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 searches 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 user 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. Our initial attempt of SSQ is a prototype system over `Wikipedia` and is accessible at http://idir.uta.edu/ssq. Experiments show the effectiveness of SSQ ranking model, which achieves NDCG of 0.921 and MAP of 0.773 for a set of test queries.

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. [1]

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

[1] We note here that, with the increasing ease of building tables and lists on Web pages, the answers for Task 1 (or other similar tasks about creating popular lists) may or may not be presented in a list. The answers for tasks like Task 2, however, remain a significant challenge for humans to construct.

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 answer tuple to co-occur 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. [2] 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, 32, 31], 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 information 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 4) 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 surrounding problems, there are several key challenges in developing SSQ. *First and foremost*, 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. A document about computer 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. In this paper, we will focus on the first two challenges, which we believe are the foundations of a full-fledged SSQ system.

[2] We acknowledge that, the effectiveness of SSQ partially relies on the user's capability in providing proper keyword constraints, just like in IR queries.

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 ranking framework for SSQ query results (Section 2). Following this framework, we propose a ranking method based on a group of shallow syntax features based on entity-keyword co-occurrence (Section 3).
- We implemented a prototype SSQ system (Section 4) and conducted comprehensive experiments to demonstrate the effectiveness of our ranking method (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 corresponding type for $v_i \in V$. Note that there can be multiple variables with the same type, 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. If $|V_p|$=1, the predicate is referred to as a *selection predicate* as it matches one entity at a time. Otherwise, it is referred to as a *join predicate* since it groups multiple entities.

– $U \subseteq V$ 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}, p_1=⟨{p}, {"Stanford", "graduate"}⟩, p_2=⟨{c}, {"Silicon Valley"}⟩, p_3=⟨{p, c}, {"found"}⟩. 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 ⟨Jerry Yang, Yahoo!⟩ and ⟨David Filo, Yahoo!⟩ are both answers to Query 1. If c is the only output variable, only one ⟨Yahoo!⟩ 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 evidences.

Definition 2 (SSQ Answer Tuple): Given an SSQ 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 $d_i \in D$.[3]

– 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 set of entities in t that are instantiations of variables in V_p); (2) all the phrases in C_p, where 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. ■

As defined, only tuples with evidences supporting all predicates are considered as answers. The unit of textual context for the evidences can be sentences, paragraphs, documents, windows of words in documents, etc., according to application preference. Different types of textual contexts have different retrieval capabilities and thus different sets of answer tuples. An SSQ system may deal with multiple types of textual contexts, using different evaluation methods. In this paper, we choose the unit of context to be sentences 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. In the following discussion, we will use evidence and sentence interchangeably unless a distinction is needed.

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 ⟨tuple, score⟩ pairs $\langle t, r\rangle$, satisfying the following conditions:

– t is a 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=$\langle V_p$={$v_{k_1}, \ldots, v_{k_t}$}, $C_p\rangle \in P$ 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 to see 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=⟨Jerry Yang, Yahoo!⟩ 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=⟨Jerry Yang, Google⟩ is less likely to be an answer since $p_3(t_2)$ is close to 0.

We shall make several important notes with regard to Definition 3:

– Our definition of SSQ 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 for particular preference. 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.

[3]We discuss how to assign types to entities in Section 4.1.

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

3. SSQ RANKING

Despite the structural query specification, SSQ is at its core an information retrieval system over unstructured text contents. Its flexibility in specifying keyword-based predicates brings two main challenges in ranking. First, the keyword-implied query semantics must be somehow captured in ranking, instead of simple keyword matching. 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 for Single-Predicate Scoring

With regard to a single predicate, a naive 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, only a few tuples have distinctively more evidences than others, while most tuples that are true answers have small numbers of evidences. In the meantime, there are false positive tuples that happen to appear in some 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.

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 (e.g. an adjective+noun 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.

Different representations may be used in different 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". In the above formula, we use $token(e,s)$ to refer to the tokens in s representing entity e. (See Section 4 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.*

In Evidence 1, the predicate has two phrases,"Stanford" and "graduate", each with one token, hence $\sum_{c \in C_p} |c|$=2. 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 ranges from "Stanford" to "Price", spanning five tokens, thus $|scope(t,p,s)|$=5. Therefore, the proximity of Dick Price in Evidence 1 is $prox(t,p,s)$=(2+2)/5=0.8. Similarly, the proximity of Barbra Ann Briggs in Evidence 2 is (2+3)/9=0.56. Based on proximity alone, we would 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 stands 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. Yet another pattern, c_1ec_2, is seldom used.

To distinguish good ordering patterns (those indicating valid evidences) from bad ones, we may assign a different weight to each pattern, so that tuples with evidences in good 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 good 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 such evidence like *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 weights of different ordering patterns for each predicate. In particular, we use the frequency $f(o)$ of each pattern o as its weight:

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

where S is the set of all evidences retrieved for a given predicate and $S(o) \subseteq S$ the set of evidences in which pattern o exists. This definition assumes that good patterns appear more often than bad 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*. 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 the pattern $o_2 = c_2c_1e$. 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. For a sentence s with colliding patterns $O(s)$, each colliding pattern $o \in O(s)$ gets a credit $credit(o, s)$, and the credits of all the colliding patterns sum up to the total credit of s, i.e., $\sum_{o \in O(s)} credit(o, s)$=1. For a sentence without collision (i.e., there is 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. Practically, for evidence s, denote $T(o, s)$ be the set of tuples in s, which are projected on V_p and follows pattern o. For $o \in O(s)$, we choose one representative tuple from $T(o, s)$, denoted by $t(o, s)$, which is the one with the highest proximity value among those that follow pattern o in s, i.e., $t(o, s)$=$\arg\max_{t \in T(o,s)} prox(t, p, s)$. 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 computed as,

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

where $S(t(o, s))$ is the set of evidences from which $t(o, s)$ is found. Note that we choose the most proximate tuple as 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 phrases in the predicate, and hence most reliable for allocating credit to the pattern.

In Evidence 3, $t(o_1, s)$=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. $t(o_2, s)$=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 including Evidence 3, i.e., $|S(t(o_1, s))|$=4 and $|S(t(o_2, s))|$=2. Therefore $credit(o_1, s)$=4/(4+2)=0.67 and $credit(o_2, s)$ = 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. The credit of o 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. Here we address the problem.

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 $S(t, o)$ 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 for both F_S and F_J is uniformly defined as

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

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

The model divides $S(t)$, t's evidences with regard to p, into $|O|$ groups, $\{S(t, o) | o \in O\}$, 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 $S(t, o)$. Group scores are then combined linearly using $f(o)$ as weights (the outer summation), such that group scores of good patterns (which are also common patterns) account more in predicate score $p(t)$.

The kernel of the scoring function, $prox(t, p, s)credit(o, s)$, 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 predicate p.

It is interesting to note that if all the features in cumulative model are turned off, setting $prox(t, p, s) \equiv 1$, $credit(o, s) \equiv 1$ and $f(o) \equiv 1$, the model will be reduced to

$$p(t) = \sum_{o \in O} 1 \sum_{s \in S(t,o)} 1 = \sum_{o \in O} |S(t, o)| = |S(t)|$$

which is equivalent to naively scoring t by the number of its supporting evidences with regard to p. In fact, the aforementioned cumulative scoring model can be customized easily by switching on and off these features in various combinations, so that we can evaluate the effectiveness of individual features. Such evaluation is presented in Section 5.

By cumulative scoring, a tuple's score on one predicate is not bounded. When multiple predicate scores are aggregated at query level, the score on one predicate might be so high as to dominate the query level score. To alleviate this problem, we propose an alternative predicate scoring model, *bounded cumulative* model as follows. It differs from cumulative model in the computation of

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

group scores. Basically, it bounds a group score between [0,1], and consequently predicate score between [0,1].

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

3.3 Query Level Ranking

After an entity tuple t has been evaluated on all the predicates, a final query-level judgment, F_A, of t is made by combining predicate scores. In this paper, we assume every predicate, either selection predicate or join predicate, is equally important and use product to aggregate 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 scenario than summation, another commonly used aggregate, because it favors answers with balanced predicate scores to those with polarized scores. To illustrate why balanced scores should be favored, let's 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. IMPLEMENTATION

We implemented our SSQ system by extending `Apache Lucene`[4], an open source full text index engine, and carried out the experiments on the 2008-07-24 dump of `Wikipedia`. We removed all the category pages, list 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, references and categories. Although tables and infoboxes also present valuable structured information, 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 specific techniques. The main body of the article is then segmented into sentences for indexing.

4.1 Identifying Entities and Types

For demonstrative purpose, We manually defined ten entity types (see Table 2) and use simple regular rexpressions to assign articles into these types based on their categories.[5] For example, if an article belongs to a category whose name ends with "novels", like "British novels", we assign it to type NOVEL. About 0.75 million out of the 2.4 million articles are assigned to these types and thus treated as entities in the system. An entity may belong to multiple types, e.g., David Beckham belongs to PLAYER and more generally to PERSON. This simple method turned out to be quite accurate and sufficient for demonstrating the effectiveness of SSQ system. We are also investigating more thorough categorization of articles as part of our future work.

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
UNIVERSIY	19,717	6,141,840	311
TOTAL	742,862	95,648,273	129

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

To identify occurrences of these 0.75 million entities, we exploited *internal link* 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 entity named "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.

4.2 Extending Full Text Index

Although emboding the notion of semantics, SSQ system is essentially based on keyword search and relies on full text index for query processing. However, traditional full text index does not directly support the notions in SSQ, like evidence and entity.

A traditional full text index build a posting list for each term, recording pairs of $\langle doc, pos \rangle$, which tell at what positions of which documents it occurs. When searching with two keywords "Stanford" and "graduate", the index will locate all their co-occurrences in same documents but does not tell whether they co-occur in the same sentences, which is required by the definition of evidence. Sentence level search [30] has been used in many tasks like question answering and novelty detection. In our implementation, we augment each $\langle doc, pos \rangle$ with a sentence number s, $\langle doc, pos, s \rangle$. If "Stanford" occurs as the 34th token in document 8 and is in the 3rd sentence. Its full position information is recorded $\langle 8, 34, 3 \rangle$. Thus while searching for evidences, each co-occurrence of "Stanford" and "graduate" will be checked to see weather their sentence numbers are the same.

To facilitate retrieval of typed entities, we extended `Lucene` with *type-independent* entity index. It has two components, a single posting list (augmented with sentence number) recording occurrences for all entities and a type hash that maps each entity to the list of types it belongs to (Figure 1).

The entity index assumes a virtual term "⟨ENTITY⟩" to stand for all entities. When an entity (internal link) is encountered during indexing, the anchor text is sent for full text indexing, while the entity's ID (mapped from entity name, a.k.a. link target) is sent

[4] http://lucene.apache.org

[5] A `Wikipedia` article may belong to one or more categories. These categories are listed at the bottom of the article.

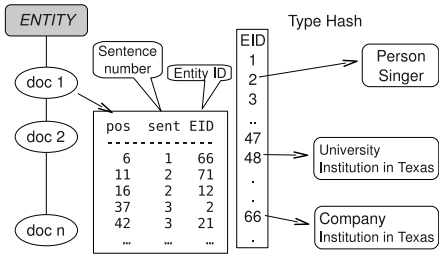


Figure 1: Type Independent Entity Index

to be to indexed as virtual term "⟨ENTITY⟩". However, different from full text index, each entity occurrence is recorded as a 5-tuple $\langle doc, pos, s, len, e\rangle$, where e records ID of the entity; len is the number of tokens in s representing e, i.e., $len = |token(e)|$, which will be used for computing proximity during query evaluation.

Type checking is done through type hash H. When a new entity ID is retrieved, the system obtains its types $H(ID)$ and check whether the required type is in the list $H(ID)$. Note that, we allow the same entity to have multiple types. For example, David Beckham belongs to PLAYER and the super type PERSON. Using type hash to allows easy maintenance of typing information. Updating the type(s) of an entity e is essentially updating the list $H(e)$, without changing the index of entity occurrences, which is very convenient.

4.3 Retrieving Query Answers

We take the selection predicate p : ["Stanford","graduate"] as an example to illustrate how SSQ retrieves entities. Basically, the system picks the posting lists of 1'⟨ENTITY⟩", "Stanford" and "graduate"[6] and perform proximity search on the three posting lists. Each co-occurrence of the three terms is checked to see whether they are in the same sentence. If so, the ID of the entity is retrieved and checked to see if it belongs to PERSON via type hash. Entities that passed typing checking are returned by the system for evaluation. Searching for join predicates is generally the same as selection predicate with slightly difference and the detail is avoided here.

The assumption of predicate independency allows us to process each predicate independently. The result of each predicate is a list of entity tuples. These lists are then joined to produce query answers. In Query 1, the three predicates result in three lists (the first three columns in Table 3) and the joining yields 4 query answers (the last column). ⟨Bill Gates, IKEA⟩ can be joined as a query answer because ⟨Bill Gates⟩ in p's list and ⟨IKEA⟩ in c's list can be joined with ⟨Bill Gates, IKEA⟩ in the (p, c)'s list.

5. EXPERIMENTAL RESULTS

In this section, we report the performance of our ranking method, in comparison with several baselines that are simplified variations of the cumulative scoring model. These ranking methods differs in predicate scoring, but uses the same F_A for aggregating scores of multiple predicates.

- **count** - This is the basic method that scores tuples by the number of supporting evidences. $p(t) = |S(t.V_p)|$.
- **prox** - To understand the effectiveness of proximity feature, we modify *count* to favor evidences of greater proximity. $p(t) = \sum_{s \in S(t.V_p)} prox(t, p, s)$
- **ex** - In this baseline, mutual exclusion rule is applied without using proximity information. $p(t) = \sum_{s \in S(t.V_p)} credit(o, s)$, where o is the pattern that $t.V_p$ follows in s. Without proximity, the representive of a colliding pattern o is randomly chosen from the tuples following that pattern.

  We will refer to our cumulative model as *cumu* and the bounded cumulative model as *bound*.

  The Entity Ranking track at INEX [7] works on a similar problem as ours. However, we were unable to compare track participants for two reasons. First, in this track, topics are given with natual language descriptions, the task of track participants is to return entities that satisfy the topic descriptions. In other words, they must automatically analysze the descriptions to form certain queries, instead taken queries provided by user as SSQ does. Therefore, it would be unfair to compare with them.

5.1 Test Queries

We also designed 28 other queries manually. Sixteen of them are single-predicate queries. Five are multi-predicate queries without join predicate. Seven are multi-predicate queries with join predicates. In all, the 28 queries cover the ten entity types in Table 2. All these are *list* queries that assume a list of true answers. The sizes of returned answers from SSQ varies greatly across queries, from about ten to thousands. Table 4 shows four queries and their top 20 answers by bounded cumulative scoring model. The correct answers are marked with leading bullets.

We have also designed some *factoid* queries that may have only one or two answers. However, most such queries appear to be easy such that all the methods can rank the right answers at top positions. Therefore we do not report them.

For each test query, we obtained its ground truth in the following manner. If SSQ returned no more than 100 answers, we manually checked each answer and labeled all the correct ones. Otherwise we took the union of the top 100 answers from all the ranking methods, checked and labeled the correct ones in the union set. In the latter case, the ground truth does not cover all the true answers in the returned set. We thus limited our accuracy measure to top 100 answers. For example, we measured precision-at-k for maximum k=100. This choice is purely due to limited manpower in manually checking the answers.

5.2 Comparison by NDCG

DCG (Discounted Cumulative Gain) is a commonly adopted measure of ranking effectiveness of an information retrieval system. It accumulates the gain of a document based on its relevance to the query and its rank. The gain of a relevant document at lower rank will be discounted more. NDCG (Normalized DCG) is the ratio between DCG of tested ranking method and the DCG of the perfect ranking.

$$DCG_k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(1+i)}$$

Though designed for retrieval result with graded relevance, NDCG can also be used for binary-relevance result. In SSQ, if an answer belongs to the ground truth it has relevance 1; otherwise 0. A plausible feature of NDCG is that it can be used to average over queries with different return list sizes. This is important to our experiments since the size of returned answers varies greatly in different queries.

We compare NDCG at position 100 for *single-predicate queries only*, *multi-predicate queries only*, and *all queries*. From Table 5,

[6]All indexed terms are stemmed and converted to lowercase.

[7]http://www.inex.otago.ac.nz/

Table 3: Joining Entities to Produce Query Answer

p	c	(p, c)	**Answer**
⟨Larry Page⟩	⟨eBay⟩	⟨Larry Page, Google⟩	⟨*Larry Page, Google*⟩
⟨Bill Gates⟩	⟨Yahoo!⟩	⟨Bill Gates, Microsoft⟩	⟨*Bill Gates, IKEA*⟩
⟨Jerry Yang⟩	⟨Cisco Systems⟩	⟨Steve Jobs, Apple Inc.⟩	⟨*Jerry Yang, Yahoo!*⟩
⟨George W. Bush⟩	⟨Google⟩	⟨Bill Gates, IKEA⟩	⟨*Scott McNealy, Cisco Systems*⟩
⟨David Filo⟩	⟨IKEA⟩	⟨Jerry Yang, Yahoo!⟩	
⟨Scott McNealy⟩	⟨IBM⟩	⟨Jack Welch, General Electric⟩	
-	⟨Apple Inc.⟩	⟨Scott McNealy, Cisco Systems⟩	

Table 4: Four Test Queries and Top 20 Answers by *bound*. (Correct answers are marked by bullets.)

us-state-capital (Q2)	**ibm-turing-award (Q3)**	**stanford-people-company (Q1)**	**robert-de-niro-director (Q4)**
•Little Rock, Arkansas	•John Backus	•David Packard, Hewlett-Packard Company	•Taxi Driver, Martin Scorsese
•Tallahassee, Florida	Niklaus Wirth	Bill Gates, Microsoft	•Goodfellas, Martin Scorsese
•Montpelier, Vermont	John McCarthy	•Jerry Yang, Yahoo!	•The Deer Hunter, Michael Cimino
•Providence, Rhode Island	•Frances E. Allen	•Vinod Khosla, Sun Microsystems	•A Bronx Tale, Robert De Niro
•Frankfort, Kentucky	Richard Hamming	•Scott McNealy, Sun Microsystems	•Raging Bull, Martin Scorsese
•Topeka, Kansas	•Edgar F. Codd	•William Hewlett, Hewlett-Packard Company	•Cape Fear (1991 film), Martin Scorsese
•Salem, Oregon	Adi Shamir	•Larry Page, Google	•Casino (film), Martin Scorsese
•Lincoln, Nebraska	Donald Knuth	•Andy Bechtolsheim, Sun Microsystems	•Heat (film), Michael Mann (director)
•Trenton, New Jersey	Ken Thompson	Bill Joy, Sun Microsystems	•Men of Honor, George Tillman, Jr.
•Salt Lake City, Utah	Manuel Blum	Andy Bechtolsheim, Cisco Systems	Taxi Driver, Robert De Niro
Washington, D.C.	Edsger W. Dijkstra	•Sergey Brin, Google	•The Godfather Part II, Francis Ford Coppola
•Boise, Idaho	•Fred Brooks	Vinod Khosla, Kleiner Perkins Caufield & Byers	Raging Bull, Robert De Niro
•Lansing, Michigan	Marvin Minsky	•David Filo, Yahoo!	•The King of Comedy, Martin Scorsese
•Richmond, Virginia	Fernando J. Corbató	Vinod Khosla, Veritas Software	•Analyze This, Harold Ramis
•Raleigh, North Carolina	•Kenneth E. Iverson	Larry Augustin, Yahoo!	•The Mission (film), Roland Joffé
•Augusta, Maine	Dennis Ritchie	James H. Clark, Silicon Graphics, Inc.	Heat (film), Robert De Niro
•Concord, New Hampshire	Christopher Strachey	•Sandra Lerner, Cisco Systems	•Midnight Run, Martin Brest
Sydney		Donna Dubinsky, Palm, Inc.	•Once Upon a Time in America, Sergio Leone
•Montgomery, Alabama		Jeff Hawkins, Palm, Inc.	•Awakenings, Penny Marshall
New York City		Robert Metcalfe, Microsoft	•Frankenstein (1994 film), Kenneth Branagh

* (Q2) us-state-capital: Find state capitals of United States. `SELECT c FROM CITY AS c WHERE c:["US" "state" "capital"]`
* (Q3) ibm-turing-award: Find Turwing Award laureates who worked for IBM.
  `SELECT p FROM PERSON AS p WHERE p:["Turing Award"] AND p:["IBM"]`
* (Q1) stanford-people-company: Query 1 in Section 1.
* (Q4) robert-de-niro-director: Find films starring Robert De Niro and the films' directors.
  `SELECT f, p FROM FILM AS f, PERSON AS p WHERE f:["Robert De Niro"] AND (f,p):["direct"]`

we can see that for single-predicate queries, mutual exclusion and proximity features are both effective, improving the NDCG over *count* by 0.025 and 0.03, respectively. *cumu* and *bound* are slightly better than *prox*, by 0.005 and 0.006, respectively. For multi-predicate queries, the observation was similar, and *bound* reached 0.878, consistently better than all other methods. Overall, *bound* appears to be the most effective ranking method.

A subtle observation shows that, *cumu* and *bound* are equally effective for single predicate queries, but differ much 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 methods, the scores assigned at predicate level are varied. When aggregated at query level, predicate scores affect the final scores and thus impact query-level ranking. *bound* appears to assign more reasonable scores at the predicate level, and consequently results in better ranking for multi-predicate queries.

5.3 Comparison by MAP

MAP (Mean Average Precision) is another popular measure in IR. It is the arithmetic mean of average precisions for a set of queries. The average precision of a query approximates the area under the precision-recall curve, and thus a good indicator of both precision and recall. However, measuring average precision requires that all the true answers in the returned list be known, which is not guaranteed by our way of collecting ground truth. Therefore, we exclude those queries for which we cannot obtain all true answers. This leaves us 15 single-predicate queries and 8 multi-predicate queries. We report the MAP results on these queries.

From Table 6, we observe a result similar to NDCG comparison. *prox* and *ex* both yield better performance than the naive *count* method. *bound* tied *cumu* as the winners for single predicate queries, and was 0.02 better than *prox*. For multi-predicate queries, *bound* clearly out-performed *cumu*, for the same reason explained for NDCG result. Overall, *bound* is more effective than others.

5.4 Comparison by Precision-at-K

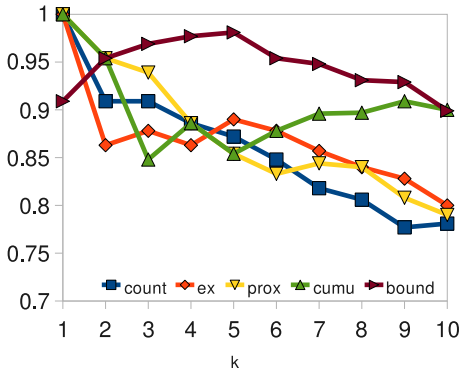
Both NDCG and MAP are aggregates that measure the overall quality of a ranked list of answers. To better understand how the ranking methods behave at different positions, we measured precision-at-k. For the 11 queries with over 100 answers from SSQ, we measured the precision of each ranking method, at every position from 1 to 100. Precisions at top 10 are shown in Figure 2(a). It can be seen that *bound* had lower precision at rank 1 but quickly catched up at rank 2 and was the best afterwards. The precision curve up to 100 is depicted in Figure 2(b). In the long run, *bound* and *cumu* are comparable and have consistently higher precisions

Table 5: NDCG at Position 100

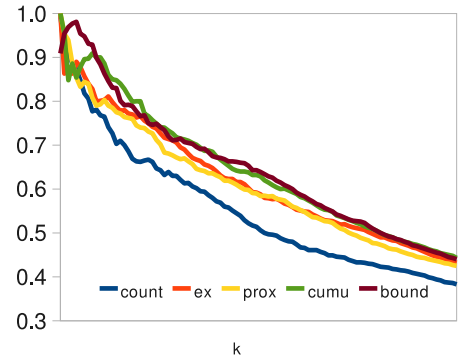
query	count	ex	prox	cumu	bound
Single-Predicate Query					
jane austen novel	0.968	0.979	0.984	**0.989**	0.988
fifa player	**0.971**	0.952	0.968	0.960	0.957
oracle acquire	0.946	0.961	0.966	**0.986**	0.975
spielberg film	0.982	0.989	0.998	0.999	**1.000**
us open champion	0.946	0.974	0.939	**0.977**	**0.977**
computer game company	0.973	0.980	0.977	**0.981**	0.965
eagles song	0.898	**0.901**	0.886	0.890	0.892
real madrid player	0.907	0.944	**0.959**	0.945	0.938
robert de niro film	0.977	0.982	**0.990**	0.988	0.981
turing award winner	0.832	0.988	0.965	0.988	**0.989**
born spain	0.818	0.866	**0.901**	0.893	0.894
celine dion songs	0.857	0.889	0.894	0.905	**0.907**
china province capital	0.952	0.967	0.983	0.990	**0.993**
pulitzer prize	0.964	0.986	0.994	0.995	**0.996**
silicon valley company	0.895	**0.897**	0.883	0.880	0.849
us state capital	0.778	0.823	0.863	0.868	**0.952**
MEAN	0.917	0.942	0.947	0.952	**0.953**
Multi-Predicate Query					
australian best actress	0.852	0.896	0.863	0.884	**0.901**
harvard us president	0.920	0.920	0.924	0.932	**0.953**
real madrid brazil player	0.709	0.716	0.783	0.790	**0.812**
ibm turing award	0.746	0.759	**0.792**	0.782	0.782
microsft game company	0.449	0.500	0.526	0.588	**0.798**
oscar film austrilian actor	0.907	**0.972**	0.936	0.966	0.936
bigten nobel prize prof	0.781	0.694	**0.865**	0.829	0.847
fifa player italian	**0.965**	0.927	0.961	0.872	0.794
nba champion mvp	0.843	0.889	0.885	0.928	**0.947**
novel film academy award	0.787	0.774	0.825	0.851	**0.909**
robert de niro film director	0.758	0.814	0.866	0.887	**0.957**
stanford people company	0.880	0.889	0.800	0.815	**0.903**
MEAN	0.800	0.813	0.836	0.844	**0.878**
OVERALL MEAN	0.866	0.887	0.899	0.906	**0.921**

Table 6: MAP on Subset of 23 Queries

query	count	ex	prox	cumu	bound
Single-Predicate Query					
jane austen novel	0.898	0.930	0.946	**0.962**	0.957
fifa player	**0.887**	0.826	0.881	0.850	0.839
oracle acquire	0.828	0.874	0.889	**0.952**	0.920
spielberg film	0.926	0.954	0.991	0.998	**1.000**
us open champion	0.716	0.847	0.729	**0.859**	0.858
eagles song	0.636	**0.652**	0.621	0.643	0.623
real madrid player	0.687	0.795	**0.858**	0.818	0.817
robert de niro film	0.862	0.914	**0.950**	0.941	0.917
turing award winner	0.630	0.945	0.860	0.951	**0.954**
born spain	0.566	0.633	**0.683**	0.673	0.674
celine dion songs	0.557	0.640	0.670	0.701	**0.710**
china province capital	0.824	0.875	0.931	0.960	**0.972**
pulitzer prize drama	0.814	0.937	0.970	0.977	**0.979**
silicon valley company	0.594	**0.619**	0.588	0.588	0.574
us state capital	0.544	0.612	0.729	0.725	**0.807**
MAP	0.731	0.804	0.820	**0.840**	**0.840**
Multi-Predicate Query					
harvard us president	0.761	0.769	0.770	0.791	**0.850**
real madrid brazil player	0.377	0.388	0.476	0.488	**0.528**
ibm turing award	0.484	0.498	**0.553**	0.533	0.533
microsft game company	0.221	0.285	0.306	0.361	**0.583**
bigten nobel prize prof	0.525	0.441	**0.673**	0.594	0.584
fifa player italian club	0.886	0.801	**0.875**	0.695	0.518
nba champion mvp	0.491	0.620	0.614	0.734	**0.795**
stanford people company	0.671	0.738	0.667	0.711	**0.778**
MAP	0.552	0.568	0.617	0.613	**0.646**
OVERALL MAP	0.669	0.721	0.749	0.761	**0.773**



(a) Precision up to rank position 10.



(b) Precision up to rank position 100.

Figure 2: Precision-at-k

than other methods across all positions up to 100.

5.5 Summary

The large amount of manual work in examination of query answers has prevented us from experimenting with more queries. Nevertheless, the experimental results show that, the shallow features exploited by SSQ worked well in concert, as integrated in our ranking model. The *bound* method, seems to work better than *cumu* for multi-predicate queries by protecting answer level scores from being dominoated by scores of single predicate.

6. RELATED WORK

Search-based entity query approaches 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. [31] 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 evaluation system leverages shallow co-occurrence features to uncover entities that satisfy the implied semantics of user-provided keyword predicates. A ranking model was developed under a linear framework. Experiments on a prototype implementation showed the effectiveness of the ranking model.

8. REFERENCES

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

[2] INEX 2009 entity-ranking track. `http://www.inex.otago.ac.nz/tracks/entity-ranking/entity-ranking.asp`. Retrieved on 2009-07-01.

[3] TREC 2009 entity track: Searching for entities and properties of entities. `http://ilps.science.uva.nl/trec-entity/guidelines/`. Retrieved on 2009-07-01.

[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*, pages 387–398, 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] Y. Uematsu, T. Inoue, K. Fujioka, R. Kataoka, and H. Ohwada. Proximity scoring using sentence-based inverted index for practical full-text search. In *ECDL*, 2008.

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

[32] 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.