Finding Relevant Information of Certain Types from Enterprise Data

Xitong Liu
Dept of Electrical and Computer Engineering
University of Delaware, DE, USA
xliu@ece.udel.edu

Hui Fang
Dept of Electrical and Computer Engineering
University of Delaware, DE, USA
hfang@ece.udel.edu

Cong-Lei Yao
HP Labs, China
conglei.yao@hp.com

Min Wang
HP Labs, China
min.wang6@hp.com

ABSTRACT

Search over enterprise data is essential to every aspect of an enterprise because it helps users fulfill their information needs. Similar to Web search, most queries in enterprise search are keyword queries. However, enterprise search is a unique research problem because, compared with the data in traditional IR applications (e.g., text data), enterprise data includes information stored in different formats. In particular, enterprise data include both unstructured and structured information, and all the data center around a particular enterprise. As a result, the relevant information from these two data sources could be complementary to each other. Intuitively, such integrated data could be exploited to improve the enterprise search quality. Despite its importance, this problem has received little attention so far. In this paper, we demonstrate the feasibility of leveraging the integrated information in enterprise data to improve search quality through a case study, i.e., *finding relevant information of certain types from enterprise data*. Enterprise search users often look for different types of relevant information other than documents, e.g., the contact information of persons working on a product. When formulating a keyword query, search users may specify both content requirements, i.e., what kind of information is relevant, and type requirements, i.e., what type of information is relevant. Thus, the goal is to find information relevant to both requirements specified in the query. Specifically, we formulate the problem as keyword search over *structured* or *semistructured* data, and then propose to leverage the complementary *unstructured* information in the enterprise data to solve the problem. Experiment results over real world enterprise data and simulated data show that the proposed methods can effectively exploit the unstructured information to find relevant information of certain types from structured and semistructured information in enterprise data.

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.
Copyright 2011 ACM 978-1-4503-0717-8/11/10 ...$10.00.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Retrieval models

General Terms

Algorithms

Keywords

Enterprise search, structured information, unstructured information, retrieval, type requirements

1. INTRODUCTION

Enterprise search allows users to access information from enterprise data to fulfill their information needs such as information seeking and decision making. For example, a customer may need to search for specifications of certain products, and a product manager may need to find persons who have expertise in certain areas. Clearly, enterprise search quality is essential to an enterprise's productivity and customer satisfaction.

Similar to Web search and traditional ad hoc search, queries of enterprise search are often keyword queries. However, enterprise search is different in many aspects [22]. One of the most unique characteristics is that enterprise data include not only *unstructured* information such as documents but also *structured* or *semistructured* information such as relational databases and RDF data. These two types of information are complementary to each other since all the information centers around the enterprise. In particular, *structured* or *semistructured* information has schema which ties a piece of information with its meaning, while *unstructured* information often contains more detailed information described in free text. The different characteristics of these two types of information make it possible to leverage one type of information to help search over the other. Intuitively, the schema of the structured databases or RDF data could be useful for providing domain knowledge in keyword search over unstructured information while the rich content information in free text could be useful for query understanding and term weighting in keyword search over structured information. There have been some previous studies on enterprise search such as document search and expert finding [13, 35, 5, 11], but few of them studied how to leverage *structured* or *semistructured* information to improve search quality over *unstructured* information or vice versa.

In this paper, we study a research problem related to enterprise search, i.e., *finding relevant information of certain types from enterprise data*. The problem is chosen because it can demonstrate the feasibility of leveraging *unstructured* information to improve search over *structured* or *semistructured* data. Instead of finding relevant documents, enterprise search users often want to search for other types of relevant information. When searching for relevant information, a user may specify not only a content requirement, i.e., what kind of information is relevant, but also a type requirement, i.e., what type of information is a user looking for. For example, a user may submit query "John Smith contact information" to find the contact information of John Smith. In this query, "John Smith" is the content requirement while "contact information" is the type requirement. Clearly, the retrieved results should be the contact information of John Smith such as his emails and phone numbers rather than a ranked list of relevant documents. Despite its importance, the problem of finding relevant information of certain types has not received much attention in IR until recently. Related efforts mainly focused on finding a particular type of information such as experts [8] and related entities [10]. It remains unclear how to solve the problem in a more general way.

We formulate the problem as finding the information that is relevant to both content and type requirements specified in keyword queries from semistructured or structured data. The first challenge is to separate the type requirement from content requirement in a keyword query. We propose to utilize term semantic similarities computed from unstructured data and the difference of term distributions in unstructured data and structured data to identify type requirements specified in keyword queries. The second challenge is to rank results based on their relevance to both requirements. We discuss how to compute the relevance scores for each requirement and combine the relevance scores for the two requirements then. Our main contribution is that we proposed methods that can utilize the unique characteristic of enterprise data, i.e., the complementary information in the structured databases and unstructured documents, and use the integrated information to improve the accuracy of enterprise search.

Figure 1 illustrates the main ideas. Give a keyword query, we first identify both the content and type requirements by utilizing the unstructured information. After that, we conduct search over the (semi-)structured information based on both requirements. Note that the semistructured or structured information could be either manually created or automatically extracted from unstructured data.

We evaluate the proposed methods over a real-world enterprise data collection and a simulated collection constructed from standard collections [1, 2]. Both of them contains unstructured documents and structured information such as relational databases or RDF data. Experiment results show that the proposed methods can effectively utilize the integrated enterprise data to find relevant information satisfying both content and type requirements. Our work is the first step towards the goal of leveraging the integrated information to improve enterprise search quality.

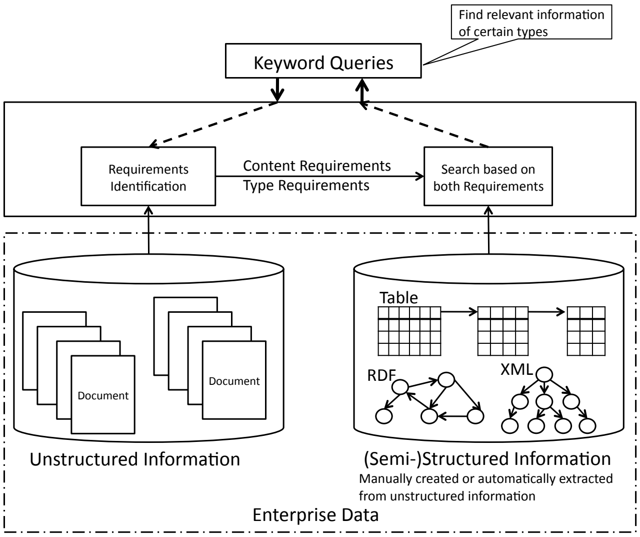

Find relevant information of certain types
Keyword Queries
Requirements Identification
Content Requirements
Type Requirements
Search based on both Requirements
Document
Document
Table
RDF
XML
Unstructured Information
(Semi-)Structured Information
Manually created or automatically extracted from unstructured information
Enterprise Data

Figure 1: Overview of finding relevant information of certain types from enterprise data.

The rest of the paper is organized as follows. We discuss the related work in Section 2. Section 3 describes the problem formulation. We discuss how to identify content requirement and type requirement in Section 4 and how to rank results based on both requirements in Section 5. Experiment design and results are discussed in Section 6. Finally, we conclude in Section 7.

2. RELATED WORK

Enterprise search is important, but it has not received much attention in the research community until recently. One study suggests that poor search quality within enterprises often causes significant loss of opportunities and productivities [18]. A more recent study takes one step further and discusses several challenges in enterprise search [22], unfortunately, no solution has been presented there. Motivated by the importance of the enterprise search, the enterprise track in TREC has been launched since 2005 [13, 35, 5, 11]. Most research efforts focus on some particular enterprise search problems such as document search [26, 30, 6, 33, 38], expert finding [8, 7, 34, 12, 9], metadata extraction [14] and user behavior in enterprise search [19, 25]. To our best knowledge, few studies looked into how to better leverage the integrated data to improve the enterprise search quality.

One limitation of traditional IR systems is that retrieved information items are limited to documents. However, users may be interested in other types of relevant information such as persons or product prices. Many recent efforts attempted to address this limitation. For example, commercial Web search engines started to direct queries to different vertical search engines based on user intentions and then aggregate the vertical search results [4]. Dalvi et. al. [15] proposed to develop a web of concepts and discussed the importance of concept search, which is in the similar line as the problem we studied in this paper. This work mainly discussed the challenges in Web domain while our work focuses on solving the problem in enterprise search. Li [29] has studied how to better understand the semantic structure of noun phrase queries by formulating them as intent heads and intend modifiers which correspond to type and content requirements in our paper. However, the work does not move further to utilize the semantic structure of query to build a more effective retrieve model. Other research efforts in IR include related entity finding [10] and expert finding [8, 7, 34, 12, 9]. These studies mainly focused on searching over *unstructured* data for a particular type of information such as experts [8] or related entities [10]. On the contrary, we focus on searching over *structured* or *semistructured* data for a broad range of information type specified in keyword queries.

Our work is closely related to semantic data search and keyword search over semistructured or structured data. Semantic data search mainly deals with the retrieval of semantic data [21], but most existing studies focused on the Web domain. DBXplorer [3] and DISCOVER [24] join tuples with primary-foreign key relation from multiple tables as *tuple trees* and rank them solely by the number of joins in the trees. EASE [28] follows the same idea to favor the compact joining graphs by utilizing the proximity between the keyword nodes to model the compactness of the joining graph. Hristidis et al. [23] adopt pivoted normalization methods to rank tuple trees. Koutrika et al. [27] use the IR-standard ranking method to compute the $tf \cdot idf$ weight of any query term in search entity. However, most of them assume that keyword queries only contain content requirement. None of them studied how to return results based on both content and type requirements and how to leverage unstructured information as what we do in this paper.

Compared with existing work, our main contribution is to leverage the unique characteristic of enterprise data, i.e., the integration of unstructured and structured information, to improve the search quality of enterprise search. In particular, we propose methods that can exploit unstructured data to identify the type and content requirements in keyword queries and to bridge the vocabulary gap between query and the structured data. Moreover, to our best knowledge, our work is the first one that studies the problem of finding relevant information of the certain type from structured and semistructured data for enterprise search.

3. PROBLEM FORMULATION

Enterprise search users often look for relevant information other than documents. When formulating a keyword query, search users may specify not only what kind of information is relevant, i.e., *content requirements*, but also what type of information is desirable, i.e., *type requirements*. Without loss of generality, we assume that a query term describes either content or type requirement and we can represent a keyword query as $Q = Q_C \bigcup Q_T$, where Q_C is the set of query terms describing the content requirement and Q_T is the set of query terms describing the type requirement. Given a keyword query and an enterprise data collection, our goal is to retrieve information satisfying both requirements from the collection.

An enterprise data collection contains unstructured, semistructured as well as structured information. Semistructured and structured data always associate the type or semantic meaning information with a piece of information. For example, the schema of a relational database specify the type of each cell in the database, and the RDF predicates or XML tags specify the semantic meaning of RDF triples or XML elements. On the contrary, unstructured data do not directly provide any information about the type or semantic meanings. A commonly used strategy for dealing with search for information of certain types over unstructured information is to apply natural language processing techniques such as NER to extract information of certain types from the unstructured information [10], and then store the extracted semantic information in structured or semi-structured format. Therefore, the problem can be simplified as finding relevant information of certain types from semistructured or structured collections for keyword queries.

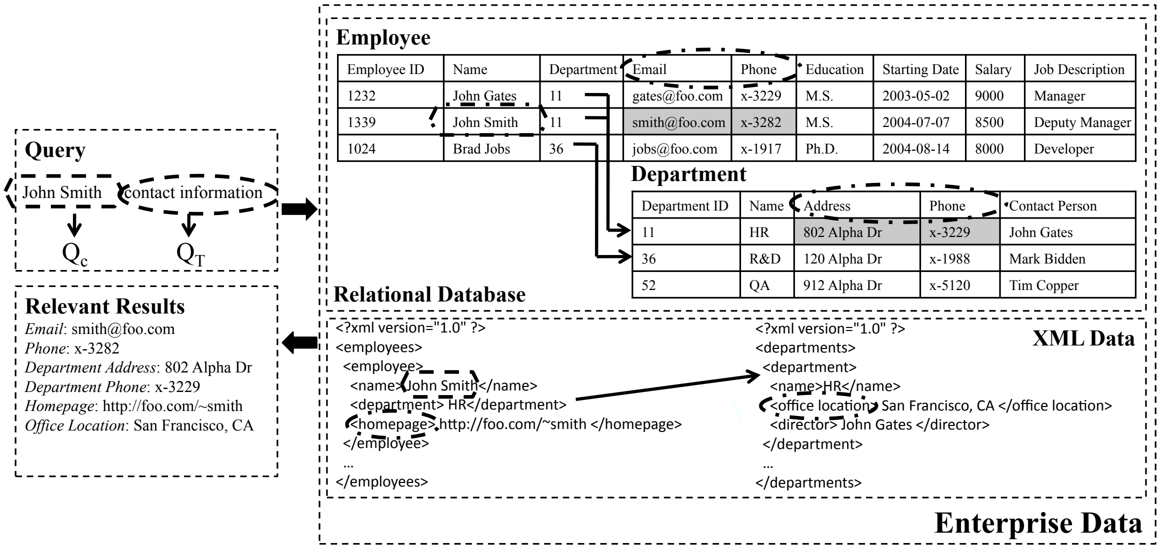
In a structured collection such as a relational database, there are multiple tables and each of them contains a set of attributes. For example, every tuple in the table corresponds to an entity, and every column corresponds to an attribute of the entity. Given the schema of a table, we know the type for each attribute based on the attribute name. Every element in the table contains the value of an attribute (i.e., type) for the corresponding entity. As an example, Figure 2 shows an relational database instance with two tables. The `Employee` table stores the information about the employees, and it has nine attributes: `Employee ID, Name, Department, Email, Phone, Education, Starting Date, Salary` and `Job Description`). The `Department` table stores the information about the departments, and it has five attributes: `Department ID, Name, Address, Phone` and `Contact Person`. Although semi-structured data do not have a strict schema as structured data, predicates or tag names often provide the meaning for corresponding information items in the collection. For example, besides the relational database, Figure 2 also shows two XML documents, one of which stores the employee data such as `name, homepage` and the other stores the department data such as `office location, director`.

An alternative way of looking at the structured or semistructured data is to represent each information item as a two dimensional vector, where one dimension represents the type and the other represents the value. Thus, the problem of finding relevant information of certain type can be reformulated as ranking all the information items in the collections based on their relevance to the query $Q = Q_C \bigcup Q_T$. Let us consider the example shown in Figure 2 again. Given query "John Smith contact information", the system is expected to return his email address and phone number stored in the `Employee` table, the address and phone number stored in the `Department` table, plus the homepage and office location stored in the two XML documents. Note that the relevant information is scattered in multiple tables and XML documents.

It is clear that there are two challenges we need to address. The first one is to separate content and type requirements in a keyword query, and the second one is to rank information based on their relevance to both requirements. We will discuss how to utilize the integrated information of enterprise data to address these two challenges in the next two sections.

4. REQUIREMENTS IDENTIFICATION

Search users may specify both type requirement, i.e., what type of information is desirable, and content requirement, i.e., what kind of information is relevant. Note that, in this paper, we assume that we know whether a query has type requirements and consider only those with both requirements. Deciding whether a query contains type requirements is out of the scope of this paper and could be solve using classification methods, which we plan to study in our future work. We now discuss how to identify these two requirements in a keyword query.



Relevant Results

Email: smith@foo.com
Phone: x-3282
Department Address: 802 Alpha Dr
Department Phone: x-3229
Homepage: http://foo.com/~smith
Office Location: San Francisco, CA

Employee

Employee ID	Name	Department	Email	Phone	Education	Starting Date	Salary	Job Description
1232	John Gates	11	gates@foo.com	x-3229	M.S.	2003-05-02	9000	Manager
1339	John Smith	11	smith@foo.com	x-3282	M.S.	2004-07-07	8500	Deputy Manager
1024	Brad Jobs	36	jobs@foo.com	x-1917	Ph.D.	2004-08-14	8000	Developer

Department

Department ID	Name	Address	Phone	Contact Person
11	HR	802 Alpha Dr	x-3229	John Gates
36	R&D	120 Alpha Dr	x-1988	Mark Bidden
52	QA	912 Alpha Dr	x-5120	Tim Copper

```
<?xml version="1.0" ?>
<employees>
  <employee>
    <name>John Smith</name>
    <department> HR</department>
    <homepage> http://foo.com/~smith </homepage>
  </employee>
  ...
</employees>
```

```
<?xml version="1.0" ?>
<departments>
  <department>
    <name>HR</name>
    <office location> San Francisco, CA </office location>
    <director> John Gates </director>
  </department>
  ...
</departments>
```

Figure 2: An example of finding relevant information of certain types from enterprise data.

4.1 Similarity based method

Intuitively, terms describing content requirement are semantically related, and those describing type requirement are semantically related. For example, consider query "John Smith contact information", terms "John Smith" are semantically related while "contact information" are semantically related. It is clear that the terms describing these two requirements naturally form two clusters. Thus, to identify requirements, we need to cluster query terms into two clusters and then assign one cluster as type requirement and the other as content requirement.

One deciding factor in clustering algorithm is the term similarity function. Fortunately, in addition to the structured databases, we also have a complementary collection of unstructured documents in enterprise data. Thus, we propose to utilize the *unstructured* information to compute similarities among query terms based on the assumption that the co-occurrences of terms indicate their semantic relations. The term similarity can be computed based on mutual information as follows [37]:

$$sim(t,u) = \sum_{\substack{Occ_t \in \{0,1\} \\ Occ_u \in \{0,1\}}} p(Occ_t, Occ_u) \log \frac{p(Occ_t, Occ_u)}{P(Occ_t)P(Occ_u)} \quad (1)$$

where Occ_t and Occ_u are two binary random variables which represents the presence or absence of two terms t and u.

Note that, unlike previous studies [16], the data collection used for computing term similarities and the collection used for searching are quite different. If these two collections are quite different, this method would not work because related terms on one domain may not be related on another. For example, "jaguar" and "drive" may be related on a set of documents about cars while they would not be related in a document collection about animals. Fortunately, since the two collections we used in this paper, i.e., structured information and unstructured information of one enterprise, are complementary and both about the enterprise, related terms discovered from one data set are likely to be related on another data set as well.

Given the term similarity function, we then cluster query term using *hierarchical agglomerative clustering* [31], which starts by treating each term in the query as one cluster, merges two clusters with highest mutual information at each step and stops when there are only two clusters left. Note that each query is treated as bag of words and terms are assumed to be independent with each other. We plan to consider the order of terms in our future work.

The remaining problem is to identify which term cluster is the content requirement and which is the type requirement. Since the database schema contains the table names as well as attributes names, they provide descriptions of different types. Thus, given a database, we can create a profile about types according to its schema by merging all the table names and attribute names together as one document. We then compute the similarity between our profile about type and each of the clusters. The similarity is computed by taking the average of term semantic similarity as shown in Equation (1). The cluster with higher similarity would be identified as the type requirement while the other corresponds to the content requirement. Similar approaches can also be applied to RDF or XML data collection.

4.2 Language Modeling based Method

We assume that each term in the query Q is independent and terms in content requirement Q_C and type requirement Q_T are generated from two different unigram language models θ_C and θ_T, respectively. The language models can be estimated using maximum likelihood estimation over the enterprise collection. In particular, we estimate $p(w_i|\theta_T)$ over all the attributes of semistructured or structured data while we compute $p(w_i|\theta_C)$ over the unstructured data.

Given a query $Q = \{q_1, q_2, \ldots, q_i, \ldots, q_n\}$, we use $R_{lm}(q_i) \in \{0,1\}$ to denote the requirements identification result for each term using language modeling approach, where $R_{lm}(q_i) = 0$ means q_i is identified as content requirement and $R_{lm}(q_i) = 1$ means q_i is identified as type requirement. Intuitively, the requirements identification for term q_i can be formulated as comparing the language modeling generation probability directly:

$$R_{lm}(q_i) = \left\{ \begin{array}{ll} 0 & \text{if } p(w_i|\theta_C) > p(w_i|\theta_T) \\ 1 & \text{if } p(w_i|\theta_C) \leq p(w_i|\theta_T) \end{array} \right. \quad (2)$$

The idea behind this method is that we assume terms from one requirement has higher probability to be generated from the associated language model than the other [32]. However, results show that this method fails to work effectively, since it ignores the similarity between terms. In order to overcome this limitation, we propose to use the language modeling generation probability as prior to adjust the results of similarity-based methods. Similarly, we use $R_{sim}(q_i) \in \{0, 1\}$ to denote the identification results of similarity based method. First, we use the results of language modeling approach to *validate* the results of similarity-based method for each term:

$$R_{sim}(q_i) = \left\{ \begin{array}{ll} R_{sim}(q_i) & \text{if } R_{sim}(q_i) = R_{lm}(q_i) \\ R_{adjust}(q_i) & \text{if } R_{sim}(q_i) \neq R_{lm}(q_i) \end{array} \right. \quad (3)$$

The reason of the validation approach is that we think for each term if the identification results from two methods are consistent, it is a strong proof that the term is correctly identified. For terms whose results from two methods differ, we will adjust the identification results to $R_{adjust}(q_i)$, which can be formulated as:

$$R_{adjust}(q_i) = \left\{ \begin{array}{ll} R_{sim}(q_i) & \text{if } \frac{sim(q_i, O_{q_i})}{sim(q_i, C_{q_i})} > prior_{lm}(q_i) \\ 1 - R_{sim}(q_i) & \text{otherwise} \end{array} \right. \quad (4)$$

where:

$$prior_{lm}(q_i) = \left\{ \begin{array}{ll} \frac{p(q_i|\theta_T)}{p(q_i|\theta_C)} & \text{if } Rsim(qi) = 0 \\ \frac{p(q_i|\theta_C)}{p(q_i|\theta_T)} & \text{if } Rsim(qi) = 1 \end{array} \right. \quad (5)$$

and $O_{q_i} = \{q_j | q_j \in Q, i \neq j, R_{sim}(q_i) = R_{sim}(q_j)\}$ is a set of terms which share the same semantic based identification results with q_i and $C_{q_i} = \{q_j | q_j \in Q, R_{sim}(q_i) \neq R_{sim}(q_j)\}$ is a set of terms whose identification results differ from that of q_i. Here $\frac{sim(q_i, O_{q_i})}{sim(q_i, C_{q_i})}$ is the probability that q_i is correctly identified by the similarity based method and $prior_{lm}(q_i)$ is the prior probability that q_i is incorrectly identified based on the knowledge of language modeling. Based on the comparison of two probabilities, we decide whether we should keep the similarity based identification result or adjust the result to another requirement.

5. RANKING METHODS

After identifying two requirements in a given keyword query, we may formulate a query as $Q = Q_C \bigcup Q_T$, where Q_C is the content requirement and Q_T is the type requirement. The problem is to rank information items such as all the table elements in the database, all the XML elements or all RDF nodes based on their relevance with respect to the query. Recall that every information item in the structured or semistructured information can be represented as a two dimensional tuple, where one dimension corresponds to its type and the other corresponds to its value. Thus, the problem boils down to model the relevance for each two dimensional information item $\mathcal{I}$ with respect to $Q = Q_C \bigcup Q_T$, which is referred to as *2-dimensional search*. The general relevance estimation approach can be formulated as:

$$Score(Q, \mathcal{I}) = \alpha \cdot Score(Q_T, \mathcal{I}) + (1 - \alpha) \cdot Score(Q_C, \mathcal{I}) \quad (6)$$

$\alpha \in [0, 1]$ controls the influence of these two scores. The estimation of $Score(Q_T, \mathcal{I})$ and $Score(Q_C, \mathcal{I})$ varies on structured and semi-structured data due to the different data characteristics.

5.1 Structured Data

In the relational databases, Q_C is used to rank tuples, i.e., rows in the table, while Q_T is used to rank attributes, i.e., columns in the table. Thus, we need to assign a relevance score to each of the table elements and rank them according to their relevance scores.

Let us start with a simple case when there is only one table in the database. For table T, we denote its m attributes as $\mathcal{AS}_T = \{\mathcal{A}_T^1, \mathcal{A}_T^2, ..., \mathcal{A}_T^m\}$. The table has n tuples which are denoted as $T = \{T_1, T_2, \ldots, T_n\}$, each tuple represents one entity with M attribute values and therefore it can be denoted as $T_i = \{T_i^1, T_i^2, \ldots, T_i^m\}$ where T_i^j denotes the jth attribute value of the ith tuple. In order to estimate the relevance score of a table element T_i^j, we can utilize its context information including T_i and $\mathcal{A}_T^j$. Specifically, T_i is the tuple which contains the table element and the values of other attributes in the tuple provide context information that can be exploited to infer the relevance of the tuple with respect to the content requirement, i.e., Q_C. $\mathcal{A}_T^j$ is the attribute name of the table element, and it can provide information related to the type requirement, i.e., Q_T. Therefore, we propose to compute the relevance score of table element T_i^j as follows:

$$\begin{aligned} Score(Q, T_i^j) &= \alpha \cdot Score(Q_T, T_i^j) + (1 - \alpha) \cdot Score(Q_C, T_i^j) \\ &= \alpha \cdot Score(Q_T, \mathcal{A}_T^j) + (1 - \alpha) \cdot Score(Q_C, T_i) \quad (7) \end{aligned}$$

where $Score(Q_T, \mathcal{A}_T^j)$ is the relevance score between the type requirement Q_T and attribute name $\mathcal{A}_T^j$, $Score(Q_C, T_i)$ is the relevance score between the content requirement Q_C and the tuple T_i. When $\alpha = 0$, only the relevance score for content requirement contributes to the final relevance score, which means that every attribute of a tuple would receive the same relevance score. When $\alpha = 1$, only the relevance score for type requirement matters, which means that all the information with the certain type would be relevant.

Let's consider the example in Figure 2. We assume there is only one `Employee` table in the database. $Score(Q_T, \mathcal{A}_T^j)$ will calculate the relevance score between the type requirement "contact information" and all the nine attribute names, and it's expected to give higher rank score to attribute `Email` and `Phone` than others. $Score(Q_C, T_i)$ will calculate the relevance score between the content requirement "John Smith" and all the three tuples, and it's expected to give higher rank score to tuple with `Employee ID` 1339 than others. By combining the results of two ranking scores using Equation (7), we can get the email and phone (i.e. smith@foo.com, x-3282) of John Smith at the top of the ranked list of attribute values.

However, tables in a real database are more complicated. In particular, the information related to an entity might be scattered in multiple tables due to normalization [20]. In the example shown in Figure 2, the address of an employee is stored in the `Department` table rather than the `Employee` table where "John Smith" occurs. Fortunately, these two table can be joined based on the foreign/primary key relationship. For example, `Department` is a foreign key from the `Employee` table to the `Department` table. After joining these two tables, we would be able to know that the attributes in the `Department` table can provide additional information to the tuples in the `Employee` table.

It is clear that, given a tuple in a table, we can find additional information stored in other tables that can be connected to the table through join operations. Let $\mathcal{N}(T)$ denote a set of all the tables that can be either directly or indirectly connected to the table T through join operations. Thus, in order to exploit the additional information, we propose to generalize the ranking function as follows:

$$\begin{aligned} & Score(Q, T_i^j) = \alpha \cdot Score(Q_T, \mathcal{A}_T^j) \\ & + (1-\alpha) \cdot \sum_{T' \in \{T_i\} \cup \mathcal{N}(T)} Score(Q_C, T'_k) \cdot Proximity(T', T_i) \end{aligned} \quad (8)$$

where T'_k corresponds to a tuple in table T' that is connected to the tuple T_i in table T, and $T'_k = T_i$ when $T = T'$.

$Proximity(T', T_i)$ denotes the proximity between the two tables. It intends to assign prior weights to the attributes in connected tables. The intuition is that we need to give more weights to the information stored in a table that is closer to the target table. The closeness between tables is defined by the number of foreign key links used to connect these two tables. There could be different ways of defining such a proximity function, and in this paper, we use the following simple strategy:

$$Proximity(T_1, T_2) = \frac{1}{1 + nl(T_1, T_2)}$$

where $nl(T_1, T_2)$ is the number of foreign key links between table T_1 and T_2. If T_1 is the same as T_2, $nl(T_1, T_2) = 0$. The estimation follows the intuition that the farther a table is from the target table T_i, the less confidence we have to conclude that the additional information in the table is relevant to the one in the target table. Therefore, for the example shown in Figure 2, we will give higher weight to the attributes in table `Employee` (e.g. `Email`, `Phone`) and lower weight to the attribute in table `Department` (e.g. `Address`).

We now describe how we estimate the two score components: $Score(Q_C, T_i)$ and $Score(Q_T, \mathcal{A}_T^j)$. Since Q_C and Q_T are essentially keyword queries, and T_i and $\mathcal{A}_T^j$ can be treated as unstructured document, we may use any retrieval functions to compute them. In this paper, we use F2-EXP, an axiomatic retrieval function [16] because previous study shows that the function is robust and insensitive to parameter settings and a fixed parameter setting can reach comparable performance with other state of the art retrieval functions.

However, the users may not be aware of the schema of the enterprise database, thus if they come up with the type requirement of the query, it may not have any overlapped terms with the target attribute names in the database. In the example shown in Figure 2, the relevant attributes `Email`, `Phone` and `Address` do not have any overlapped term with the type requirement "contact information". The attribute `Contact Person` has one overlapped term, but it's not a relevant attribute with regard to the query. We call this the *vocabulary gap* between database and user queries. Therefore, using ordinary keyword search retrieval models will fail definitely because these models depends on the overlapped terms between the query and document. To bridge the vocabulary gap, we follow the existing work on semantic term matching [17] and utilize the unstructured documents collection to get the K most semantic similar terms of the original type requirement based on the mutual information estimated in Equation (1). This works under the assumption that the type requirement and its target attribute have high semantic similarity. We then expand the original type requirement Q_T with the K weighted terms into $Q_{T_{EXP}}$, which is used to replace Q_T to estimate the relevance score.

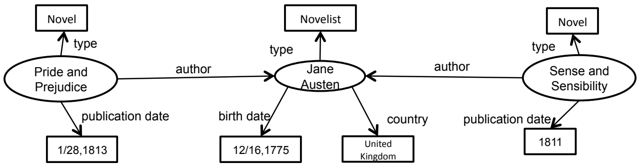


Figure 3: An example RDF graph.

5.2 Semi-Structured Data

Due to space limit, here we only take RDF as example of semi-structured data. Our method is also applicable to other semi-structured data like XML. In RDF graph, each entity is represented as one node, and one entity may have several attributes. Each attribute is a pair of attribute name and attribute value. In the terminology of RDF, each entity and one of its attribute is expressed in one statement called *triple*, in which the entity is subject, attribute name is predicate and attribute value is object. Consider the example shown in Figure 3, there are three entities in total, and entity "Pride and Prejudice" has three attributes (i.e. `type`, `author` and `publication date`). Intuitively, the *type* attribute describes the type information of the entity and the other attributes as well as entity name describes all the other information related to the entity. Therefore, for each entity E_i, we take the `type` attribute as the *type aspect* E_i^T and all the other attributes as *content aspect* E_i^C and eventually each entity can be expressed as 2-dimensional vector $E_i = \{E_i^C, E_i^T\}$. For example, consider the entity $E_i =$ "Pride and Prejudice" in Figure 3, $E_i^C =$ "Pride and Prejudice, 1/28, 1983, Jane Austen". Currently we only consider the type attributes with directly connections to the entities. However, they may also inherit type attributes from others through hierarchical relations. We will take them into consideration in our future work.

To retrieve entities from RDF graph, we need to estimate the relevance between E_i and both the content requirement Q_C and type requirement Q_T. Similar to Equation (7), the relevance score can be formulated as:

$$\begin{aligned} Score(Q, E_i) &= \alpha \cdot Score(Q_T, E_i) + (1-\alpha) \cdot Score(Q_C, E_i) \\ &= \alpha \cdot Score(Q_T, E_i^T) + (1-\alpha) \cdot Score(Q_C, E_i^C) \end{aligned} \quad (9)$$

Note that for one entity not all of the attributes values are plain text. Some of the attribute values may also be other entities in the graph, therefore entities are connected with each other through such attributes as links. In the example shown in Figure 3 both the entity "Pride and Prejudice" and "Sense and Sensibility" share the same attribute "author" with value "Jane Austen" which is also an entity. These three entities are connected through the directed arcs. Therefore, the related information for one entity is scattered into several nodes, similar to the normalization [20] in relational database. Intuitively, to get a more accurate estimation of the content requirement score of each entity, we should not only consider the node itself, but also its neighbors. By leveraging the graph information, after we have done the initial estimation of $Score(Q_C, E_i^C)$, we update it in the following way:

$$Score(Q_C, E_i^C) = (1-\beta)Score(Q_C, E_i^C) + \beta\frac{1}{|\mathcal{N}(E_i)|}\sum_{j\in\mathcal{N}(E_i)} Score(Q_C, E_j^C) \quad (10)$$

where $\mathcal{N}(E_i)$ is the set of all the neighbor nodes of entity E_i, $|\mathcal{N}(E_i)|$ is the size of $\mathcal{N}(E_i)$, $\beta \in [0,1]$ controls the influence of the scores from neighbor nodes.

We now describe how we estimate the two score components: $Score(Q_C, E_i^C)$ and $Score(Q_T, E_i^T)$. Similar to the score component estimation over structured data as described in end of Section 5.1, we may also use any retrieval functions to do so by treating Q_C and Q_T as keyword queries and E_i^C and E_i^T as unstructured documents, respectively. Again, considering the vocabulary gap between the type requirement and the attribute names in the RDF data, we expand the original type requirement Q_T with the K most semantic similar terms into $Q_{T_{EXP}}$ based on the mutual information by utilizing the unstructured documents collection.

6. EXPERIMENTS

We describe the experiment design and results in this section. We choose two data sets for evaluation: one is a real world enterprise data set, and the other one is a simulated data set constructed using standard collections. Over each data set, we conduct two sets of experiments to evaluate the effectiveness of requirement identification methods and ranking methods.

6.1 Real-world Enterprise Data Set

We now describe the enterprise data set used in this paper. The data set contains both *unstructured* and *structured* information about HP, which is referred to as **REAL**.

We first study a query log with queries submitted to the HP web search engine. The log includes 1,060,792 queries in total. The average number of terms in each query is 2.2, which is similar to the observation in web search [36]. After analyzing the query log, we find that the majority of the queries are about product search. Thus, the enterprise data we use for this paper is built around products of HP.

The first part of the data collection includes all the web pages of HP. There are 477,800 web pages, which leads to a total size of 18 GB *unstructured* information. On average each document has 877 terms. The second part of the collection includes a relational database with 25 tables that contain the information about products of HP with a total size of 39,524 KB *structured* information. A table could be either independent or related to other tables. When a table is related to another table, they could be joined together based on their foreign/primary keys.

Specifically, there are five types of tables in the database: *product*, *product series*, *product marketing subcategory*, *product marketing category* and *product type*. A product belongs to one product series, a product series belongs to one product marketing subcategory, a product subcategory belongs to one product category, and a product category belongs to one product type. Clearly, these five types form a concept hierarchy of the products in the databases. The database contains 3,238 products, 983 product series, 134 product marketing subcategories, 54 product marketing categories and 8 product types. Each product table contains information about one type of products. Each product type has its own schema because different types of products may have different specifications. For example, the specification of a printer may include resolution while that of a camera may include storage. Within a product table, each tuple in the table contains the information of a product with the corresponding type. For instance, every tuple in the printer table corresponds to a printer.

Methods	Equation	Prec	Recall	F1
Similarity	(1)	0.752	0.746	0.746
Language Modeling	(4)	0.890	0.887	0.887

Table 1: Performance of type requirement identification over REAL.

Note that although the enterprise data we used center around products, our proposed algorithms are general enough and can be applicable to any enterprise data.

Based on the analysis over the query log, we find that 23.21% queries have mentioned any attribute in the product database, thus these queries have great potential to benefit from our proposed 2-dimensional search methods. Within these queries, we manually select a set of 50 queries which mention both valid product name and attribute in the database. Clearly, all the 50 queries contain both the content and type requirement. The search results of the 2-dimensional search need to follow the format below:

```
<tuple ID, attribute name, attribute value>
```

Each result is specified by the triple of tuple ID (the primary key of the tuple), attribute name and attribute value. Moreover, we create a judgment file which assign relevance labels to table elements for every query by manually retrieving relevant elements from the table. A relevant result should be an exactly same triple in the judgment file. We use MAP (Mean Average Precision) over the 50 queries as the main measurement for the performance. Since each query has 9 relevant results on average, we also use $P@10$ (Precision at rank 10). Besides that, the performance measured with R-precision (R is the number of relevant results for a given topic) is also reported.

We first compare performance of two requirement identification methods proposed in Section 4. To evaluate their performance, we first create judgment files for all the query set by manually identify both content and type requirements for each query. Since the query requirement identification essentially is a classification problem in which each term has to be classified into two classes: either content requirement or type requirement, we evaluate the performance by calculate the macro-average of precision, recall and F1 for each query, and take the average over all the queries then. Table 1 shows the performance of two query requirements identification methods over **REAL**. We can see that both methods can reach reasonable performance, and the language modeling method can reach better performance than similarity based method. In the following experiments, we only report the ranking results using the identification results of

Models	Equation	MAP	P@10	R-Precision
1dBL	(11)	0.0637	0.1466	0.0111
2dBL	(12)	0.5416	0.5640	0.5111
2dRank	(8)	**0.6655***	**0.7139***	**0.6546***

Table 2: Optimal performance comparison on REAL. $\star$ means improvement over 2dBL is statistically significant.

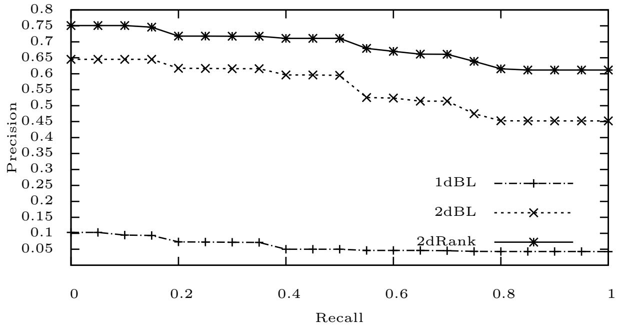


Figure 4: Precision-recall curves over REAL.

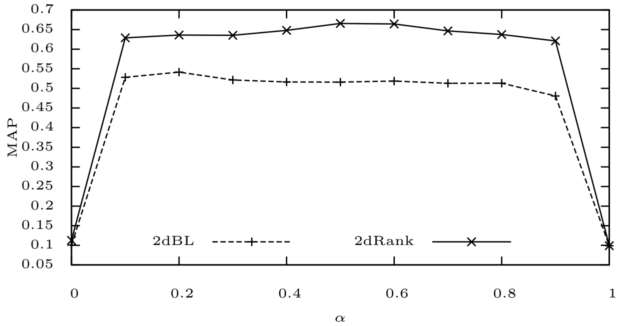


Figure 5: Performance sensitivity of 2dBL and 2dRank over REAL.

Identification Methods	MAP	P@10	R-Precision
Similarity	0.5349	0.5542	0.5170
Language Modeling	0.6655	0.7139	0.6546

Table 3: Optimal performance of ranking based on different type requirement identification results on REAL.

language modeling method because it is expected to lead to better performance.

Baseline Methods: Our work is related to keyword search over structured databases. The main difference is that existing methods only consider ranking joined tuples rather than table elements [23], i.e., attributes. The first baseline method we use is one of the state of the art methods for keyword search over structured databases [23], which is referred to as **1dBL**. In order to apply existing methods to solve our problem, we can use them to rank tuples and then return all the attributes in a random order as follows:

$$Score_{1dBL}(Q, T_i^j) = \frac{1}{|\mathcal{N}(T)|} \sum_{T' \in \{T\} \in \mathcal{N}(T)} Score(Q_C, T'_k), \quad (11)$$

where $\mathcal{N}(T)$ denotes a set of all the tables that can be either directly or indirectly connected to the table T through join operations and $|\mathcal{N}(T)|$ is the size of $\mathcal{N}(T)$.

Moreover, we also implement a stronger baseline where we combine our proposed method by incorporating the relevance score based on the type requirement into the first baseline. This method is referred to as **2dBL**.

$$\begin{aligned} Score_{2dBL}(Q, T_i^j) &= \alpha \cdot Score(Q_{T_{EXP}}, \mathcal{A}_T^j) \\ &+ (1-\alpha) \cdot \frac{1}{|\mathcal{N}(T)|} \sum_{T' \in \{T\} \in \mathcal{N}(T)} Score(Q_C, T'_k) \quad (12) \end{aligned}$$

In fact, **1dBL** is a special case of **2dBL** when α is set to 0.

Implementation Details: Our proposed method is referred to as **2dRank**. There are two components in the proposed ranking function and two baseline methods. (Note that Q_T is expanded into $Q_{T_{EXP}}$ as described at the end of Section 5.1.)

Table 2 shows the optimal performance comparison of the three methods measured with three standard evaluation metrics. In addition, we also plot the precision-recall curve for all the three models in Figure 4. It is clear that the **2dRank** performs much better than the two baseline methods. First, we can see that the *2-dimensional search* can clearly yield to better search performance because it can return direct answers. Second, the proximity function in Equation 8 is more effective for 2-dimensional search. Moreover, **1dBL** shows that even without incorporating type constraints into the ranking, our model can still find some relevant information, although with the penalty of great performance loss.

We have manually checked the results of some queries, and found that for some queries whose type requirements have vocabulary gap with the database schema (for example, one such type requirement is "dimension"), some relevant attributes ("width", "depth", "height") are successfully ranked at the top. This proves the effectiveness of our method on bridging the vocabulary gap.

We also examine the performance sensitivity with respect to the parameter α for **2dBL** and **2dRank**, as shown in Figure 5. We find that **2dRank** always performs better than **2dBL** at all α levels. When $\alpha = 1$, both models are essentially the same as their ranking functions regress to $Score(Q_T, \mathcal{A}_T^j)$. For α set to either 0 or 1, both models regress to one-dimensional search, therefore the performance is definitely poor. We also notice that **2dBL** reaches the optimal performance when $\alpha = 0.2$ and **2dRank** reaches the optimal performance when $\alpha = 0.5$.

To verify the correlation between the ranking performance and quality of query requirements identification, we report the optimal performance of the ranking results using different identification methods in Table 3. Connecting with the results in Table 1 we find that that better requirements identification results can lead to better ranking results, as we expected.

6.2 Simulated Data Set

Enterprise data collection is hard to find since most enterprises do not want to publish their internal data for security concerns. In order to evaluate our proposed methods on

Methods	Equation	Prec	Recall	F1
Similarity	(1)	0.771	0.748	0.757
Language Modeling	(4)	0.605	0.618	0.601

Table 4: Performance of type requirement identification over SIMU.

Method	MAP	P@10	R-Precision
1dBL	0.1542	0.1780	0.1553
2dBL	0.1585	0.1920	0.1767
2dSem	**0.1675***	0.1900	0.1636
2dGraph	0.1659*	0.2120	**0.1835**
2dGraphSem	0.1653	**0.2180**	0.1831

Table 5: Optimal performance comparison on SIMU. $\star$ means improvement over 2dBL is statistically significant.

Identification Methods	MAP	P@10	R-Precision
Similarity	0.1675	0.1900	0.1636
Language Modeling	0.1612	0.1850	0.1609

Table 6: Optimal performance of ranking based on different type requirement identification results on SIMU.

semistructured data, we constructed a simulated data set by choosing the Billion Triple Challenge 2009 dataset [1], which consists of a RDF graph with 1,464,829,200 nodes, as semi-structured data. Beside this, we choose the Category B (first 50 million pages of English part) of ClueWeb09 collection [2] as the complementary unstructured data. The reason that they can serve as complementary to each other is because both data sets contain entity information over the web. The data set is referred to as **SIMU**.

We choose the query set used in the List Search Track of SemSearch 2011 Challenge [21]. The query set consists of 50 queries which are hand-written by the organizing committee. The average number of terms in each query is 5.4. Here are some sample queries:

```
astronauts who walked on the Moon
nations where Portuguese is an official language
```

The reason to choose it is because based on the analysis we find all the queries have mentioned both the content and type requirements. The judgment file, which is created by system pooling from the participants' submissions, is also provided along with the query set. We report the performance evaluation using MAP, P@10 and R-Precision.

We first report the performance of two query requirements identification methods over **SIMU** data set in Table 4. According to the results, we find that the language modeling based method failed to outperform similarity based method. Results analysis shows that the quality of language modeling estimation of θ_C on ClueWeb09 is not as accurate as that on unstructured part of **REAL** data set, since ClueWeb09 has a much wider domain coverage. In the following experiments, we only report the ranking results using the identification results of similarity based method.

Baseline Methods: By treating each entity as an unstructured document by merging all the attributes, we use the whole original query directly to retrieve entities from the data collection directly using the F2-EXP retrieval function. This method essentially performs 1-dimensional retrieval, and is denoted as **1dBL**.

Moreover, we implemented another baseline by estimating $Score(Q_T, E_i^T)$ and $Score(Q_C, E_i^C)$ directly using F2-EXP retrieval function and combine them as shown in Equation (9). As it is essentially 2-dimensional search, we refer it as **2dBL**.

Implementation Details: Based on **2dBL**, we use our proposed graph based method in Equation (10) to estimate $Score(Q_C, E_i^C)$. This method is referred as **2dGraph**.

Moreover, based on **2dBL**, we also implement another ranking method by substituting Q_T with the semantic expanded requirement $Q_{T_{EXP}}$ in the estimation of $Score(Q_T, E_i^T)$. This method is referred as **2dSem**.

Finally, we implement one method by both using the graph based method to estimate $Score(Q_C, E_i^C)$ and substituting Q_T with $Q_{T_{EXP}}$. It is referred to as **2dGraphSem**. The ranking function is:

$$\begin{aligned} Score(Q_C, E_i^C) = {} & \alpha \cdot Score(Q_{T_{EXP}}, E_i^T) \\ & + (1-\alpha) \cdot \Big((1-\beta) Score(Q_C, E_i^C) \\ & + \beta \frac{1}{|\mathcal{N}(E_i)|} \sum_{j \in \mathcal{N}(E_i)} Score(Q_C, E_j^C) \Big) \end{aligned} \tag{13}$$

The optimal performance for the five methods are shown in Table 5. By comparing 1dBL with other methods, we find that 2-dimensional search can always outperforms 1 dimensional search. What's more, both the graph based estimation of the content requirement score and type requirement score using semantic expanded queries can lead to better performance.

Finally, we verify the correlation between the ranking performance and quality of query requirements identification, by reporting the optimal performance of the ranking results using different identification methods in Table 6. It shows again that better requirements identification results can lead to better ranking results.

7. CONCLUSIONS AND FUTURE WORK

The paper aims to demonstrate the feasibility of leveraging unstructured information to improve the search quality over structured and semi-structured information. The problem is to find relevant information of certain types from *structured* and *semi-structured* information. We propose ranking methods that can utilize *unstructured* information to identify type requirement in keyword queries and bridge the vocabulary gap between the query and the data collection. We conduct experiments over one real world enterprise data set and one simulated data set. Experiment results show that our proposed ranking strategy is effective to retrieve relevant information with the type specified in keyword queries.

Enterprise search is important. Our proposed work is only a small step toward improving enterprise search quality. There are many interesting future research directions based on this work. First, it would be interesting to study how to identify queries with type requirements. Second, we would like to extend our query requirements identification model so that it can process queries with multiple content or type requirements. Third, we plan to study how to retrieve information of certain types from unstructured information directly in our future work. Finally, it would be interesting to study how to provide seamless support for search over both unstructured and structured information.

8. ACKNOWLEDGEMENTS

This material is based upon work supported by the HP Labs Innovation Research Program. We thank the anonymous CIKM reviewers for their useful comments.

9. REFERENCES

[1] Billion Triple Challenge 2009 dataset: http://vmlion25.deri.ie/.

[2] The ClueWeb09 Dataset: http://www.lemurproject.org/clueweb09/.

[3] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *ICDE*, pages 5–16, 2002.

[4] J. Arguello, F. Diaz, J. Callan, and J.-F. Crespo. Sources of Evidence for Vertical Selection. In *SIGIR*, pages 315–322, 2009.

[5] P. Bailey, N. Craswell, A. P. de Vries, and I. Soboroff. Overview of the TREC 2007 Enterprise Track. In *Proceedings of TREC'07*, 2007.

[6] P. Bailey, D. Hawking, and B. Matson. Secure Search in Enterprise Webs: Tradeoffs in Efficient Implementation for Document Level Security. In *CIKM*, pages 493–502, 2006.

[7] K. Balog. People Search in the Enterprise. In *SIGIR*, pages 916–916, 2007.

[8] K. Balog, L. Azzopardi, and M. de Rijke. Formal Models for Expert Finding in Enterprise Corpora. In *SIGIR*, pages 43–50, 2006.

[9] K. Balog and M. de Rijke. Non-Local Evidence for Expert Finding. In *CIKM*, pages 489–498, 2008.

[10] K. Balog, A. P. de Vries, P. Serdyukov, P. Thomas, and T. Westerveld. Overview of the TREC 2009 Entity Track. In *Proceedings of TREC'09*, 2009.

[11] K. Balog, I. Soboroff, P. Thomas, P. Bailey, N. Craswell, and A. P. de Vries. Overview of the TREC 2008 Enterprise Track. In *Proceedings of TREC'08*, 2008.

[12] J. Brunnert, O. Alonso, and D. Riehle. Enterprise People and Skill Discovery Using Tolerant Retrieval and Visualization. In *ECIR*, pages 674–677, 2007.

[13] N. Craswell, A. P. de Vries, and I. Soboroff. Overview of the TREC 2005 Enterprise Track. In *Proceedings of TREC'05*, 2005.

[14] M. Şah and V. Wade. Automatic Metadata Extraction from Multilingual Enterprise Content. In *CIKM*, pages 1665–1668, 2010.

[15] N. Dalvi, R. Kumar, B. Pang, R. Ramakrishnan, A. Tomkins, P. Bohannon, S. Keerthi, and S. Merugu. A Web of Concepts. In *PODS*, pages 1–12, 2009.

[16] H. Fang and C. Zhai. An Exploration of Axiomatic Approaches to Information Retrieval. In *SIGIR*, pages 480–487, 2005.

[17] H. Fang and C. Zhai. Semantic Term Matching in Axiomatic Approaches to Information Retrieval. In *SIGIR*, pages 115–122, 2006.

[18] S. Feldman and C. Sherman. The High Cost of Not Finding Information. In *Technical Report No. 29127, IDC*, 2003.

[19] L. Freund and E. G. Toms. Enterprise Search Behaviour of Software Engineers. In *SIGIR*, pages 645–646, 2006.

[20] H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice-Hall, 2008.

[21] H. Halpin, D. Herzig, P. Mika, J. Pound, H. Thompson, and T. T. Duc. SemSearch Evaluation 2011: Semantic Search Challenge. In *WWW*, 2011.

[22] D. Hawking. Challenges in Enterprise Search. In *Proceedings of ADC'04*, pages 15–24, 2004.

[23] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style Keyword Search over Relational Database. In *VLDB*, pages 850–861, 2003.

[24] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. In *VLDB*, pages 670–681, 2002.

[25] A. Kale, T. Burris, B. Shah, T. L. P. Venkatesan, L. Velusamy, M. Gupta, and M. Degerattu. iCollaborate: Harvesting Value from Enterprise Web Usage. In *SIGIR*, pages 699–699, 2010.

[26] M. Kolla and O. Vechtomova. Retrieval of Discussions from Enterprise Mailing Lists. In *SIGIR*, pages 881–882, 2007.

[27] G. Koutrika, Z. M. Zadeh, and H. Garcia-Molina. Data Clouds: Summarizing Keyword Search Results over Structured Data. In *EDBT*, pages 391–402, 2009.

[28] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. EASE: an Effective 3-in-1 Keyword Search Method for Unstructured, Semi-structured and Structured Data. In *SIGMOD*, pages 903–914, 2008.

[29] X. Li. Understanding the Semantic Structure of Noun Phrase Queries. In *ACL*, pages 1337–1345, 2010.

[30] C. Macdonald and I. Ounis. Combining Fields in Known-Item Email Search. In *SIGIR*, pages 675–676, 2006.

[31] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval.* Cambridge University Press, 2008.

[32] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing.* The MIT Press, 1999.

[33] J. Peng, C. Macdonald, B. He, and I. Ounis. A Study of Selective Collection Enrichment for Enterprise Search. In *CIKM*, pages 1999–2002, 2009.

[34] P. Serdyukov, H. Rode, and D. Hiemstra. Modeling Multi-step Relevance Propagation for Expert Finding. In *CIKM*, pages 1133–1142, 2008.

[35] I. Soboroff, A. P. de Vries, and N. Craswell. Overview of the TREC 2006 Enterprise Track. In *Proceedings of TREC'06*, 2006.

[36] A. Spink, D. Wolfram, M. Jansen, and T. Saracevic. Searching the Web: The Public and Their Queries. *Journal of the American Society for Information Science and Technology*, pages 226–234, 2001.

[37] C. J. van Rijsbergen. *Information Retrieval.* Butterworths, 1979.

[38] S. Yang, J. Jin, and Y. Xiong. Using Weighted Tagging to Facilitate Enterprise Search. In *ECIR*, pages 590–593, 2010.