EntityRank: Searching Entities Directly and Holistically[*]

Tao Cheng, Xifeng Yan, Kevin Chen-Chuan Chang

Computer Science Department, University of Illinois at Urbana-Champaign
{tcheng3, xyan, kcchang }@uiuc.edu

ABSTRACT

As the Web has evolved into a data-rich repository, with the standard "page view," current search engines are becoming increasingly inadequate for a wide range of query tasks. While we often search for various data "entities" (*e.g.*, phone number, paper PDF, date), today's engines only take us indirectly to pages. While entities appear in many pages, current engines only find each page individually. Toward searching directly and holistically for finding information of finer granularity, we study the problem of *entity search*, a significant departure from traditional document retrieval. We focus on the core challenge of ranking entities, by distilling its underlying conceptual model *Impression Model* and developing a probabilistic ranking framework, *EntityRank*, that is able to seamlessly integrate both local and global information in ranking. We evaluate our online prototype over a 2TB Web corpus, and show that *EntityRank* performs effectively.

1. INTRODUCTION

The immense scale and wide spread of the Web has rendered it as an ultimate information repository– as not only the sources where we *find* but also the destinations where we *publish* our information. These dual forces have enriched the Web with all kinds of *data,* much beyond the conventional *page view* of the Web as a corpus of HTML pages, or "documents." Consequently, the Web is now a collection of *data-rich* pages, on the "surface Web" of static URLs (*e.g.*, personal homepages) as well as the "deep Web" of database-backed contents (*e.g.*, flights from aa.com), as Figure 1 shows. While the richness of data represents a promising opportunity, it challenges us for effectively finding information we need.

With the Web's sheer size, our ability to find "stuff" we want mainly relies on how *search engines* respond to our *queries*. As current engines search the Web inherently with the conventional page view, they are becoming increasingly inadequate for a wide range of queries. To focus on the "stuff" we want, or data "entities", this paper studies the *entity search* problem, formulates the search framework, and in particular addresses the central issue of *entity ranking*.

[*]This material is based on work partially supported by NSF Grants IIS-0133199, IIS-0313260, the 2004, 2005 IBM Faculty Awards.

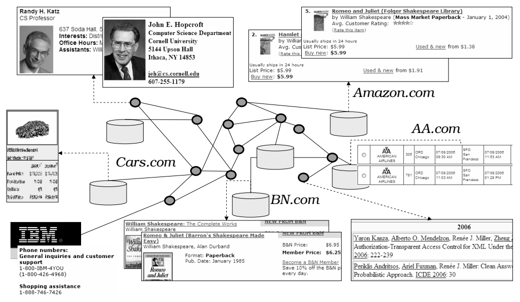


Figure 1: The Current Web: Proliferation of Data-Rich Pages

Motivating Scenarios: The Barriers

To begin with, we reflect: As users, what have we been looking for on the Web? The richness of data has tempted us to search for various "stuff"– Let us consider a few scenarios, for user Amy:

Scenario 1: Amy wants to call Amazon.com for her online purchase; how can she find the "phone number" of their customer service? To begin with, what should be the right keywords for finding pages with such numbers? Query "amazon customer service phone" may not work, as often a phone is simply shown without keyword "phone" (*e.g.*, customer service: (800) 717-6688). Or, "amazon customer service" could be too broad to return many pages. Amy must sift through the returned pages to dig for the phone number. This task can be time consuming, since some vendors may "hide" their service numbers (to reduce their workload). Fortunately, such information might reside in other probably less authoritative (or lower ranked) pages, *e.g.*, user forums, business reviews, and blogs. □

Scenario 2: Amy wants to apply for graduate schools; how can she find the *list* of "professors" in the database area? She might have to go through all the CS department homepages (in the hope that there is a page that lists professors by research areas), or look through faculty homepages one by one. This could be a very laborious process. □

Scenario 3: As a graduate student, Amy needs to prepare a seminar presentation for her choice of some recent papers; how can she find papers that come readily with presentations, *i.e.*, a "PDF file" *together with* a "PPT file," say from SIGMOD 2006? □

Scenario 4: Done with the presentation, now Amy wants to buy a copy of Shakespeare's Hamlet to read; how can she find the "prices" and "cover images" of available choices from, say, Borders.com and BN.com (She has seen the copy she likes before, so the cover page will be helpful to find out). She would have to look at the results from multiple online bookstores one by one and compare the listed price of each. □

In these scenarios, like every user in many similar situations, Amy is looking for particular *types* of information, which we call *entities*, *e.g.*, a phone number, a book cover image, a PDF, a PPT, a name, a date, an email address, etc. She is *not*, we stress, looking for pages as "relevant documents" to read, but entities as data for her subsequent tasks (to contact, to apply, to present, to buy, *etc.*).

However, the current search systems, with their inherent page view of the Web, are inadequate for the task of finding data entities, the focus of this paper. There are two major barriers:

First, in terms of the *input and output*, current engines are searching *indirectly*. 1) Users cannot directly describe what they want. Amy has to formulate her needs indirectly as keyword queries, often in a non-trivial and non-intuitive way, with a hope to hit "relevant pages" that may or may not contain target entities. For Scenario 1, will "amazon customer service phone" work? (A good page may simply list the phone number, without the word "phone.") 2) Users cannot directly get what they want. The engine will only take Amy to a list of pages, and she must scrutinize them to find the phone number. Can we help Amy to *search directly* in both describing and getting what they want?

Second, in terms of the *matching mechanism*, current search engines are finding each page *individually*. The target entities are often available in multiple pages. In Scenario 1, the same phone number of Amazon.com may appear in the company's Web site, online user forums, or even blogs. In this case, we should collect, for each phone, all its occurrences from multiple pages as supporting evidences of matching. In Scenario 2, the list of professors probably cannot be found in any single page. In this case, again, we must look at many pages to come up with the list of promising names (and similar to Scenario 1, each name may appear in multiple pages). Can we help users to *search holistically* for matching entities across the Web corpus as a whole, instead of individual pages?

Our Proposal: Entity Search

Toward searching directly and holistically, for finding specific types of information, we propose to support *entity search*.

First, as *input*, users formulate queries to directly describe what they are looking for: She can simply specify what her *target entities* are, and what keywords may appear in the *context* with a right answer. To distinguish between entities to look for and keywords in the context, we use a prefix #, *e.g.*, #phone for the phone entity. Our scenarios will naturally lead to the following queries:

Query $Q1$: ow (amazon customer service #phone)
Query $Q2$: (#professor #university #research="database")
Query $Q3$: ow (sigmod 2006 #pdf_file #ppt_file)
Query $Q4$: (#title="hamlet" #image #price)

In these queries, there are two components: (1) *Context* pattern, how will the target entities appear? $Q1$ says that the entity #phone will appear with these keywords in the pattern of ow or "ordered-window", *i.e.*, in that order and as close in a window as possible. We may also omit the pattern, *e.g.*, $Q2$ and $Q4$, in which case the implicit default uw or "unordered-window" is used (which means proximity– the closer in a window, the better). (The exact patterns depend on implementations. Section 4 will discuss the notion of such "recognition models.") (2) *Content* restriction: A target entity will match any instances of that entity type, subject to option restriction on their content values– *e.g.*, $Q1$ will match every phone instance, while $Q2$ will only match research area "database." (In addition to equality "=", other restriction operators are possible, such as "contain.")

Second, as *output*, users will directly get the entities that they are looking for. That is, as a query specifies what entity types are the targets, its results are those entity *instances* (or literal values) that match the query, in a ranked order by their matching scores. (We will discuss this matching next.) Figure 2 shows some example results for $Q1$ and $Q3$.

rank	phone number	score	urls
1	800-201-7575	0.9	amazon.com/support.htm myblog.org/shopping
2	800-988-0886	0.8	Dell.com/supportors
3	800-342-5283	0.6	xyz.com
4	206-346-2992	0.2	hp.com
...	...	...	...

rank	PDF	PPT	score	urls
1	sigmod6.pdf	sigmod6.ppt	0.8	db.com,sigmod.com
2	surajit21.pdf	surajit21.ppt	0.7	ms.com
...	...	...	...	...

Figure 2: Query Results of $Q1$ and $Q3$

Third, as *search mechanism*, entity search will find matching entities holistically, where an instance will be found and matched in all the pages where it occurs. For instance, a #phone 800-201-7575 may occur at multiple URLs as Figure 2 shows. For each instance, all its matching occurrences will be aggregated to form the final ranking– *e.g.*, a phone number occurs more *frequently* at where "amazon customer service" is mentioned may rank higher than those less frequent ones. (This ranking is our focus– See next.) Thus, while our search target is entities, as supporting "evidences," entity search will also return where each entity is found. Users can examine these snippets for details.

We note that, the usefulness of entity search is three-fold, as the sample results in Figure 2 illustrate. *First*, it returns relevant answers at top rank places, greatly saving search time and allowing users or applications to focus on top results. *Second*, it collects all the evidences regarding the query in the form of listing supporting pages for every answer, enabling results validation (by users) or program-based post-processing (by applications). *Third*, by targeting at typed entities, such an engine is data-aware and can be integrated with DBMS for building novel information systems– imagine the results of $Q1$ to $Q4$ are connected with SQL-based data.

Core Challenge: Ranking Entities

Toward building an entity search engine, we believe the core challenge lies in the entity ranking model– Obviously, while promising, such a system is only useful if good entity results can be found at the top ranks, much like today's search engines that strive to achieve the central mission of ranking relevant pages high.

As our discussion has hinted, there are several unique requirements of entity search. Entity search is 1) *contextual*, as it is mainly matching by the surrounding context; 2) *holistic*, entities must be matched across their multiple occurrences over different pages; 3) *uncertain*, since entity extraction is imperfect in nature; 4) *associative*, entities can be associated in pairs, *e.g.*, #phone and #email and it is important to tell true association from accidental; and 5) *discriminative*, as entities can come from different pages, and not all such "sources" are equivalent.

With these requirements, this paper focuses on entity ranking: We build our foundation by proposing the *impression model*, an "ideal" conceptual framework. With the conceptual guidance, we build the *EntityRank* scheme, taking a principled probabilistic view for scoring and ranking: We conceptualize the matching of a result as to estimate the *probability* how the entity instances are associated as a *tuple*, and compare it to a *null hypothesis* to discriminate accidental associations. With a *local recognition layer* for quantifying each instance occurrence, a *global access layer* for aggregating across pages, and a *validation layer* for hypothesis testing, *EntityRank* materializes the conceptual impression model.

Our results show that *EntityRank* is effective: In our prototype indexing 2 TB of real Web corpus, for Scenario 1, it consistently finds the right matches at top-3, for a sample of Fortune 500 companies, and similarly for a systematic querying of SIGMOD 2007 PC members. Section 6 will demonstrate the results for all four scenarios. We validate the seamless integration of local recognition and global access models– without either, the results are significantly degraded– as well as the need for hypothesis testing.

We start in Section 2 to formalize entity search. Section 3 presents the ideal conceptual model and Section 4 materializes it into the *EntityRank* scheme. We relate to existing studies in Section 5, and Section 6 reports our prototype system and experiments.

Contributions

1. We study and define the **characteristics and requirements** of entity search as the guideline for supporting effective ranking.
2. We distill the conceptual model **Impression Model**, and develop a concrete **EntityRank** framework for ranking entities.
3. We have implemented an online **prototype** with **real Web corpus**, and demonstrated the effectiveness of entity search.

2. THE PROBLEM: ENTITY SEARCH

To support entity-based querying, the system must be fundamentally *entity-aware*: That is, while current search engines are built around the notion of pages and keywords, we must generalize them to support entity as a first-class concept. With this awareness, as our data model, we will move from the current page view, *i.e.*, the Web as a document collection, to the new *entity view*, *i.e.*, the Web as an entity repository. Upon this foundation, we develop entity search, where users specify what they are looking for with keywords and target entities, as $Q1$ – $Q4$ illustrated. We have introduced our data model and formalized the problem of entity search in [12]. We now briefly describe these notions.

2.1 Data Model: Entity View

How should we view the Web as our database to search over? In the standard page view, the Web is a set of documents (or pages) $D = \{d_1, \ldots, d_n\}$. We assume flat set for discussion here; Section 4 will specialize D as a set of linked documents.

In our data model, we take an *entity view*: We consider the Web as primarily a repository of entities (in addition to the notions of pages): $E = \{E_1, E_2, \ldots, E_n\}$, where each E_i is an entity type. For instance, to support Scenario 1 (Section 1), the system might be constructed with entities $E = \{E_1 : \text{\#phone}, E_2 : \text{\#email}\}$. Further, each entity type E_i is a set of *entity instances* that are extracted from the corpus, *i.e.*, literal values of entity type E_i that occur somewhere in some $d \in D$. We use e_i to denote an entity instance of entity type E_i. In the example of phone-number patterns, we may extract #phone = {"800-201-7575", "244-2919", "(217) 344-9788, ...}

In this work, we consider only entities that can be recognized offline before query-time, and the extracted instances will be indexed for efficient query processing. The extraction can be done using simple pattern matching or state-of-the-art entity extractors. This paper focuses on the ranking model. More detailed information could be found in our system description [13]. To facilitate query matching, we will record the "features" of each occurrence e_i– These local occurrence features will facilitate our local model (Section 4.2) to quantify the strength of local matchings.

- Position $e_i.pos$: the document id and word offset of this instance occurrence, *e.g.*, $instance\ e_1$ may occur at (d_2, 23).
- Confidence $e_i.conf$: the probability estimation that indicates how this occurrence is regarded as an instance of E_i.

***Entity-Search* Query**.

- **Given**: *Entity collection* $\mathcal{E} = \{E_1, \ldots, E_N\}$, over *Document collection* $\mathcal{D} = \{d_1, \ldots, d_n\}$.
- **Input**: *Query* $q(\langle E_1, \ldots, E_m \rangle) = \alpha(E_1, \ldots, E_m, k_1, \ldots, k_l)$, where α is a *tuple pattern*, $E_i \in \mathcal{E}$, and k_j a keyword.
- **Output**: *Ranked list* of $t = \langle e_1, \ldots, e_m \rangle$, where $e_i \in E_i$, sorted by $Score(q(t))$, the *query score* of t.

Figure 3: The Entity Search Problem

With entities extracted and indexed, we transform the page view into our entity view. Note that the set of supported entity types must be determined, depending on the actual application setting, much like the "schema" of the system. In our system evaluation (Section 6) we demonstrate several application scenarios with the respective entities extracted and indexed offline.

We stress that each of these entities are *independently* extracted from the corpus, and are only associated by ad-hoc queries at query time. Thus, users may ask #phone with "ibm thinkpad" or "bill gates", or they ask to pair #phone with, say, #email for "white house". Supporting such online matching and association is exactly the challenge (and usefulness) of entity search.

2.2 Search Problem: Finding Entity Instances

We now state our entity search problem, as Figure 3 summarizes. First, for *input*, as queries, our entity search system lets users search for entities by specifying target entity types and keywords together in a *tuple pattern* α, which indicates users' intention of what the desired entities are, and how they may appear in D by certain patterns. We note that, as Section 1 motivated, entity search is essentially *search by context* over the document collection: As α intends to capture, our desired data often appear in some context patterns with other keywords or entities, indicating how they together combine into a desired *tuple* by their textual occurrences. A system will support, as its implementation decisions, a set of such patterns, *e.g.*, doc (the same document), ow (ordered window), uw (unordered window), and phrase (exact matching) (Section 4). A query can either explicitly specify a pattern (*e.g.*, $Q1$ and $Q3$) or implicitly assume the system default pattern (*e.g.*, $Q2$ and $Q4$).

Second, for *output*, the results are a ranked list of m-ary *entity tuples*, each of the form $t = \langle e_1, \ldots, e_m \rangle$, *i.e.*, a combined instance of each e_i as an instance of entity E_i desired in the query. A result tuple t will be ranked higher, if it matches the query better. We denote this measure of how well t matches q by a *query score* $Score(q(t))$, which should capture how t appears, by the desired tuple pattern α, across every document d_j in D, *i.e.*,

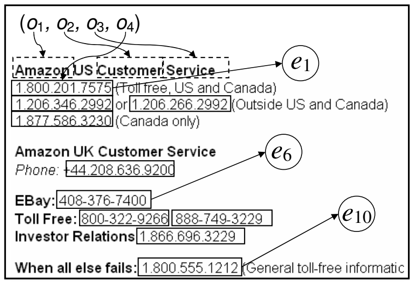
$$Score(q(t)) = Score(\alpha(e_1, \ldots, e_m, k_1, \ldots, k_l)).$$

We stress that, since the assessment of the query score defines the ranked list, it is the central function of an entity search system. The objective of entity search is thus to find from the space of $t \in E_1 \times \ldots \times E_m$, the matching tuples in ranked order by how well they match q, *i.e.*, how well entity instances and keywords in tuple t associate in the desired tuple pattern. As the focus of this paper, we will develop this scoring and ranking in Section 3 and 4.

2.3 Entity Ranking: Requirements

For effective entity ranking, it is crucial to capture the unique characteristics of entity search. Let's examine a sample query: find the phone number of "Amazon Customer Service"; q = (Amazon Customer Service #phone).

For each query, conceptually, an entity search system should analyze all pages available on the Web that contain the keywords "Amazon Customer Service" and a #phone instance. Figure 4 is



Amazon US Customer Service
1.800.201.7575 (Toll free, US and Canada)
1.206.346.2992 or 1.206.266.2992 (Outside US and Canada)
1.877.586.3230 (Canada only)

Amazon UK Customer Service
Phone: +44.208.636.9200

EBay: 408-376-7400
Toll Free: 800-322-9266 888-749-3229
Investor Relations 1.866.696.3229

When all else fails: 1.800.555.1212 (General toll-free informatic

Figure 4: Example Page: Amazon Customer Service #phone

a text snippet from an example webpage that contains keywords "Amazon Customer Service" and #phone instances. There could be many such pages. The first step, for such a page, is to match the keywords "Amazon Customer Service" and identify the entity instances desired (#phone). Next, we must rank these entity instances since there might be multiple phone numbers co-located with "Amazon Customer Service" in webpages. The ranking function eventually will single out which phone number is associated with "Amazon Customer Service" more strongly. An entity search system needs to take into account the following major factors (as Section 1 briefly mentioned).

- R-*Contextual*: The probability of association between keywords and entity instances in various contexts might be different. There are mainly two factors to consider:
  a) Pattern: The association of keywords and entity instances sometimes formulates a regular pattern; *e.g.*, a company's name often appears *before* its phone number is mentioned. Given a text snippet "Amazon 1-800-201-7575 eBay," the phone number is more likely to associate with "Amazon" than "eBay".
  b) Proximity: The association between the keywords and the entity instances is not equally probable with respect to how "tightly" they appear in the web page. Often, the association is stronger when the occurrences are closer. Use Figure 4 as an example. Phone number e_1 1-800-201-7575 is more likely associated with Amazon than phone number e_6 408-376-7400 as e_1 appears in closer proximity to keywords "Amazon Customer Service" than e_6.
- R-*Holistic*: As a specific phone number instance may occur with "Amazon Customer Service" multiple times in many pages, all such matchings must be aggregated for estimating their association probability.
- R-*Uncertainty*: Entity extraction is always not perfect, and its extraction confidence probability must be captured.
- R-*Associative*: We must carefully distinguish true associations from accidental. Again use Figure 4 as an example. Phone number e_{10} 1-800-555-1212 might occur very frequently with "Amazon Customer Service". However, this is just by random association since this phone number, being the general toll free number for US, also appears with other companies frequently. It is thus important to make "calibration" to purify the association we get.
- R-*Discriminative*: Intuitively, entity instances matched on more popular pages should receive higher scores than entity instances from less popular pages. This characteristic is especially useful when the document collection is of varying quality, such as the Web.

3. CONCEPTUALLY: IMPRESSION MODEL

Toward a principled ranking model, we start with developing the insights– What is the *conceptual* model that captures the "ideal" behavior of entity search?

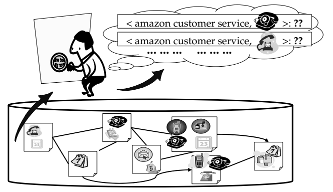

< amazon customer service, >: ??
< amazon customer service, >: ??
... ...

Figure 5: Impression Model: Conceptual Illustration

```
The Observer's Impression Model: Conceptual Execution of Entity Search.
Given:  $\mathcal{E} = \{E_1, \ldots, E_N\}$ over $\mathcal{D} = \{d_1, \ldots, d_n\}$. Let $E = E_1 \times \ldots \times E_m$.
Input:  $q = \alpha(E_1, \ldots, E_m, k_1, \ldots, k_l)$.
0: $\tau = 1$; /* time tick.
1: while ($\tau = \tau + 1$ and $\tau \leq T$):
3:     $d^\tau$ = access document from $\mathcal{D}$; /* access layer.
4:     $\forall t \in E$: $Score(q(t)|d^\tau)$ = recognize $p(q(t)|d^\tau)$; /* recognizition layer.
5: $\forall t \in E$: output $Score(q(t)) = \frac{\sum_{\tau=1}^{T} Score(q(t)|d^\tau)}{T}$; /* average impression.
```

Figure 6: Impression Model: Basic Framework.

3.1 Impression Model

To begin with, assuming no resource or time constraints, we speculate, what is an *ideal* realization of entity search, over the Web as D? To be concrete, consider query $Q1$ for finding tuple ⟨"amazon customer service", #phone⟩. As most users will probably do, we can access "amazon"-related pages (say, from a search engine), browse their contents, and follow links to read more, until we are satisfied (or give up), and returning our overall findings of promising #phone numbers.

Let's cast this process into an ideal execution, a conceptual model which we call the *Impression Model*, as Figure 5 shows. With unlimited time and resource, we dispatch an *observer* to repeatedly access the Web D and collect every evidence for substantiating any potential answer. This observer will visit as many documents and for as many times as he wishes. He will examine each such document d for any #phone that matches $Q1$ (*i.e.*, following and near "amazon customer service") and form his judgement of how good the matches are. With an unlimited memory, he will remember all his judgements– *i.e.*, his *impression*. The observer will stop when he has sufficient impression, according to which he will score and rank each phone entity instance that occurs in D.

To formalize this impression model, we take a probabilistic view to capture the observer's "impression." For a query q, given entity collection E over document collection D, Figure 6 sketches the impression framework. To execute entity search, at time τ, the observer *accesses* a document, which we denote d^τ, from D– Let's abstract this mechanism as the *access layer* of the observer. Examining this d^τ, he will *recognize* if any potential tuple t occurs there. Let's abstract this assessment function as the observer's *recognition layer*. Formally, this assessment results in the *association probability* $p(q(t)|d^\tau)$– *i.e.*, how likely tuple $q(t)$ holds true given the "evidence" of d^τ.

Eventually, at some time $\tau = T$, the observer may have sufficient "trials" of this repeated document visits, at which point his impression stabilizes (*i.e.*, with sufficient sampling from D). To capture this convergence statistically, let's characterize the access layer by $p(d)$, the *access probability* of document d, *i.e.*, how likely d may be drawn in each trial. Thus, over T trials, d will appear $T \times p(d)$ times. If T is sufficiently large, the average impression (*i.e.*, statistical mean) will converge– which we similarly refer to as the *association probability* of $q(t)$ over D.:

$$p(q(t)|D) = \lim_{T\to\infty} \frac{\sum_{\tau=1}^{T} p(q(t)|d^\tau)}{T} = \sum_{d \in D} p(d) \cdot p(q(t)|d) \quad (1)$$

As this association probability characterizes how likely t forms a tuple matching $q(t)$, when given the entire collection D as evidence, it is the "query score" we are seeking. While we will (in Section 4) further enhance it (with hypothesis testing), for now, we can view it as the final query score, *i.e.*,

$$Score(q(t)) = p(q(t)|D).$$

The impression model provides a conceptual guideline for the entity search task, by a tireless observer to explore the collection for all potential entity tuples. While the framework is conceptually ideal, all the key component layers remain open. We start with a "naive" materialization to motivate our full design.

3.2 Baseline: Naive Observer

As a first proposal, we develop the impression model with a simple but intuitive observer behavior, which uniformly treats every document in D and check if all entities and keywords are present. The final score is the aggregation of this simple behavior.

• *Access Layer*: The observer views every document equally, with a uniform probability $p(d) = \frac{1}{n}, \forall d \in D$ (recall that $|D| = n$).

• *Recognition Layer*: The observer assesses $p(q(t)|d)$ simply by the document "co-occurrence" of all the entity instances e_i and keywords k_j specified in $q(t)$: $p(q(t)|d)$ = 1 if they all occur in d; otherwise 0.

• *Overall*: Filling the details into Eq. 1, we derive the score, or the expected impression, of a candidate tuple t as follows:

$$Score(q(t)) = \sum_{d \in D} \frac{1}{n} \cdot \begin{cases} 1 & \text{if } q(t) \in d \\ 0 & \text{otherwise} \end{cases} = \frac{1}{n} C(q(t)), \quad (2)$$

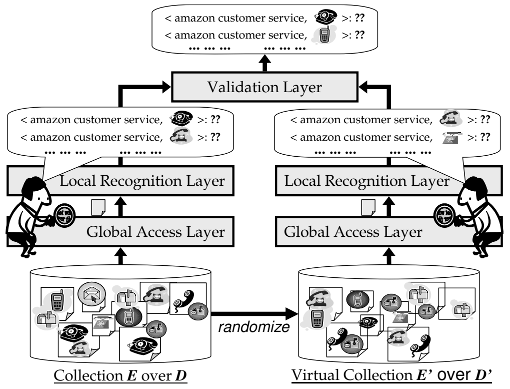
where $C(q(t))$ is the *document co-occurrence frequency* of $q(t)$, *i.e.*, the number of documents d in D such that $q(t) \subseteq d$.

The naive impression model, while simple, intuitively captures the spirits of entity search– that of identifying entities from each documents *locally*, by the recognition layer, and aggregates across the entire collection *globally*, by the access layer. Overall, the naive approach results in using co-occurrence frequency of entities and terms as the query score of tuple t– a simple but reasonable first attempt.

As a starting point, to build upon the naive observer for a full realization of the ideal impression model, we ask: What are its limitations? As our checklist, we examine the five requirements as outlined in Section 2. The naive observer does meet the *holistic* requirement, as it aggregates the impressions across all documents– which is the essence of the impression model (and thus every materialization will satisfy). Systematically over the requirement list, we identify three limitations.

Limitation 1: *The access layer does not discriminate sources.* As R-*Discriminative* states, not all documents, as sources of information, are equal. However, with a uniformly-picking access layer, the naive observer bypasses R-*Discriminative*. It will thus not be able to leverage many document collections where intrinsic (*e.g.*, link or citation structure) or extrinsic structures (*e.g.*, user rating and tagging) exist to discriminate documents as sources of information. How to enhance the access layer, so that documents are properly discriminated?

Limitation 2: *The recognition layer is not aware of entity uncertainty and conceptual patterns.* As R-*Uncertain* and R-*Contextual* mandate, entity instances are not perfectly extracted, and their matching with the query depends on keywords and other entities in the surrounding context. However, with an occurrence-only recognition layer, the naive observer does not respect either requirements. How to enhance the recognition layer, so that the tuple probabilities at each document are effectively assessed?



< amazon customer service, >: ??
< amazon customer service, >: ??
Validation Layer
< amazon customer service, >: ??
< amazon customer service, >: ??
< amazon customer service, >: ??
< amazon customer service, >: ??
Local Recognition Layer
Local Recognition Layer
Global Access Layer
Global Access Layer
randomize
Collection E over D
Virtual Collection E' over D'

Figure 7: Impression Model: Complete Framework

Limitation 3: *A validation layer is lacking.* As R-*Associative* states, our search should distinguish "intended" association of tuples from those "accidental" ones. The naive observer, however, believes in whatever "impression" he saw from the documents, and thus can be fragile regarding R-*Associative*. As we take a statistical view of the impression, we should equip the observer with statistical validation of his impression– or his "hypothesis"– and assess its significance. Our impression model, and therefore the naive observer, is missing such a critical *validation layer*.

4. CONCRETELY: *EntityRank*

Upon the basic impression model (and the naive materialization), we now fully develop our entity-search scheme: *EntityRank*.

Motivated by the limitations of the basic model, we begin with presenting the complete *impression model* as Figure 7 shows. The full model completes the initial sketch in Figure 5 in two aspects: First, we add a "virtual observer" (at the right side), who will perform the same observation job, but now over a "virtual" collection D' as a randomized version of D. Second, we add a new validation layer to validate the impression of the real observer (over D) by comparing it with that of the virtual observer (over D'). Overall, our impression model consists of three layers:

- (Section 4.1) As Limitation 1 motivated, the *access layer* defines how the observer picks documents, and is thus responsible for *globally* aggregating tuple scores across the entire collection.
- (Section 4.2) As Limitation 2 motivated, the *recognition layer* defines how the observer examines a document, and thus considers *locally* assessing tuple probabilities in each document visited.
- (Section 4.3) As Limitation 3 motivated, the *validation layer* statistically validates the significance of the 'impression" by comparing it with the null hypothesis from the virtual observer.

This section will concretely materialize the three layers, to develop our *EntityRank* scheme. Figure 8 summarizes *EntityRank*. Overall, Equation (1) gives $Score(q(t))$, the scoring function of how tuple t matches query q. Refer to our search task as Figure 3 defines, this scoring function determines the results of entity search. While we will develop step by step, as a road map, our objective is to, *first*, derive $p(q(t)|D)$ by materializing Eq. 1– which requires that we concretely define $p(d)$ and $p(q(t)|d)$. Sections 4.1 and 4.2 will develop these two parts respectively, which will together combine into $p(q(t)|D)$ given in Figure 8; we call this the *observed* probability, or p_o for short. Section 4.3 will similarly derive the same probability for the random collection, which we call the *random* probability and denote p_r. *Second*, the final score $Score(q(t))$ is determined by comparing p_o and p_r, in terms of hypothesis testing, which Section 4.3 will study.

While the *EntityRank* scheme satisfies all our "semantic" requirements of entity search– Is it admissible to efficient imple-

• **Query**: $q(\langle E_1, \ldots, E_m\rangle) = \alpha(E_1, \ldots, E_m, k_1, \ldots, k_l)$ **over** $\mathcal{D}$
• **Result**: $\forall\ t \in E_1 \times \cdots E_m$: Rank all t by computing $Score(q(t))$ as follows.

$$\textbf{(1)}\quad Score(q(t)) = p_o \cdot \log \frac{p_o}{p_r}, \textbf{where}$$

$$\textbf{(2)}\quad p_o \equiv p(q(t)|D) = \sum_{d \in D} \mathbf{PR}[d] \times \max_{\gamma}(\prod_{e_i \in \gamma} e_i.conf \times \alpha_B(\gamma) \times p(s|\gamma))$$

$$\textbf{(3)}\quad p_r \equiv p(q(t)|D') = \prod_{j=1}^{m}(\sum_{e_j \in d, d \in D} p(d)) \times \prod_{i=1}^{l}(\sum_{k_i \in d, d \in D} p(d)) \times \prod_{j=1}^{m} \overline{e_j.conf} \times \frac{\sum_s p(q(t)|s)}{|s|}$$

Figure 8: ***EntityRank*: The Scoring Function**

mentation? After all, we are pursuing entity search in the context of building a novel search engine, and efficiency is crucial for any online interactive search. Although query optimization techniques are beyond the focus of this paper, we summarize our overall *EntityRank* algorithm and its implementation strategies in Section 4.4.

4.1 Access Layer: Global Aggregation

The *access layer* defines how the observer selects documents, and is thus responsible for *globally* aggregating tuple scores across the entire collection. This layer must determine $p(d)$– how likely each document d will be seen by the observer (and thus how d will contribute to the global aggregation). As its objective (by R-*Discriminative*), this probability should discriminate documents by their "quality" as a source of information. The specific choice of an effective discriminative measure of quality depends on the document collection– what intrinsic structure or extrinsic meta data is available to characterize the desired notion of quality.

In the context of the Web, which is our focus, as a design choice, we decide to adopt the common notion of *popularity* as a metric for such quality. As the Web is a hyperlinked graph, the popularity of a page can be captured as how a page will be visited by users traversing the graph. Upon this view, we can materialize the access layer by the *random walk* model, where $p(d)$ can be interpreted as the probability of visiting a certain page d in the whole random walk process. With its clear success in capturing page popularity, PageRank [6] is a reasonable choice for calculating $p(d)$ over the Web. That is, computing **PR** as the PageRank vector, as the overall result of the access layer, we have

$$p(d) = \mathbf{PR}[d]. \tag{3}$$

For this choice, we make two remarks: First, *like* in a typical search engine setting, this page discriminative metric can be computed *offline*. Second, *unlike* in a typical search engine, we are *not* using **PR** to directly rank result "objects" by their popularity. Instead, we are using these popularity values as a discriminative measure to distinguish each page as a *source* of information– For a tuple t, its score will aggregate, by the impression model (Eq. 1), over all the pages d_i it occurs, each weighted by this $p(d_i)$ value.

We stress that, while we implement the access layer with "popularity" and in particular adopt PageRank, there are many other possibilities. Recall that our essentially objective is to achieve source discrimination– so that "good" pages are emphasized. First, such discrimination is not limited to link-based popularity– *e.g.*, user or editorial rating, tagging and bookmarking (say, del.icio.us), and query log analysis. Second, such discrimination may even be query-specific and controlled by users– say, to focus entity search on a subset of pages. For instance, we may restrict query $Q2$ and $Q3$ to only pages within the *.edu* domain; or, we may want to execute $Q4$ only for pages from amazon.com and bn.com. Such *corpus restriction*– with its access focus, not only speeds up search but also isolate potential noises that may interfere with the results. In general, our notion of the global access layer is to capture all these different senses of source discrimination.

4.2 Recognition Layer: Local Assessment

The recognition layer defines how the observer examines a document d, and thus accounts for locally assessing tuple probabilities in each document visited. Given a document d, we are to assess how a particular tuple t=$\langle e_1, \cdots, e_m\rangle$ matches the query $q(\langle E_1, \ldots, E_m\rangle) = \alpha(E_1, \ldots, E_m, k_1, \ldots, k_l)$. That is, this layer is to determine $p(q(t)|\ d)$, *i.e.*, how likely tuple $q(t)$, in the form of $\alpha(e_1, \ldots, e_m, k_1, \ldots, k_l)$, holds true, given d as evidence. Consider $Q1$: Given d as the snippet in Figure 4, for t=$\langle e_1$="800-201-7575"$\rangle$, the question becomes asking: $p(Q1(t)|d)$ = p(ow(amazon customer service e_1) | d)=?

To begin with, we note that a query tuple $q(t) = \alpha(e_1, \ldots, e_m, k_1, \ldots, k_l)$ may appear in d in multiple occurrences, because each e_i or k_j can appear multiple times. For instance, in Figure 4, while e_1="800-201-7575" occurs only once, "amazon customer service" appears two times– so they combine into two occurrences. Let's denote such an occurrence of $(e_1, \ldots, e_m, k_1, \ldots, k_l)$ in document d as γ = $(o_1, \ldots, o_n)$, where $n = m + l$; each o_i is an *object occurrence*, *i.e.*, a specific occurrence of entity instance e_i or keyword instance k_j in d. *E.g.*, Figure 4 shows $(o_1, \ldots, o_4)$ as one occurrence for the above example.

For each potential tuple t, the recognition of the association probability $p(q(t)|d)$ must evaluate all its occurrences, and "aggregate" their probabilities– because each one contributes to supporting t as a desired tuple. For this aggregation, we take the maximum across all occurrences of t as its overall probability, *i.e.*,

$$p(q(t)|d) = \max_{\gamma \text{ is a tuple occurrence of } q(t)} p(\alpha(\gamma)) \tag{4}$$

With this aggregation in place, we now focus on the assessment of each occurrence $\gamma = \{o_1, \ldots, o_n\}$ for $p(\alpha(\gamma))$– *i.e.*, how this tuple occurrence matches the desired context pattern. This task involves two largely orthogonal considerations:

- Extraction uncertainty: For each o_i that represents an entity instance e_i– Is it indeed an instance of type E_i? As Section 2 discussed, after performing extraction on D to prepare our "entity view" E, this probability $p(e_j \in E_j|d)$ is recoded in $e_i.conf$.
- Association context: Do the occurrences of $o_1, \ldots, o_n$ *together* match the pattern α that suggests their association as a desired tuple for query q?

With the two orthogonal factors, given $e_i.conf$, we can readily factor out the extraction uncertainty, and focus on the remaining issue: defining $p_{context}$– how t matches q in the context of d.

$$p(\alpha(\gamma)) = (\prod_{e_i \in \gamma} e_i.conf) \times p_{context}(\alpha(\gamma))$$

To determine $p_{context}$, this *contextual analysis* primarily consists of two parts, boolean pattern analysis and fuzzy proximity analysis, as we motivated in Section 2. Boolean pattern analysis serves the purpose of instantiating tuples, after which fuzzy proximity analysis serves the purpose of estimating in-document association strength of tuples.

Context Operators α. We are to evaluate $p_{context}(\alpha(\gamma))$, to see how γ occurs in a way matching α, in terms of the surrounding context. Recall that, as Section 2 defined, in entity search, a query q specifies a context operator α, which suggests how the desired tuple instances may appear in Web pages.

As a remark, we contrast our usage of patterns in entity search with its counterparts in document search (e.g, current search engines). On the one hand, such pattern restriction is not unique in entity search. In typical document search, it is also commonly used– *e.g.*, a user can put "" around keywords to specify matching these keywords as a phrase. However, on the other hand, our entity search uses textual patterns in a rather different way– a tuple pattern α describes the possible appearance of the surrounding *context* of a desired tuple. In contrast, keyword patterns in document search are intended for the *content* within a desired document.

In this study, we treat a Web page as a linear sequence of words. As Section 2 mentioned, in the entity view, each entity or keyword occurrence o_i is extracted with its positions at $o_i.pos$. An operator shall match on the *order* or *proximity* of objects: Each operator α applies to match an occurrence γ, *i.e.*, of the form $\alpha(o_1, \ldots, o_m)$. We will explain a few operators: doc, phrase, uw, ow.

We stress that, the context matching operator is for users (end-users or applications) to specify how the desired tuple may appear in documents. As in any query language, it also involves the *trade-off* of simplicity (or ease of use) and expressiveness. We believe the exact balance of such tradeoff must depend on the actual application settings– Note that, while we develop it as a generic system, the entity search system can be deployed in a wide range of settings, such as a general search engine for end users or a specialized vertical application, as Section 1 motivated.

Therefore, we advocate a two-fold strategy to balance the trade-off, with a set of system *built-in* operators: On the one hand, for *simplicity*, while users may specify explicitly an α operator, the system must support a default when omitted (*i.e.*, order in our current system), as Section 1 shows (*e.g.*, ,$Q1$). On the other hand, for *expressiveness*, while an application may choose to use our supported operators, the system must support a *plug-in* framework for applications to define their own specific context patterns.

In what follows, we explain some supported operators, each in the form of $\alpha(o_1, \ldots, o_m)$. Our purpose is to demonstrate how an operator is defined, in order to be plugged into the recognition layer. Each operator consists of two parts (as R-*Contextual* states), *i.e.*, the *pattern* constraint α_B and *proximity* function α_P. Thus, we can further define our context probability (of Eq. 5) as

$$p_{context}(\alpha(\gamma)) \equiv \alpha_B(o_1, \ldots, o_m) \times \alpha_P(o_1, \ldots, o_m).$$

1. Boolean Pattern Qualification. As the first step, α will qualify, by a constraint α_B that returns 1 or 0 (*true* or *false*), whether some pattern constraint on the order and adjacency of o_i is satisfied.

- doc($o_1, \ldots, o_m$): objects o_i must all occur in the same document.
- phrase($o_1, \ldots, o_m$): objects o_i must all occur in exactly the same sequence (*i.e.*, order and adjacency) as specified.
- uw(n)($o_1, \ldots, o_m$): objects o_i must all occur in a window of no more than n words; n default as the document length.
- ow(n)($o_1, \ldots, o_m$): in addition to uw, o_i must all occur in the order.

2. Probabilistic Proximity Quantification. As the second step, the operator α will quantify, by a probability function α_P, how well the proximity between objects match the desired tuple– *i.e.*, how the objects' positions indicate their association as a tuple, once they are qualified (above). Essentially, each operator will thus define a probabilistic distribution that assesses such association probabilities given object positions.

We propose an intuitive and practical model, the *span proximity model*, to capture our basic intuition that the closer they appear to each other, the more likely they are associated with each other. While our system currently only supports the span proximity model, our experimental results show that it is effective for a wide range of scenarios.

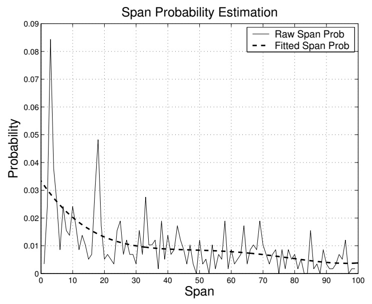


Figure 9: The Span Proximity Model

In the span model, we characterize the proximity strength of a tuple occurrence $\gamma = (o_1, \ldots, o_m)$ as depending on its *span* length– the shortest window that covers the entire occurrence, denoted by s. We define the context association probability of γ, *i.e.*, how e_i and k_j associate into a tuple, as solely determined by span.

$$\alpha_P(\gamma) \equiv p(\gamma \text{ is a tuple}|s), \text{or simply } p(\gamma\ |s).$$

Finally, we must estimate the distribution $p(\gamma|s)$. We can "learn" this distribution by collecting some true tuples (as our "labelled data")– *i.e.*, γ's that are really tuples. With sufficient examples, we can obtain an empirical distribution of, for real tuples, how their spans vary, *i.e.*, $p(s|\gamma)$. In our current implementation, we obtain our span distribution as Figure 9 shows, using labelled tuples that are company names and their phone numbers (*e.g.*, "IBM", "877-426-2223") from the Web corpus. Finally, by Bayes' theorem,

$$p(\gamma\ |s) = \frac{p(\gamma)}{p(s)} p(s|\gamma) \propto p(s|\gamma),$$

where we remove the prior probabilities $p(\gamma)$ and $p(s)$: Note that, for practical implementation, as the priors are hard to measure, we assume they are the same, for all different γ and s. Thus, these constant priors will not affect ranking of entities.

Putting together the derivations so far back to where we started, *i.e.*, Eq. 4, as the overall result of the recognition layer, we obtain:

$$p(q(t)|d) = \max_{\gamma} \prod_{e_i \in \gamma} e_i.conf \times \alpha_B(\gamma) \times p(s|\gamma) \qquad (5)$$

As a related approach, Chakrabarti et al. [11] propose a *discriminative* model to measure the association between an entity instance and keywords within a document. Our span proximity model is probabilistic– it bridges the proximity between objects to their probability of associating into a tuple. By relating each span with a probability, we can thus integrate our local recognition model with the global access model in a probabilistic framework seamlessly.

Ideally, it would be best to learn such a span model for each kind of query– because different types tuples may vary in their spans. However, it is impractical to do so due to the unpredictable and large number of different queries. It is possible to classify queries into different categories and learn a approximate span model for each category. In this paper, we approximate this by using the one span model we learned in Figure 9. Although our "span" model may not be as accurate as other more sophisticated local models, such as the one presented in [11], we believe it still captures the basic insights of a local recognition model.

We stress that our recognition layer is a general framework that can plug-in any pattern and proximity models. Our current choice of basic patterns and the span model is just one reasonable possibility. They are robust and easy to apply, which gives us fast online processing.

4.3 Validation Layer: Hypothesis Testing

As a new layer, the *validation layer* statistically validates the significance of the "impression." Our approach is by comparing it with the null hypothesis– simulating an observer over a virtual collection. We note that the construction of a random collection D' is only conceptual– it will derive a formula that can be efficiently computed (Section 4.4) without materializing D'.

Since our impression model dispatches the observer (Figure 7) to collect his impression of entity associations, we must ask– Are these associations significant? Recall from Eq. 1 that $p(q(t)|D)$ is an average impression, *i.e.*, the statistical mean, over D. By taking a probabilistic view, we are now equipped with principled statistical tools to "validate" if the impression of association, as our *hypothesis* to test, is significant. As the *null hypothesis*, we suppose the associations are "unintended"– *i.e.*, as the result of randomly associating entities and keywords in a randomized collection D'. By contrasting observed probability p_o with random probability p_r, we will obtain a final score $Score(q(t))$ that captures the significance of our hypothesis– *i.e.*, intended tuple association.

Null Hypothesis: Randomized Associations. As our null hypothesis, we suppose that the association of $t = (e_1, \ldots, e_m, k_1, \ldots, k_l)$ is simply accidental– instead of semantically intended as a tuple. To simulate the outcome of the null hypothesis, we must create D' as a randomized version of D. In comparison, D' should resemble D as much as possible, except that in D' the associations of keywords and entities are purely random. The creation of a random document d' in D' is as follows:

First, we will randomly draw entities and keywords into d'. Each e_i or k_j will be drawn independently, with a probability the same as that of appearing in any document of D. We thus preserve the individual entity/keyword probabilities from D, but randomize their associations. This probability can be derived as below, which ensures that the probability of observing a keyword/entity instance in the process of visiting the D' through a uniform access layer will be the same as observing it in the original D.

$$p(e_i \in d') = \textstyle\sum_{e_i \in d, d \in D} p(d); \quad p(k_j \in d') = \textstyle\sum_{k_j \in d, d \in D} p(d)$$

Further, if a keyword/entity instance is drawn to appear in virtual document d', its position in the document will also be randomly chosen. We also preserve the entity extraction probability of entity instances, by using the average extraction probability $\overline{e_j.conf}$ (the average over all the extraction probabilities of all the occurrences of entity instance e_j in D) of entity instance e_j as the extraction probability of all the occurrences of entity instance e_j in D'.

Supposing we construct D' with all such random documents d', we ask: What will be the "average impression" that our observer will perceive in this random collection? By Eq. 1, this average impression is the summation over all documents. We can simplify it by noting that 1) if $q(t)$ does not even appear in d', then $p(q(t)|d')$ = 0, 2) otherwise, $p(q(t)|d')$ has the same value– since we create all d' in exactly the same way. Consequently, we get:

$$\begin{aligned} p(q(t)|D') &= \textstyle\sum_{d' \in D' \textbf{and } q(t) \in d'} p(d') \times p(q(t)|d') \\ &= p(q(t)|d') \times \textstyle\sum_{d' \in D' \textbf{and } q(t) \in d'} p(d') \\ &= p(q(t)|d') \times p(q(t) \in d'). \end{aligned} \quad (6)$$

Here, $p(q(t) \in d')$ is the probability of t appearing in some d'. As keywords and entity instances are drawn independently into d', this probability is simply the product of the probabilities of each keyword and entity instances appearing in d', *i.e.*,

$$p(q(t) \in d') = \textstyle\prod_{j=1}^{m} p(e_j \in d') \prod_{i=1}^{l} p(k_i \in d')$$

Next, we derive the random association probability, $p(q(t)|d')$, of tuple t in document d'. As we discussed in Section 4.2, it is the product of entity probability and contextual probability:

$$p(q(t)|d') = (\textstyle\prod_{j=1}^{m} \overline{e_j.conf}) \times p_{context}(q(t)|d').$$

The contextual probability $p_{context}(q(t)|d')$ is dependent on the span of tuple t, as Section 4.2 discussed. As keywords and entity instances appear randomly in d', the possible spans of tuple t should also appear randomly. This means different span values are equally likely. Thus, we can use the average of all the span probabilities (*e.g.*, as Figure 9 shows) for this probability estimation:

$$p_{context}(q(t)|d') = \overline{p}(q(t)|s) = \frac{\sum_s p(q(t)|s)}{|s|},$$

where $|s|$ refers to the number of distinct span values.

Putting together these derivations back to Eq. 6, we will obtain $p(q(t)|D')$, which Figure 8 shows as the random probability p_r.

Hypothesis Testing: G-Test. To judge if the association of t is statistically significant, we should compare the *observed* p_o versus the *random* p_r, which we have both derived in Figure 8. We can use standard statistics testing, such as mutual information, χ^2, G-test [16], *etc.*. Our implementation adopts G-test to check whether p_o indicates random association or not,

$$Score(q(t)) = 2(p_o log \frac{p_o}{p_r} + (1 - p_o) log \frac{1 - p_o}{1 - p_r}) \quad (7)$$

To interpret, we note that the higher the G-test score is, the more likely entity instances t together with keywords k are truly associated. We take this score for the final ranking of the tuples. In practice, given a large web page collection containing many keywords and entity instances, the chance that an entity instance occurs in the collection is extremely small. Hence, $p_o, p_r \ll 1$, Eq. (7) can be simplified as:

$$Score(q(t)) \propto p_o \cdot log \frac{p_o}{p_r} \quad (8)$$

4.4 EntityRank: Implementation Sketch

We have developed *EntityRank*, as a concrete materialization of the impression model, and it satisfies all our requirements (Section 2) of entity search– but, can it be efficiently implemented for online interactive search– at a speed comparable to current page-oriented search engines? Our analysis of the scheme (Figure 8) shows that, in fact, the *EntityRank* framework can be easily built upon current engines, thus immediately leveraging the impressive infrastructure and scalable efficiency already in place. While this paper focuses on the ranking scheme, and not query processing, we show how *EntityRank* can be efficiently realized.

Why Feasible? Let's examine how we may realize *EntityRank*. In Figure 8, $Score(q(t))$ has two components, p_o and p_r:

First, p_r can mostly be pre-computed off-line (before query), and thus its cost is negligible. There are four factors: Factors 1 and 2 are the "document frequencies" of entities e_j and keywords k_i. We can pre-compute them for every term; when the query arrives, we simply lookup in a table. We can similarly handle factors 3 and 4.

Second, p_o boils down to "pattern matching," which is a major function of today's page-based search engine. The first factor requires PageRank, which can be done off-line. The second factor requires matching specific tuple occurrences γ (Section 4.2), which can only be executed when the query terms (*e.g.*, "amazon" and #phone) and patterns (*e.g.*, ow) are specified. Nevertheless, such pattern matching is well supported in current engines, by using inverted lists– our realization can build upon similar techniques.

Possible Implementation. To begin with, we assume that a document collection D has been transformed by way of entity extraction into an entity collection E with entities $\{E_1, E_2, \dots, E_n\}$.

• *Indexing:* To support entity as a first-class concept in search, we index entities in the same way as indexing keywords.

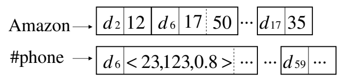

Amazon
d_2 12 d_6 17 50 ··· d_{17} 35
#phone
d_6 < 23,123,0.8 > ··· ··· d_{59} ···

Figure 10: A Snippet of Index.

We use the standard inverted index for indexing keywords. We index the extracted entities similar to keywords. To index an entity type E_i, our system will produce a list containing all the information regarding the extracted instances of E_i. Specifically, the list records for each occurrence of entity instance e_i, the position $e_i.pos$ of the extraction in the documents, *e.g.* position 23 at document d_6, the entity instance ID $e_i.id$, *e.g.* ID 123 for representing phone number "805-213-4545", and the confidence of extraction $e_i.conf$, *e.g.* 0.8. All the occurrence information of entity instances of a certain entity type is stored in an ordered list according to document number as shown in Figure 10.

• *Search:* Figure 11 shows the pseudo code of our *EntityRank* algorithm for supporting entity search.

Let's walk through the algorithm for query "ow(amazon #phone)" upon the index in Figure 10. We first load the inverted list for keyword "Amazon" and entity type "#phone" (line 0). Then we iterate through the two lists in parallel , checking the intersection of documents (line 1). In this example, the algorithm will first report d_6 as the intersecting document. Then the algorithm will further check the postings of keywords and entity types to see if the specified query pattern "ow" is satisfied in d_6 (line 2). In this case, entity instance with ID "123" at position 23 is matched with keyword "Amazon" at position 17 (not position 50). A tuple of entity instance with ID "123" is therefore instantiated, and its local association probability will be calculated according to the local recognition layer 4.2 (line 3 and 4). All the instantiated tuples and their local scores are stored for later on aggregation and validation purposes. Once the parallel scan of lists ends, we can sum up all the local scores (multiplied by the popularity of document) into forming the observed average association probability for each tuple t (line 6). As just discussed, the cost of p_r is negligible.

Observe that the core of our *EntityRank* algorithm (line 1-4) is essentially sort-merge-join of ordered lists. This only requires scanning through all the lists once, in parallel. Therefore, the algorithm is linear in nature and could be run very efficiently. Moreover, this sort-merge-join operates on a document basis. This implies that this procedure (line 1-4) can be fully parallelized, by partitioning the collection into sub-collections. This parallelism allows the possibility of realizing entity search in very large-scale, supported by a cluster of nodes. Our prototype system, to be discussed in Section 6.3, leverages such parallelism on a cluster of 34 nodes.

```
The EntityRank Algorithm: Actual Execution of Entity Search.
Given:  $L(E_i), L(k_j)$: Ordered lists for all the entity and keywords.
Input:  $q = \alpha(E_1, \dots, E_m, k_1, \dots, k_l)$.
0: Load inverted lists: $L(E_1), \dots, L(E_m), L(k_1), \dots, L(k_l)$ ;
   /* intersecting lists by document number
1: For each doc $d$ in the intersection of all lists
2:     Use pattern $\alpha$ to instantiate tuples; /* matching
3:     For each instantiated tuple $t$ in document $d$
4:         Calculate $p(q(t)|d)$ ; /* Section 4.2
5: For each instantiated tuple $t$ in the whole process
6:     calculate $p(q(t)|D) = \sum_d p(q(t)|d)p(d)$; /* observed probability
7:     output $Score(\mathrm{q(t)}) = p(q(t)|D) \log \frac{p(q(t)|D)}{p(q(t)|D')}$; /* Section 4.3
```

Figure 11: The *EntityRank* Execution Framework.

5. RELATED WORK

The objective of our system is to build an effective and efficient entity-aware search framework, with a principled underlying ranking model, to better support many possible search and mining applications. Our previous work [12] formulates the problem of entity search and emphasize its application on information integration (*i.e.*, Scenario 4 in Section 6.2), while this work focuses on the core entity ranking problem. To the best of our knowledge, we did not witness any similar exploration of combining local analysis, global aggregation and result verification in a principled way, especially over a network of documents of varying quality. Our work is related with the existing literature in three major categories: information extraction (*i.e.*, IE), question answering (*i.e.*, QA) and emerging trend on utilizing entity and relation for various search tasks. We now study these categories one by one.

First, our system on one hand relies on IE techniques to extract entities; on the other hand, our system could be regarded as online relation extraction based on association. There have been many comprehensive IE overviews recently ([14], [15], [4]) summarizing the state of the art. On the special Web domain, there have been many excellent IE works (*e.g.*, SemTag [17], KnowItAll [18], AVATAR [19]) Furthermore, many open source frameworks that support IE (*e.g.*, GATES [1], UIMA [2]) are readily available. While most IE techniques extract information from single documents, our system discovers the meaningful association of entities holistically over the whole collection.

Second, our system can be used as a core component to support QA more directly and efficiently. Unlike most QA works (*e.g.*, [3], [5], [21], [22]), which retrieve relevant documents first and then extract answers, our work directly builds the concept of entity into the search framework. While many QA systems' focus is on developing interesting QA system framework, most of them have adopted simple measures for ranking and lack a principled conceptual model and a systematic study of the underlying ranking problem. The SMART IR system [3] and the AskMSR QA system [5] mainly use the entity occurrence frequency for ranking. The Mulder system [21] ranks answer candidates mainly according to their closeness to keywords, strengthened by clustering similar candidates for voting. The Aranea system [22] mainly uses the frequency of answer candidates weighted by idf of keywords in the candidates as the scoring function.

Finally, we are recently witnessing an emerging research trend towards searching with entity and relationship over unstructured text collection (such as [10] and [24] advocate).

Towards enriching keyword query with more semantics, AVATAR [20] semantic search tries to interpret keyword queries for the intended entities and utilize such entities in finding documents.

Towards searching over fully extracted entities and relationships from the Web, ExDB [8] supports expressive SQL-like query language over an extracted database of singular objects and binary predicates, of the Web; Libra [23] studies the problem of searching web objects as records with attributes. Due to the different focus on information granularity, its language retrieval model is very different from ours. While these approaches rely on effective entity and relationship extraction for populating an extraction database, our approach only assumes entity level extraction and replies on large-scale analysis in the ranking process.

Towards searching over typed entities in or related with text documents, BE [7] develops a search engine based on linguistic phrase patterns and utilizes a special index for efficient processing. It lacks overall system support for general entity search with a principled ranking model. Its special index, "neighborhood index", and query language, "interleaved phrase" query, are limited to phrase queries only; Chakrabarti et al. [11] introduce a class of text proximity queries and study scoring function and index structure optimization for such queries. Its scoring function primarily uses local proximity information, whereas we investigate effective global aggregation and validation methods, which we believe are indispensable for robust and effective ranking in addition to local analysis. Our query primitive is also more flexible in allowing expressive patterns and multiple entities in one query; ObjectFinder [9] views an "object" as the collection of documents that are related with it and therefore scores an "object" by performing aggregation over document scores. In contrast, our approach views an entity tuple as all its occurrences over the collection. Therefore, its score aggregates over all its occurrences, where we consider uncertain, contextual factors other than the document score.

6. PROTOTYPE AND EXPERIMENTS

This section first discusses the prototype system we built for supporting entity search. Then we describe a few applications that could be easily built upon on system, which clearly show the usefulness of entity search qualitatively. Finally, we use our large-scale experiment to quantitatively verify the effectiveness of our ranking algorithm *EntityRank*, as compared to other ranking schemes.

6.1 System Prototype

We build our entity search system upon the Lemur IR Toolkit. We mainly morphed the indexing module of the Lemur Toolkit to be able to index entities in addition to keywords and implemented our own query modules for supporting *EntityRank*.

For getting data from the Web, we obtained our corpus from the Stanford WebBase Project, as our "virtual" Web Crawler.

For entity extraction, we have implemented two types of entity extractors. First, our *rule-driven* extractor tags entities with regular patterns– *e.g.*, #phone entity and #email entity, etc. Second, our *dictionary-driven* extractor works for entities whose domains, or *dictionaries* are enumerated– *e.g.*, #university entity, #professor entity, #research entity(as areas in CS), etc.

For more details regarding the system, please refer to our work [13] on the overall system architecture for entity search.

6.2 Qualitative Analysis: Case Studies

This subsection studies some of the possible applications that could be built upon entity search to show its promise qualitatively.

Question Answering: Scenario 1

Entity search could be a good candidate as the core search component for supporting question answering tasks. By using existing techniques, such as removing stop words, identifying the entity type for the question, we could effectively turn a lot of questions into entity search queries supported by our system.

Towards this goal, we built a yellowpage application with the following setting:

- Entity collection: $E = \{$#phone, #email$\}$
- Document collection: $D =$ the Web

Such a system addresses our motivating query $Q1$, finding the Costumer Service number of Amazon, in Section 1.

Relation Discovery: Scenario 2&3

Our query primitive and ranking algorithm for entity search could afford more than one entity at a time. Such queries, containing multiple entities, could be viewed as relational discovery queries.

Chris Clifton	Purdue Univ	Data Mining
Sunil Prabhakar	Purdue Univ	Database Systems
Jiawei Han	UIUC	Data Mining
David J. Dewitt	Univ of Wisc	Database Systems

Table 1: Profs in DB-related Areas (Partial)

sigmod04-040611.pdf	sigmod04-040617.ppt
URL: http://www.cs.ucsb.edu/∼su/tutorials/sigmod2004.html	
publications/tr-db-01-02.pdf	publications/sigmod2001.ppt
URL: http://www.ics.uci.edu/∼iosif/index.html	

Table 2: Pairs of PDF and PPT Files for SIGMOD (Partial)

Towards this goal, we built an application on Computer Science domain with the following setting:

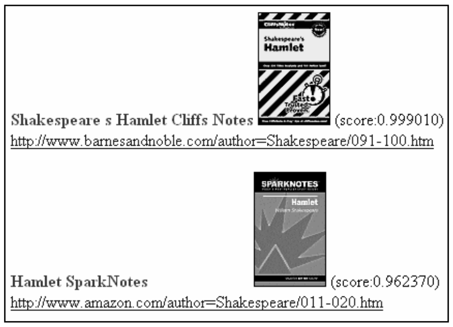
- $E = \{$#professor, #research, #university, #pdf_file, #ppt_file,#phone, #email$\}$
- $D =$ a collection of computer science related webpages

This application could answer user queries $Q2$ and $Q3$ we raised in scenario 2 and 3 in Section 1. Partial results are shown in Table 1 for query $Q2$ and in Table 2 for query $Q3$ respectively.

Information Integration: Scenario 4

Entity search could also be a good candidate for supporting adhoc on-the-fly information integration, whose goal is to assemble different attributes (entities) together into one relation.

Towards this goal we built an online book shopping application based on the query results returned from multiple deep web sources in the Book Domain.



Shakespeare s Hamlet Cliffs Notes (score:0.999010)
http://www.barnesandnoble.com/author=Shakespeare/091-100.htm

Hamlet SparkNotes (score:0.962370)
http://www.amazon.com/author=Shakespeare/011-020.htm

Figure 12: Images of Books with "Hamlet" in Title (Partial)

- $E = \{$#title, #author #date, #price, #image$\}$
- $D =$ result pages returned from deep Web sources regarding book queries

Users can ask queries regarding possible combinations of keywords and the entity types. Figure 12 shows the result for a query that tries to find images of books with keyword "Hamlet" in title, motivated by query $Q4$ we discussed in Section 1.

For all the four scenarios, we have built applications with different datasets and witnessed great usefulness of entity search in all of them. In particular, we will systematically evaluate Scenario 1 because it has a large realistic corpus. For all other scenarios, the results were also clearly promising: For instance, for queries finding relations $\langle professor, research\rangle$ and $\langle professor, email\rangle$, when counting the top-match research area and email for each professor, the result achieves between 80% - 90% precision and recall.

6.3 Quantitative Systematic Evaluation

In order to demonstrate the effectiveness of our ranking algorithm *EntityRank* for supporting entity search, we have build a large scale, distributed system on a real Web corpus. In this subsection, we will first briefly discuss the setup of our system. Then, we will use typical query sets to show the accuracy of our ranking model over other ranking schemes.

6.3.1 Experiment Setup

To empirically verify that our ranking model is effective for supporting entity search, we decide to use the Web, the ultimate information source, as our corpus. Our corpus, a general Web crawl in Aug, 2006, is obtained from the WebBase Project. The total size is around 2TB, containing 48974 websites and 93 million pages.

To process such terabyte-sized data set, we ran our indexing and query processing modules on a cluster of 34 machines, each with Celeron 2.80GHz CPU, 1 GB memory and 160 GB of disk. We evenly distribute the whole corpus across these nodes.

On this corpus, we target at two entity types: phone and email. They are extracted based on a set of regular expression rules. We extracted around 8.8 million distinctive phone entity instances and around 4.6 million distinctive email entity instances.

6.3.2 Accuracy Comparison

To get a first feeling of our ranking model, we tested a few finding-phone-number and finding-email-address queries, that are similar to our motivating scenario 1 in Section 1.

In addition to analyzing the results returned by *EntityRank*, we also try to compare our results with the following five approaches:

- N (Naive Approach): Tuples are ranked according to the percentage of documents in which they are matched.
- L (Local Model Only Approach): Tuples are solely ranked using their highest observed local association probability.
- G (Global Aggregation Only Approach): Tuples are ranked according to the summation of the score of documents in which they are matched. Pagerank score is used as the document score.
- C (Combination of Local Model and Global Aggregation Approach): Tuples are ranked according to the summation of the local association probabilities from matched documents.
- W (*EntityRank* Without G-test): Tuples are ranked according to their observed association probability, without performing the G-test for validation. The purpose of testing this approach is to see the effect of hypothesis test in the ranking framework.

Query	Telephone	ER	L	N	G	C	W
Citibank Customer Service	800-967-2400	**1**	4	7	43	**1**	**1**
New York DMV	800-342-5368	**2**	**2**	213	882	5	3
Amazon Customer Service	800-201-7575	**1**	**1**	52	83	**1**	**1**
Ebay Customer Service	888-749-3229	**1**	7	859	118	2	13
Thinkpad Customer Service	877-338-4465	5	12	249	127	19	**4**
Illinois IRS	800-829-3676	**1**	**1**	157	697	3	2
Barnes & Noble Customer Service	800-422-7717	**1**	2	2158	1141	7	**1**

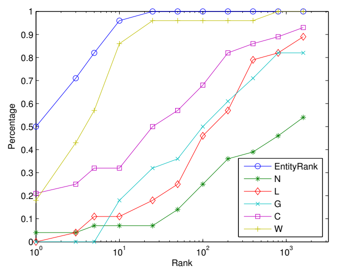
Figure 13: Telephone Number Queries

The results of executing motivating queries using different ranking models are shown in Figure 13 and 14. The first column lists keywords used in the query. The second column lists the correct entity instance returned for each query (manually verified). The third, fourth, fifth, sixth and seventh columns list the rank of the correct phone number in results by the various approaches described above respectively.

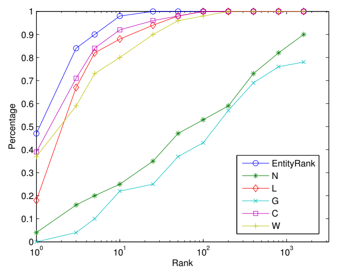
As we can see, *EntityRank* (ER) consistently outperforms other ranking methods. Almost all the right answers are returned within top 3 for finding phone numbers and more than half of the right answers are returned within top 4 for finding email addresses.

Query	Email	ER	L	N	G	C	W
Bill Gates	bgates@microsoft.com	**4**	44	2502	376	21	23
Oprah Winfrey	oprah@aol.com	**2**	6	745	80	4	3
Elvis Presley	elvis@icomm.com	**5**	56	1106	267	20	8
Larry Page	larrypage@google.com	**8**	24	9968	26932	12	11
Arnold Schwarzenegger	governor@governor.ca.gov	**4**	45	165	169	5	6

Figure 14: Email Queries



(a) Query Type I



(b) Query Type II

Figure 15: Satisfied Query Percentage under Various Ranks

To study the performance of our method in a more systematical way, we use the following ways to come up with typical queries for each query types. **Query Type I (phone)**: We use the names of top 30 companies in Fortune 500, 2006 as part of our query, together with phone entity type in the query. **Query Type II (email):** We use the names of 88 PC members of SIGMOD 2007 as part of our query, together with email entity type in the query. 37 out of the 88 names that don't have any hit with any email entity instance are excluded. This is due to the reason that our corpus is not complete (2TB is only a small proportion of the whole Web).

To measure the performance, we use the mean reciprocal rank (MRR) as our measure. This measure essentially calculates the mean of the inverse ranks of the first relevant answer according to queries. The value of this measure lies between 0 and 1, and the higher the better. As it is time consuming to manually check all the returned results to identify the first relevant tuple in each result, especially given the fact that in lots of the results the first related entity tuple appears very high in rank. We come up with the following two approximate methods for estimating the rank where the first related entity tuple is returned.

In evaluation method 1, we first manually go through the result returned by our *EntityRank* algorithm, finding the first related tuple (usually within top 5). Then, we look up the place where this tuple appears using other ranking algorithms. Although this method is biased towards our *EntityRank* algorithm, it still makes sense as intuitively the related tuple (manually verified) returned by *EntityRank* should also be ranked high using other methods. At minimum, this evaluation method gives a meaningful lower bound of the MRR, referred as $\lfloor MRR \rfloor$, of the first relevant tuple.

Evaluation method 2 is a much more conservative evaluation method. We manually check the top 10 entries of the results returned by each ranking algorithm. If a relevant tuple is found within top 10 entries, we will record the rank, otherwise, we will just use rank 10. The intuition of using this method is that normally the top 10 results (in the first pages) are most important for users. This evaluation method gives a meaningful higher bound of the MRR, referred as $\lceil MRR \rceil$, of the first relevant tuple.

Figure 15 shows the percentage of queries returning relevant answers within various top K ranks for the two query types respectively. The x axis represents various top K ranks, ranging from 1 to

1600 in log scale. The y axis reports the percentage of the tested queries returning relevant answers under a certain rank. Evaluation method 1 is used for getting the rank of relevant answers of queries.

Table 3 and 4 give comparison of the six ranking algorithm, on Query Type I and II respectively using $\lfloor MRR \rfloor$ and $\lceil MRR \rceil$.

Measure	*EntityRank*	L	N	G	C	W
$\lfloor MRR \rfloor$	**0.648**	0.047	0.037	0.050	0.266	0.379
$\lceil MRR \rceil$	**0.648**	0.125	0.106	0.138	0.316	0.387

Table 3: Query Type I MRR Comparison

Measure	*EntityRank*	L	N	G	C	W
$\lfloor MRR \rfloor$	**0.659**	0.112	0.451	0.053	0.573	0.509
$\lceil MRR \rceil$	**0.660**	0.168	0.454	0.119	0.578	0.520

Table 4: Query Type II MRR Comparison

As we can see from all the results, our *EntityRank* algorithm is highly effective with MRR around 0.65, outperforming all the other algorithms. "Ordered Window" pattern "ow" is used in our *EntityRank* algorithm for evaluating those queries.

Although the focus of this paper is on the effectiveness of entity search, we have also carried out some preliminary study on the efficiency of our system, in terms of space and time.

First, in terms of space consideration, supporting entity search only incurs minimal index overhead. Our current entity search system, by indexing email and phone entities, only incurs less than 0.1% overall space overhead in the size of indices.

Second, in terms of time consideration, our system adds negligible overhead in offline indexing time and performs online entity search rather efficiently. Indexing entities could be done at the same time as we index keywords. Similar to the space overhead it creates, the time overhead in indexing entities in addition to keywords is almost negligible. For online query processing, as the nature of the query processing is linear as discussed in Section 4.4, query processing is rather efficient. For our examples queries listed in Figure 13 and 14, the average query response time is 2.45 seconds.

6.3.3 Observations and Discussions

We now analyze why the other five approaches perform not as well as our *EntityRank* algorithm.

In the local model only approach (L), as long as there is some false association where keywords and entities appear very close to each other, the tuple will be ranked high. Our global aggregation of the local scores is more robust, as the results are collective from many sources across the web and such false association is not likely to appear in lots of web pages. The results for the local model only approach may get improved by having more accurate local models, however, it doesn't solve the problem from the root as we analyzed. It is necessary to conduct global aggregation upon local analysis.

On the other extreme, using pure global aggregation without any local constraint, *e.g.* the N and G ranking methods, performs poorly as our result shows. This is because without local constraint, lots of false associations will be returned, which leads to the aggregation of false information. Local model helps in reducing such noises, generating high quality information for later on aggregation.

Experiments show that both a simple combination of the local scores with global aggregation and performing *EntityRank* without hypothesis testing perform worse than *EntityRank*. This validates our analysis on the important factors for entity search in that the lacking of any factor, in this case the discriminative and associative factors, would result in significant reduction in effectiveness.

Finally, we would like to point out an intriguing merit of *EntityRank* in that it performs holistic analysis , by leveraging the scale of the corpus, together with effective local analysis. Therefore, the larger the corpus size, the more information (evidence) we can aggregate, and therefore the more effective the ranking. Other simple methods, especially the local model only approach, may suffer from such situations as the larger the corpus the more the noise.

7. CONCLUSIONS

In this paper, we propose the notion of entity search. Toward building such a system, we address the central challenge of entity ranking. Our solution, the *EntityRank* model, seamlessly integrate local recognition and global access information into a probabilistic view of estimating tuple likelihood of results. Our experiments show that the concept of entity search is useful, and the *EntityRank* model is effective for finding good results in top ranks.

8. REFERENCES

[1] Gate - general architecture for text engineering,.

[2] Unstructured information management architecture.

[3] S. Abney, M. Collins, and A. Singhal. Answer extraction. In *ANLP*, 2000.

[4] E. Agichtein and S. Sarawagi. Scalable information extraction and integration (tutorial). In *SIGKDD*, 2006.

[5] E. Brill, S. Dumais, and M. Banko. An analysis of the askmsr question-answering system. In *EMNLP*, 2002.

[6] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *WWW*, pages 107–117, 1998.

[7] M. Cafarella and O. Etzioni. A search engine for large-corpus language applications. In *WWW*, 2005.

[8] M. Cafarella, C. Re, D. Suciu, and O. Etzioni. Structured querying of web text data: A technical challenge. In *CIDR*, 2007.

[9] K. Chakrabarti, V. Ganti, J. Han, and D. Xin. Ranking objects based on relationships. In *SIGMOD*, 2006.

[10] S. Chakrabarti. Breaking through the syntax barrier: Searching with entities and relations. In *ECML*, pages 9–16, 2004.

[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] T. Cheng and K. C.-C. Chang. Entity search engine: Towards agile best-effort information integration over the web. In *CIDR*, pages 108–113, 2007.

[13] T. Cheng, X. Yan, and K. C.-C. Chang. Supporting entity search: A large-scale prototype search system. In *SIGMOD*, 2007.

[14] W. Cohen. Information extraction (tutorial). In *SIGKDD*, 2003.

[15] A. Doan, R. Ramakrishnan, and S. Vaithyanathan. Managing information extraction: state of the art and research directions (tutorial). In *SIGMOD*, 2006.

[16] T. Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19, 1994.

[17] S. D. et al. SemTag and Seeker: Bootstrapping the semantic web via automated semantic annotation. In *WWW*, 2003.

[18] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in knowitall. In *WWW*, 2004.

[19] T. S. Jayram, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar information extraction system. *IEEE Data Eng. Bull.*, 29(1):40–48, 2006.

[20] 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.

[21] C. C. T. Kwok, O. Etzioni, and D. S. Weld. Scaling question answering to the web. In *WWW*, pages 150–161, 2001.

[22] J. J. Lin and B. Katz. Question answering from the web using knowledge annotation and knowledge mining techniques. In *CIKM*, pages 116–123, 2003.

[23] Z. Nie, Y. Ma, S. Shi, J.-R. Wen, and W.-Y. Ma. Web object retrieval. In *WWW*, 2007.

[24] G. Weikum. DB&IR: both sides now. In *SIGMOD*, 2007.