's type and the WE posting lists for all keywords in C_p. It then performs a two-level multiway merge join of these posting lists. At the entity level, it finds all entities that belong to the given type and also co-occur with every keywords (not necessarily co-occurring with all keywords in the same sentence). Then, at occurrence level, the algorithm finds all sentences in which the entity indeed co-occurs with all keywords. Each of the sentence joined at the occurrence level is returned as an evidence. As a simple example, Figure 1 shows that the two posting lists, PERSON and "graduate", can be joined (at entity level) on entity 6 at its first occurrence (occurrence level). Processing join predicate follows the general two-level merge join processing style. Due to space limit, we omit the details in this paper.

Algorithm 1: Processing Selection Predicate

```
Input: Predicate $p = \langle\{v\}, C_p\rangle$
Output: $\Phi_p$ - Evidences with regard to $p$
$d \leftarrow$typeOf($v$);
$L \leftarrow$\{PostingList($d$)\};
foreach $w \in C_p$ do
    $L \leftarrow L\cup$\{PostingList($w$)\};
while succefully join $L$ on some entity $e$ do
    while succeffully join $L$ on occurrence $\phi$ do
        $\Phi_p \leftarrow \Phi_p \cup \{\phi\}$;
```

To process a multi-predicate SSQ query, one naive strategy is to run Algorithm 1 for each predicate independently. The query processing time of this basic strategy grows linearly to the number of predicates. However, entity-centric index lends us an optimized strategy in this case. To simplify the discussion, we focus on two selection predicates on the same entity variable, where each predicate contains only one keyword. I.e., the query involves one type and two keywords in total. Algorithm 2 performs only one entity level join on all three posting lists. For each successfully joined entity, two occurrence level joins are performed, one between type d and word w_1, another between d and w_2, i.e., performing occurrence level join for each predicate separately.

This optimized processing strategy improves efficiency in two ways. First, it reduces the number of times a posting list is traversed. Using basic strategy, PostingList(d) needs to be traversed twice, once in L_1 another in L_2. Using the optimized strategy, it is only traversed once in L. Second, it reduces the number of occurrence level join. Let E_1 be the set of entities that can be joined from L_1, E_2 be the set of entities joined from L_2. The basic strategy will perform occurrence level join for every entity in E_1 and E_2, a total of $|E_1| + |E_2|$ joins, while the optimized strategy only performs occurrence level join for entities in $E = E_1 \cap E_2$.

Algorithm 2: Optimized Processing of Multiple Predicates

```
Input: Two predicates, $p_1$=$\langle\{v\}, \{w_1\}\rangle$, $p_2$=$\langle\{v\}, \{w_2\}\rangle$
Output: $\Phi_1$ - Evidences for $p_1$, $\Phi_2$ - Evidences for $p_2$
$d \leftarrow$TypeOf($v$);
$L_1 \leftarrow$ \{PostingList($d$), PostingList($w_1$)\};
$L_2 \leftarrow$ \{PostingList($d$), PostingList($w_2$)\};
$L \leftarrow L_1 \cup L_2$;
while successfully join $L$ on some entity $e$ do
    while successfully join $L_1$ on occurrence $\phi$ do
        $\Phi_1 \leftarrow \Phi_1 \cup \{\phi\}$;
    while successfully join $L_2$ on occurrence $\phi$ do
        $\Phi_2 \leftarrow \Phi_2 \cup \{\phi\}$;
```

4.4 Type-Specific Entity-Centric Index

Traversing a TE posting list is potentially very costly since each TE posting list encodes occurrences of all entities of a type, e.g., about 40 million occurrences for PERSON. To reduce the overhead of traversing TE posting lists, we propose type-specific entity-centric index as an alternative. It creates multiple posting lists for each unique word, one for each type. In other words, each posting list is created for a combination of word w and type d, encoding all co-occurrence information between w and entities of type d. Each entity e in the posting list further records the document IDs, sentence numbers, positions of w and itself. Conceptually, a type specific entity-centric index is a fully materialized join between posting lists from TE and WE components. Query processing with type specific index can be done as before. We omit the details.

Table 2: 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

5. EXPERIMENTS

Our initial attempt of SSQ is a prototype system over `Wikipedia` and is accessible at **http://idir.uta.edu/ssq**. In this section, we provide experimental results on the effectiveness of SSQ in comparison with various customizations of our ranking model as well as other systems. We also experimented efficiency of our entity-centric index, comparing with the most commonly used indexing approach by other systems as well as its extended version.

5.1 Prototype and Experimental Settings

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 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 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 2) 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 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.

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. ∎

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.

[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 3: Retrieval Ability of 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

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 join predicates). OWN28 contains 28 manually designed queries, including 16 single predicate queries, 5 multi-predicate queries without join and 7 multi-predicate queries with join.

5.2 SSQ vs. IE-based System

Open IE is a well known approach that has been proposed to support structured queries on large-scale textual documents. We compare our SSQ 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 11 single predicate queries from INEX17, converted them into TextRunner-friendly format, 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, "by" "Neil Gaiman" are converted into "Neil" "Gaiman" for TextRunner (current TextRunner does not support phrases). Table 3 compared the recalls of SSQ and TextRunner on the 11 single-predicate queries. For each query, the table shows the number of correct answers returned by each system. Surprisingly, TextRunner provided much less correct answers than SSQ for most of the queries.

This improvement over TextRunner is surprising (since TextRunner extracts from a much larger corpus), but understandable. 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 term co-occurrences.

5.3 Analyzing Alternative Ranking Methods

In this section, we compare and analyze the multiple ranking methods discussed earlier, namely BCM, CM, PROX, MEX, and COUNT, to understand the effectiveness of the different shallow features exploited in our ranking model. Note that all the methods differ only in how they compute predicate level scores, and they use the same aggregate function to compute query level score. We compared these ranking methods using three popular measures: *nDCG*, *MAP*, and *Precision-at-k*.

nDCG: Normalized Discounted Cumulative Gain is a commonly adopted measure for ranking effectiveness of an IR system. Table 4(a) shows nDCG for each query in INEX17. Both MEX and PROX outperform COUNT, by 0.027 and 0.038 on average, respectively. The improvement is much more distinguishable on multi-predicate queries. We observed CM and BCM in general outperforms (or are at least as good as) the other methods.

MAP: Mean Average Precision computes the arithmetic mean of average precisions for a set of queries. It is a good indicator of both precision and recall. Table 4(b) shows the average precision of each individual query in INEX17 and the MAP for single-predicates queries, multi-predicate queries, and all queries. The

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

Table 4: Comparison of Alternative Ranking Methods

Query	COUNT	MEX	PROX	CM	BCM	ER
			nDCG			
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			
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

(a) nDCG and MAP on INEX17

			nDCG			
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			
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

(b) nDCG and MAP on OWN28

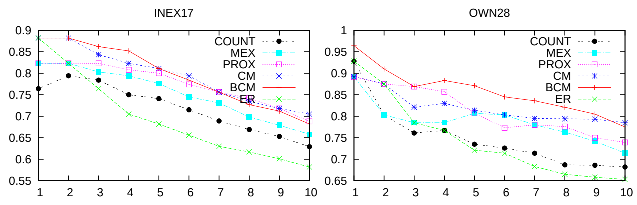

Figure 2: Precision-at-k on INEX17 and OWN28

observations are similar to those from the nDCG analysis, although BCM perfroms slightly better than CM on multi-predicate queries. Table 4(c) summarizes the same analysis on the OWN28 query set. Most comparisons are consistent with INEX17. However, for multi-predicate queries on OWN28, BCM demonstrates clear superiority over CM in both nDCG (by 0.034) and MAP (by 0.064).

Precision-at-k: Both nDCG and MAP measure the overall quality of a ranked list of answers. However, most IR systems focus on precision at top answers because top answers are more important. While nDCG captures this bias to a certain extent, it is still not quite illustrative. Therefore, we also analyze the results by measuring the precision for top-k answers, where $k = 1...10$. For each k, the precision at rank k is the averaged precision of all queries. Figure 2 illustrates the analysis results. We observe that Both CM and BCM are among the best methods. BCM appears to be consistently the best among all, while CM has inconsistent performance on INEX17 and OWN28.

In summary, each individual components in the cumulative ranking model is effective in ranking SSQ query answers and they work best in concert when integrated into the CM model. The BCM model has the same ranking performance as the CM model for single-predicate queries, but seems to be better for multi-predicate queries. The phenomenon can be explained as follows. Even though the ranking at predicate level does not differ much for the two models, the actual scores assigned at predicate level vary. When aggregated at query level, predicate scores affect the final scores and thus impact query-level ranking. BCM appears to assign more proper scores than CM at the predicate level, and consequently results in better ranking for multi-predicate queries. This difference is more distinguishable on OWN28 than on INEX17 is because

Table 5: MAP of SSQ and Other Systems

	SSQ	EntityRank	INEX-XER	INRIA
MAP	0.860	0.831	0.341	0.390

OWN28 has more multi-predicate queries and the observation becomes more stable.

5.4 SSQ vs. Other Entity Ranking Systems

We further investigate how well SSQ, the BCM model in particular, performs in comparison with other entity ranking systems. Three recent systems are compared: *EntityRank*, *INEX-XER*, and *INRIA*.

EntityRank(ER)[13]: This is a search-based entity retrieval and ranking system. We re-implemented their system for comparison.

INEX-XER[2]: The INEX Entity Ranking track also adopts `Wikipedia` as its corpus and entity set. We picked the MAP achieved by the best system published in the 2008 track. In order to avoid the overhead in assessing all ranking results, INEX used a sampling strategy to estimate the MAP of participating systems. Therefore, the MAP measure shown for INEX-XER is not the actual MAP but an estimation of MAP based on sampling.

INRIA[30]: This system developed at INRIA and RMIT ranks entities using traditional document ranking techniques, combining link analysis (HITS) and tf-idf (BM25) weighting. It also exploits entities' category similarity (Jaccard similarity) in the ranking model. The MAP of the system is obtained from 18 topics adapted from INEX 2006 ad hoc track. The system also used `Wikipedia` as data set.

From Table 5, we observe that SSQ out-performs (with MAP as the measure) the two INEX based systems (INEX-XER and INRIA) by a large margin. We acknowledge that the results are impacted by two inherent factors: First, they use different query sets and snapshots of `Wikipedia`. Second, they have different query formats. As required by the track, INEX-XER takes natural language descriptions as query input, rather than simple keywords, let alone structured query. However, the large margin indicates that SSQ can at least be complementary to these systems.

The performance of EntityRank are shown in the last columns of Table 4 and Figure 2. Especially from Figure 2, we observe that EntityRank performs well at ranking top 2 answers, comparable to CM and BCM, but deteriorates very fast for $k > 2$, becoming even worse than the baseline, COUNT. This is in part consistent with the high MRR (Mean Reciprocal Rank) the authors have reported, which measures the effectiveness of a ranking system according to the rank of the first true answer.

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.

5.5 Efficiency

This section reports our experimental results on query processing performance. We compared entity-centric index (referred to as WEO[5]) with document-centric index (referred to as WOE), the most commonly used index by existing systems [13, 11] to support entity retrieval. The WOE index typically consists of two components, full text index and type index. The type index creates one posting list for each type encoding the occurrences of entities of that type. However, it organizes occurrences in traditional document-centric way. In other words, it orders entity occurrences in ascending order of document ID and for each document it records a list of occurrences. Each occurrence encodes the sentence number, entities in the sentence and their positions. Entity retrieval with WOE index is pretty much the same as standard proximity search in IR. We also implemented a type specific version for WOE index, which is parallel to our type specific WEO. They are referred to as T-WEO and T-WOE respectively.

[5]W for Word, E for Entity and O for Occurrence

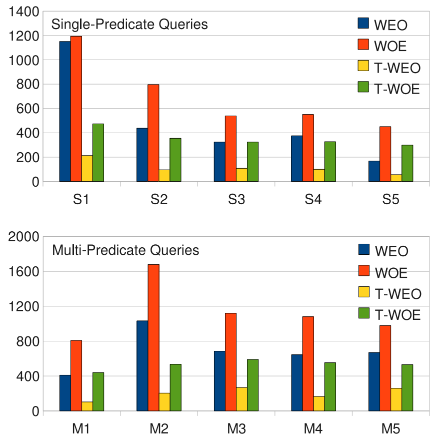

Figure 3: Execution Time of WEO,WOE,T-WEO and T-WOE

We randomly drew five single predicate queries and five multi-predicate queries in total from INEX17 and OWN28. All are non-join predicates. Figure 3 compares all four indexes for each test query by retrieval time cost (in milliseconds). For each index, the time is averaged over 10 executions of the query. Each execution starts with a cold operating system cache and restarting of the retrieval system (Note that we only measure the query execution time, excluding system restarting time).

From the figure, we can conclude that the commonly adopted WOE index clearly has the worst performance, generally costs 50% more of the time needed by WEO, for both single-predicate and multi-predicate queries. In many cases, WEO even shows comparable efficiency to T-WOE. Our T-WEO is yet significantly faster than T-WOE, reducing the time by more than 50% in general.

Table 6 provides a detailed outlook on the performances of different indexes for multi-predicate queries. Each multi-predicate query is split into a set of single-predicate queries (in our case, two sub-queries) and measured the execution time of sub-queries.

As seen from the table, processing a multi-predicate query as a whole costs less time than the summation of independent execution time of sub-queries. We believe that this is primarily due to system caching. However, intrinsically system caching can never make a multi-predicate query processed faster than any of its sub-queries. However, our optimized processing strategy for WEO/T-WEO could even reduce the execution time to below a sub-query (values in boxes). For example, T-WEO takes 103ms to execute query M1, almost half the time to execute sub-query M1.2 alone. This unique improvement is achieved by pruning unnecessary posting list traversals and occurrence level join.

Our current results on query processing, especially WEO/T-WEO index is very encouraging. In future we would like to further investigate this problem and study the behavior of the new index on large corpus. Without surprise, the document entries in WOE index would grow linearly with the corpus size. However, entity entries in WEO index is unlikely to grow as fast, which is nice property of WEO index to be exploited.

6. RELATED WORK

Table 6: Analysis of Multi-Predicate Query Precessing (in milliseconds)

	M1.1	M1.2	M1	M2.1	M2.2	M2	M3.1	M3.2	M3	M4.1	M4.2	M4	M5.1	M5.2	M5
WEO	235	356	410	1000	1035	$\boxed{1033}$	992	390	$\boxed{685}$	329	540	644	471	387	670
WOE	497	660	808	1110	1138	1679	1044	634	1119	600	894	1080	719	600	979
T-WEO	47	177	$\boxed{103}$	182	216	$\boxed{204}$	322	25	$\boxed{268}$	46	284	$\boxed{165}$	218	161	259
T-WOE	316	389	440	398	457	534	576	265	590	283	526	553	429	361	531

While we have already compared with various related works both analytically in Section 1 and empirically in Section 5, in this section we summarize some of the most relevant ones. Search-based approaches for querying entities have emerged recently. By incorporating linguistic annotations in its *neighborhood index*, Binding Engine [9] enables querying with linguistic patterns. The method is intended for smoothing linguistic tasks and is limited to phrase patterns. Chakrabarti et al. [11] studied proximity features in detail and learned an exponential scoring function on proximity. However, it only focuses on evidence-level scoring and makes no attempt to integrate evidences found in multiple documents to improve ranking. EntityRank [13] proposed the Impression Model to address this issue. Given an answer, the model evaluates each of its evidences and aggregates all its evidence scores to arrive at an answer score. A hypothesis testing step is included to assess the significance of answer scores and rank entities by the significance. SSQ differs from these methods in its ability in addressing more complex multi-predicate queries and join queries. Vercoustre et al. [30] retrieves and ranks entities in traditional IR fashion. Their work is also on `Wikipedia`, treating each `Wikipedia` article as an entity. An article is retrieved as an answer if its content contains the query keywords. The scoring function is based on term frequency and link analysis, both being classic document ranking methods.

7. CONCLUSION

In this paper, we proposed a novel querying paradigm, Shallow Semantic Query, which enables users to issue structured entity queries over textual content and obtain direct answers. Our SSQ ranking model leverages shallow syntax features to evaluate whether entities satisfy implied semantics of user-provided keyword predicates. We also designed the novel entity-centric index and various query processing strategies to support efficient entity retrieval. Experiments on a prototype showed both our ranking model and query processing strategies are effective, in comparison with various other systems, indicating that SSQ is a promising approach to address entity-related, structured information needs.

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

[7] 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.

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

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

[17] 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.

[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] 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.

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

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

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

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

[29] 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.

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

[31] 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.