Efficient Prediction of Difficult Keyword Queries over Databases

Shiwen Cheng, Arash Termehchy, and Vagelis Hristidis

Abstract—Keyword queries on databases provide easy access to data, but often suffer from low ranking quality, i.e., low precision and/or recall, as shown in recent benchmarks. It would be useful to identify queries that are likely to have low ranking quality to improve the user satisfaction. For instance, the system may suggest to the user alternative queries for such hard queries. In this paper, we analyze the characteristics of hard queries and propose a novel framework to measure the degree of difficulty for a keyword query over a database, considering both the structure and the content of the database and the query results. We evaluate our query difficulty prediction model against two effectiveness benchmarks for popular keyword search ranking methods. Our empirical results show that our model predicts the hard queries with high accuracy. Further, we present a suite of optimizations to minimize the incurred time overhead.

Index Terms—Query performance, query effectiveness, keyword query, robustness, databases

1 Introduction

Keyword query interfaces (*KQIs*) for databases have attracted much attention in the last decade due to their flexibility and ease of use in searching and exploring the data [1]–[5]. Since any entity in a data set that contains the query keywords is a potential answer, keyword queries typically have many possible answers. KQIs must identify the information needs behind keyword queries and rank the answers so that the desired answers appear at the top of the list [1], [6]. Unless otherwise noted, we refer to *keyword query* as *query* in the remainder of this paper.

Databases contain entities, and entities contain attributes that take attribute values. Some of the difficulties of answering a query are as follows: First, unlike queries in languages like SQL, users do not normally specify the desired schema element(s) for each query term. For instance, query Q_1: *Godfather* on the IMDB database (http://www.imdb.com) does not specify if the user is interested in movies whose *title* is *Godfather* or movies distributed by the *Godfather* company. Thus, a KQI must find the desired attributes associated with each term in the query. Second, the schema of the output is not specified, i.e., users do not give enough information to single out exactly their desired entities [7]. For example, Q_1 may return movies or actors or producers. We present a more complete analysis of the sources of difficulty and ambiguity in Section 4.2.

- *S. Cheng is with the University of California, Riverside, CA 92521 USA. E-mail: schen064@cs.ucr.edu.*
- *A. Termehchy is with the Oregon State University, Corvallis, OR 97331 USA. E-mail: termehca@eecs.oregonstate.edu.*
- *V. Hristidis is with the University of California, Riverside, CA 92521 USA. E-mail: vagelis@cs.ucr.edu.*

Manuscript received 12 Sep. 2012; revised 20 Apr. 2013; accepted 27 July 2013. Date of publication 14 Aug. 2013; date of current version 29 May 2014.
Recommended for acceptance by S. Sudarshan.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier 10.1109/TKDE.2013.140

Recently, there have been collaborative efforts to provide standard benchmarks and evaluation platforms for keyword search methods over databases. One effort is the data-centric track of INEX Workshop [8] where KQIs are evaluated over the well-known IMDB data set that contains structured information about movies and people in show business. Queries were provided by participants of the workshop. Another effort is the series of Semantic Search Challenges (SemSearch) at Semantic Search Workshop [9], where the data set is the Billion Triple Challenge data set at http://vmlion25.deri.de. It is extracted from different structured data sources over the Web such as Wikipedia. The queries are taken from Yahoo! keyword query log. Users have provided relevance judgments for both benchmarks.

The Mean Average Precision (MAP) of the best performing method(s) in the last data-centric track in INEX Workshop and Semantic Search Challenge for queries are about 0.36 and 0.2, respectively. These results indicate that even with structured data, finding the desired answers to keyword queries is still a hard task. More interestingly, looking closer to the ranking quality of the best performing methods on both workshops, we notice that they all have been performing very poorly on a subset of queries. For instance, consider the query *ancient Rome era* over the IMDB data set. Users would like to see information about movies that talk about ancient Rome. For this query, the state-of-the-art XML search methods which we implemented return rankings of considerably lower quality than their average ranking quality over all queries. Hence, some queries are more difficult than others. Moreover, no matter which ranking method is used, we cannot deliver a reasonable ranking for these queries. Table 1 lists a sample of such hard queries from the two benchmarks. Such a trend has been also observed for keyword queries over text document collections [10]. These queries are usually either under-specified, such as query *carolina* in Table 1, or overspecified, such as query *Movies Klaus Kinski actor good rating* in Table 1.

TABLE 1
Some Difficult Queries from Benchmarks

INEX	SemSearch
ancient Rome era	Austin Texas
Movies Klaus Kinski actor good rating	Carolina
true story drugs addiction	Earl May

It is important for a KQI to recognize such queries and warn the user or employ alternative techniques like query reformulation or query suggestions [11]. It may also use techniques such as query results diversification [12].

To the best of our knowledge, there has not been any work on predicting or analyzing the difficulties of queries over databases. Researchers have proposed some methods to detect difficult queries over plain text document collections [10], [13]. However, these techniques are not applicable to our problem since they ignore the structure of the database. In particular, as mentioned earlier, a KQI must assign each query term to a schema element(s) in the database. It must also distinguish the desired result type(s). We empirically show that direct adaptations of these techniques are ineffective for structured data.

In this paper, we analyze the characteristics of difficult queries over databases and propose a novel method to detect such queries. We take advantage of the structure of the data to gain insight about the degree of the difficulty of a query given the database. We have implemented some of the most popular and representative algorithms for keyword search on databases and used them to evaluate our techniques on both the INEX and SemSearch benchmarks. The results show that our method predicts the degree of the difficulty of a query efficiently and effectively.

We make the following contributions:

- We introduce the problem of predicting the degree of the difficulty for queries over databases. We also analyze the reasons that make a query difficult to answer by KQIs (Section 4).
- We propose the *Structured Robustness (SR)* score, which measures the difficulty of a query based on the differences between the rankings of the same query over the original and noisy (corrupted) versions of the same database, where the noise spans on both the content and the structure of the result entities (Section 5).
- We present an algorithm to compute the SR score, and parameters to tune its performance (Section 6).
- We introduce efficient approximate algorithms to estimate the SR score, given that such a measure is only useful when it can be computed with a small time overhead compared to the query execution time (Section 7).
- We show the results of extensive experiments using two standard data sets and query workloads: INEX and SemSearch. Our results show that the SR score effectively predicts the ranking quality of representative ranking algorithms, and outperforms non-trivial baselines, introduced in this paper. Also, the time spent to compute the SR score is negligible compared to the query execution time (Section 8).

Section 2 discusses related work and Section 3 presents basic definitions. Section 9 concludes the paper and presents future directions.

2 Related Work

Researchers have proposed methods to predict hard queries over unstructured text documents [10], [13]–[17]. We can broadly categorize these methods into two groups: *pre-retrieval* and *post-retrieval* methods.

Pre-retrieval methods [14], [18] predict the difficulty of a query without computing its results. These methods usually use the statistical properties of the terms in the query to measure *specificity*, *ambiguity*, or *term-relatedness* of the query to predict its difficulty [19]. Examples of these statistical characteristics are average inverse document frequency of the query terms or the number of documents that contain at least one query term [14]. These methods generally assume that the more discriminative the query terms are, the easier the query will be. Empirical studies indicate that these methods have limited prediction accuracies [10], [20].

Post-retrieval methods utilize the results of a query to predict its difficulty and generally fall into one of the following categories.

Clarity-score-based: The methods based on the concept of *clarity score* assume that users are interested in a very few topics, so they deem a query easy if its results belong to very few topic(s) and therefore, sufficiently distinguishable from other documents in the collection [10], [14], [15], [20]. Researchers have shown that this approach predicts the difficulty of a query more accurately than pre-retrieval based methods for text documents [10]. Some systems measure the distinguishability of the queries results from the documents in the collection by comparing the probability distribution of terms in the results with the probability distribution of terms in the whole collection. If these probability distributions are relatively similar, the query results contain information about almost as many topics as the whole collection, thus, the query is considered difficult [10]. Several successors propose methods to improve the efficiency and effectiveness of clarity score [14], [15], [20].

However, one requires domain knowledge about the data sets to extend idea of clarity score for queries over databases. Each topic in a database contains the entities that are about a similar subject. It is generally hard to define a formula that partitions entities into topics as it requires finding an effective similarity function between entities. Such similarity function depends mainly on the domain knowledge and understanding users' preferences [21]. For instance, different attributes may have different impacts on the degree of the similarity between entities. Our empirical results in Section 8 confirms this argument and shows that the straightforward extension of clarity score predicts difficulties of queries over databases poorly.

Ranking-score-based: The ranking score of a document returned by the retrieval systems for an input query may estimate the similarity of the query and the document. Some recent methods measure the difficulty of a query

based on the score distribution of its results [16], [17]. Zhou and Croft argue that the information gained from a desired list of documents should be much more than the information gained from typical documents in the collection for an easy query. They measure the degree of the difficulty of a query by computing the difference between the weighted entropy of the top ranked results' scores and the weighted entropy of other documents' scores in the collection [17]. Shtok *et al*. argue that the amount of non-query-related information in the top ranked results is negatively correlated with the deviation of their retrieval scores [16]. Using language modeling techniques, they show that the standard deviation of ranking scores of top-k results estimates the quality of the top ranked results effectively. We examine the query difficulty prediction accuracy of this set of methods on databases in Section 8, and show that our model outperforms these methods over databases.

Robustness-based: Another group of post-retrieval methods argue that the results of an easy query are relatively stable against the perturbation of queries [22], documents [13] or ranking algorithms [23]. Our proposed query difficulty prediction model falls in this category. More details of some related work will be given in Section 4, where we discuss the difference of applying these techniques on text collection and database.

Some methods use machine learning techniques to learn the properties of difficult queries and predict their hardness [22]. They have similar limitations as the other approaches when applied to structured data. Moreover, their success depends on the amount and quality of their available training data. Sufficient and high quality training data is not normally available for many databases. Some researchers propose frameworks that theoretically explain existing predictors and combine them to achieve higher prediction accuracy [24], [25].

A preliminary version of this work has been published [26].

3 Data and Query Models

We model a database as a set of entity sets. Each entity set S is a collection of entities E. For instance, *movies* and *people* are two entity sets in IMDB. Fig. 1 depicts a fragment of a data set where each subtree whose root's label is *movie* represents an entity. Each entity E has a set of attribute values A_i, $1 \leq i \leq |E|$. Each attribute value is a bag of terms. Following current unstructured and (semi-) structure retrieval approaches, we ignore stop words that appear in attribute values, although this is not necessary for our methods. Every attribute value A belongs to an *attribute* T written as $A \in T$. For instance, *Godfather* and *Mafia* are two attribute values in the movie entity shown in the subtree rooted at node 1 in Fig. 1. Node 2 depicts the attribute of *Godfather*, which is *title*.

The above is an abstract data model. We ignore the physical representation of data in this paper. That is, an entity could be stored in an XML file or a set of normalized relational tables. The above model has been widely used in works on *entity search* [3], [5] and *data-centric XML retrieval* [8], and has the advantage that it can be easily mapped to both XML and relational data. Further, if a KQI method relies on the intricacies of the database design (e.g. deep syntactic nesting), it will not be robust and will have considerably different degrees of effectiveness over different databases [27]. Hence, since our goal is to develop principled formal models that cover reasonably well all databases and data formats, we do not consider the intricacies of the database design or data format in our models.

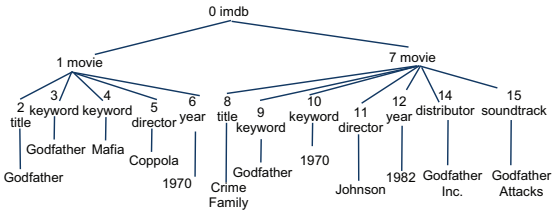

Fig. 1. IMDB database fragment.

A *keyword query* is a set $Q = \{q_1, \ldots, q_{|Q|}\}$ of terms, where $|Q|$ is the number of terms in Q. An entity E is an *answer* to Q iff at least one of its attribute values A contains a term q_i in Q, written $q_i \in A$[1]. Given database DB and query Q, retrieval function $g(E, Q, DB)$ returns a real number that reflects the relevance of entity $E \in DB$ to Q. Given database DB and query Q, a keyword search system returns a ranked list of entities in DB called $L(Q, g, DB)$ where entities E are placed in decreasing order of the value of $g(E, Q, DB)$.

4 Ranking Robustness Principle for Structured Data

In this section we present the *Ranking Robustness Principle*, which argues that there is a (negative) correlation between the difficulty of a query and its ranking robustness in the presence of noise in the data. Section 4.1 discusses how this principle has been applied to unstructured text data. Section 4.2 presents the factors that make a keyword query on structured data difficult, which explain why we cannot apply the techniques developed for unstructured data. The latter observation is also supported by our experiments in Section 8.2 on the *Unstructured Robustness Method* [13], which is a direct adaptation of the Ranking Robustness Principle for unstructured data.

4.1 Background: Unstructured Data

Mittendorf has shown that if a text retrieval method effectively ranks the answers to a query in a collection of text documents, it will also perform well for that query over the version of the collection that contains some errors such as repeated terms [28]. In other words, the degree of the difficulty of a query is positively correlated with the robustness of its ranking over the original and the corrupted versions of the collection. We call this observation the *Ranking Robustness Principle*. Zhou and Croft [13] have applied this principle to predict the degree of the difficulty of a query over free text documents. They compute the similarity between the rankings of the query over the original and the artificially corrupted versions of a collection

1. Some works on keyword search in databases [1] use conjunctive semantics, where all query keywords must appear in a result.

to predict the difficulty of the query over the collection. They deem a query to be more difficult if its rankings over the original and the corrupted versions of the data are less similar. They have empirically shown their claim to be valid. They have also shown that this approach is generally more effective than using methods based on the similarities of probability distributions, that we reviewed in Section 2. This result is especially important for ranking over databases. As we explained in Section 2, it is generally hard to define an effective and domain independent categorization function for entities in a database. Hence, we can use Ranking Robustness Principle as a domain independent proxy metric to measure the degree of the difficulties of queries.

4.2 Properties of Hard Queries on Databases

As discussed in Section 2, it is well established that *the more diverse the candidate answers of a query are, the more difficult the query is* over a collection of the text documents. We extend this idea for queries over databases and propose three sources of difficulty for answering a query over a database as follows:

1) The more entities match the terms in a query, the less specificity of this query and it is harder to answer properly. For example, there are more than one person called *Ford* in the IMDB data set. If a user submits query Q_2: *Ford*, a KQI must resolve the desired *Ford* that satisfy the user's information need. As opposed to Q_2, Q_3: *Spielberg* matches smaller number of people in IMDB, so it is easier for the KQI to return its relevant results.
2) Each attribute describes a different aspect of an entity and defines the context of terms in attribute values of it. If a query matches different attributes in its candidate answers, it will have a more diverse set of potential answers in database, and hence it has higher attribute level ambiguity. For instance, some candidate answers for query Q_4: *Godfather* in IMDB contain its term in their *title* and some contain its term in their *distributor*. For the sake of this example, we ignore other attributes in IMDB. A KQI must identify the desired matching attribute for *Godfather* to find its relevant answers. As opposed to Q_4, query Q_5: *taxi driver* does not match any instance of attribute *distributor*. Hence, a KQI already knows the desired matching attribute for Q_5 and has an easier task to perform.
3) Each entity set contains the information about a different type of entities and defines another level of context (in addition to the context defined by attributes) for terms. Hence, if a query matches entities from more entity sets, it will have higher entity set level ambiguity. For instance, IMDB contains the information about movies in an entity set called *movie* and the information about the people involved in making movies in another entity set called *person*. Consider query Q_6: *divorce* over IMDB data set whose candidate answers come from both entity sets. However, movies about divorce and people who get divorced cannot both satisfy information need of query Q_6. A KQI has a difficult task to do as it has to identify if the information need behind this query is to find people who got divorced or movies about divorce. In contrast to Q_6, Q_7: *romantic comedy divorce* matches only entities from *movie* entity set. It is less difficult for a KQI to answer Q_7 than Q_6 as Q_7 has only one possible desired entity set.

The aforementioned observations show that we may use the statistical properties of the query terms in the database to compute the diversity of its candidate answers and predict its difficulty, like the pre-retrieval predictors introduced in Section 2. One idea is to count the number of possible attributes, entities, and entity sets that contain the query terms to estimate the query specificity and ambiguity and use them to predict the difficulty of the query. The larger this value is the more difficult the query will be. We have shown empirically in Section 8.2 that such approach predicts the difficulty of queries quite poorly. This is because the distribution of query terms over attributes and entity sets may also impact the difficulty of the query. For instance, assume database DB_1 contains two entity sets *book* and *movie* and database DB_2 contains entity sets *book* and *article*. Let term *database* appear in both entity sets in DB_1 and DB_2. Assume that there are far fewer movies that contain term *database* compared to books and articles. A KQI can leverage this property and rank books higher than movies when answering query Q_8: *database* over DB_1. However, it will be much harder to decide the desired entity set in DB_2 for Q_8. Hence, a difficulty metric must take in to account the skewness of the distributions of the query term in the database as well. In Section 5 we discuss how these ideas are used to create a concrete noise generation framework that consider attribute values, attributes and entity sets.

5 A Framework to Measure Structured Robustness

In Section 4 we presented the Ranking Robustness Principle and discussed the specific challenges in applying this principle to structured data. In this section we present concretely how this principle is quantified in structured data.

5.1 Structured Robustness

Corruption of structured data. The first challenge in using the Ranking Robustness Principle for databases is to define data corruption for structured data. For that, we model a database DB using a generative probabilistic model based on its building blocks, which are terms, attribute values, attributes, and entity sets. A corrupted version of DB can be seen as a random sample of such a probabilistic model. Given a query Q and a retrieval function g, we rank the candidate answers in DB and its corrupted versions $DB', DB'', \ldots$ to get ranked lists L and $L', L'', \ldots$, respectively. The less similar L is to $L', L'', \ldots$, the more difficult Q will be.

According to the definitions in Section 3, we model database DB as a triplet $(\mathcal{S}, \mathcal{T}, \mathcal{A})$, where $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{A}$ denote

the sets of entity sets, attributes, and attribute values in DB, respectively. $|\mathcal{A}|$, $|\mathcal{T}|$, $|\mathcal{S}|$ denote the number of attribute values, attributes, and entity sets in the database, respectively. Let V be the number of distinct terms in database DB. Each attribute value $A_a \in \mathcal{A}$, $1 \leq a \leq |\mathcal{A}|$, can be modeled using a V-dimensional multivariate distribution $X_a = (X_{a,1}, \ldots, X_{a,V})$, where $X_{a,j} \in X_a$ is a random variable that represents the frequency of term w_j in A_a. The probability mass function of X_a is:

$$f_{X_a}(\vec{x}_a) = Pr(X_{a,1} = x_{a,1}, \ldots, X_{a,V} = x_{a,V}), \quad (1)$$

where $\vec{x}_a = x_{a,1}, \ldots, x_{a,V}$ and $x_{a,j} \in \vec{x}_a$ are non-negative integers.

Random variable $X_{\mathcal{A}} = (X_1, \ldots, X_{|\mathcal{A}|})$ models attribute value set $\mathcal{A}$, where $X_a \in X_{\mathcal{A}}$ is a vector of size V that denotes the frequencies of terms in A_a. Hence, $X_{\mathcal{A}}$ is a $|\mathcal{A}| \times V$ matrix. The probability mass function for $X_{\mathcal{A}}$ is:

$$\begin{aligned} f_{X_{\mathcal{A}}}(\vec{x}) &= f_{X_{\mathcal{A}}}(\vec{x_1}, \ldots, \vec{x_{|\mathcal{A}|}}) \\ &= Pr(X_1 = \vec{x_1}, \ldots, X_{|\mathcal{A}|} = \vec{x_{|\mathcal{A}|}}), \end{aligned} \quad (2)$$

where $\vec{x_a} \in \vec{x}$ are vectors of size V that contain non-negative integers. The domain of $\vec{x}$ is all $|\mathcal{A}| \times V$ matrices that contain non-negative integers, i.e. $M(|\mathcal{A}| \times V)$.

We can similarly define $X_{\mathcal{T}}$ and $X_{\mathcal{S}}$ that model the set of attributes $\mathcal{T}$ and the set of entity sets $\mathcal{S}$, respectively. The random variable $X_{DB} = (X_{\mathcal{A}}, X_{\mathcal{T}}, X_{\mathcal{S}})$ models corrupted versions of database DB. In this paper, we focus only on the noise introduced in the content (values) of the database. In other words, we do not consider other types of noise such as changing the attribute or entity set of an attribute value in the database. Since the membership of attribute values to their attributes and entity sets remains the same across the original and the corrupted versions of the database, we can derive $X_{\mathcal{T}}$ and $X_{\mathcal{S}}$ from $X_{\mathcal{A}}$. Thus, a corrupted version of the database will be a sample from $X_{\mathcal{A}}$; note that the attributes and entity sets play a key role in the computation of $X_{\mathcal{A}}$ as we discuss in Section 5.2. Therefore, we use only $X_{\mathcal{A}}$ to generate the noisy versions of DB, i.e. we assume that $X_{DB} = X_{\mathcal{A}}$. In Section 5.2 we present in detail how X_{DB} is computed.

Structured Robustness calculation. We compute the similarity of the answer lists using Spearman rank correlation [29]. It ranges between 1 and -1, where 1, -1, and 0 indicate perfect positive correlation, perfect negative correlation, and almost no correlation, respectively. Equation 3 computes the Structured Robustness score (SR score), for query Q over database DB given retrieval function g:

$$\begin{aligned} &SR(Q, g, DB, X_{DB}) \\ &\quad = \mathbb{E}\{Sim(L(Q, g, DB), L(Q, g, X_{DB}))\} \\ &\quad = \sum_{\vec{x}} Sim(L(Q, g, DB), L(Q, g, \vec{x})) f_{X_{DB}}(\vec{x}), \end{aligned} \quad (3)$$

where $\vec{x} \in M(|\mathcal{A}| \times V)$ and Sim denotes the Spearman rank correlation between the ranked answer lists.

5.2 Noise Generation in Databases

In order to compute Equation 3, we need to define the noise generation model $f_{X_{DB}}(M)$ for database DB. *We will show that each attribute value is corrupted by a combination of three corruption levels: on the value itself, its attribute and its entity set.* Now the details: Since the ranking methods for queries over structured data do not generally consider the terms in V that do not belong to query Q [1], [4], we consider their frequencies to be the same across the original and noisy versions of DB. Given query Q, let $\vec{x}$ be a vector that contains term frequencies for terms $w \in Q \cap V$. Similarly to [13], we simplify our model by assuming the attribute values in DB and the terms in $Q \cap V$ are independent. Hence, we have:

$$f_{X_{\mathcal{A}}}(\vec{x}) = \prod_{x_a \in \vec{x}} f_{X_a}(\vec{x_a}). \quad (4)$$

and

$$f_{X_a}(\vec{x}_a) = \prod_{x_{a,j} \in \vec{x}_a} f_{X_{a,j}}(x_{a,j}). \quad (5)$$

where $x_j \in \vec{x}_i$ depicts the number of times w_j appears in a noisy version of attribute value A_i and $f_{X_{i,j}}(x_j)$ computes the probability of term w_j to appear in A_i x_j times.

The corruption model must reflect the challenges discussed in Section 4.2 about search on structured data, where we showed that it is important to capture the statistical properties of the query keywords in the attribute values, attributes and entity sets. We must introduce content noise (recall that we do not corrupt the attributes or entity sets but only the values of attribute values) to the attributes and entity sets, which will propagate down to the attribute values. For instance, if an attribute value of attribute *title* contains keyword *Godfather*, then *Godfather* may appear in any attribute value of attribute *title* in a corrupted database instance. Similarly, if *Godfather* appears in an attribute value of entity set *movie*, then *Godfather* may appear in any attribute value of entity set *movie* in a corrupted instance.

Since the noise introduced in attribute values will propagate up to their attributes and entity sets, one may question the need to introduce additional noise in attribute and entity set levels. The following example illustrates the necessity to generate such noises. Let T_1 be an attribute whose attribute values are A_1 and A_2, where A_1 contains term w_1 and A_2 does not contain w_1. A possible noisy version of T_1 will be a version where A_1 and A_2 both contain w_1. However, the aforementioned noise generation model will not produce such a version. Similarly, a noisy version of entity set S must introduce or remove terms from its attributes and attribute values. According to our discussion in Section 4, we must use a model that generates all possible types of noise in the data.

Hence, we model the noise in a DB as a *mixture* of the noises generated in attribute value, attribute, and entity set levels. Mixture models are typically used to model how the combination of multiple probability distributions generates the data. Let $Y_{t,j}$ be the random variable that represents the frequency of term w_j in attribute T_t. Probability mass function $f_{Y_{t,j}}(y_{t,j})$ computes the probability of w_j to appear $y_{t,j}$ times in T_t. Similarly, $Z_{s,j}$ is the random variable that denotes the frequency of term w_j in entity set S_s and probability mass function $f_{Z_{s,j}}(z_{s,j})$ computes the probability of w_j to appear $z_{s,j}$ times in S_s. Hence, the noise generation

models attribute value A_i whose attribute is T_t and entity set is S_s:

$$\hat{f}_{X_{a,j}}(x_{a,j}) = \gamma_A f_{X_{a,j}}(x_{a,j}) + \gamma_T f_{Y_{t,j}}(x_{t,j}) + \gamma_S f_{Z_{s,j}}(x_{s,j}), \quad (6)$$

where $0 \leq \gamma_A, \gamma_T, \gamma_S \leq 1$ and $\gamma_A + \gamma_T + \gamma_S = 1$. $f_{X_{a,j}}$, $f_{Y_{t,j}}$, and $f_{Y_{s,j}}$ model the noise in attribute value, attribute, and entity set levels, respectively. Parameters γ_A, γ_T and γ_S have the same values for all terms $w \in Q \cap V$ and are set empirically.

Since each attribute value A_a is a small document, we model $f_{X_{i,j}}$ as a Poisson distribution:

$$f_{X_{a,j}}(x_{a,j}) = \frac{e^{-\lambda_{a,j}} \lambda_{a,j}^{x_{a,j}}}{x_{a,j}}. \quad (7)$$

Similarly, we model each attribute T_t, $1 \leq t \leq |\mathcal{T}|$, as a bag of words and use Poisson distribution to model the noise generation in the attribute level:

$$f_{Y_{t,j}}(x_{t,j}) = \frac{e^{-\lambda_{t,j}} \lambda_{t,j}^{x_{t,j}}}{x_{t,j}}. \quad (8)$$

Using similar assumptions, we model the changes in the frequencies of the terms in entity set S_s, $1 \leq s \leq |\mathcal{S}|$, using Poisson distribution:

$$f_{Z_{s,j}}(x_{s,j}) = \frac{e^{-\lambda_{s,j}} \lambda_{s,j}^{x_{s,j}}}{x_{s,j}}. \quad (9)$$

In order to use the model in Equation 6, we have to estimate $\lambda_{A,w}$, $\lambda_{T,w}$, and $\lambda_{S,w}$ for every $w \in Q \cap V$, attribute value A, attribute T and entity set S in DB. We treat the original database as an observed value of the space of all possible noisy versions of the database. Thus, using the maximum likelihood estimation method, we set the value of $\lambda_{A,w}$ to the frequency of w in attribute value A. Assuming that w are distributed uniformly over the attribute values of attribute T, we can set the value of $\lambda_{T,w}$ to the average frequency of w in T. Similarly, we set the value of $\lambda_{S,w}$ as the average frequency of w in S. Using these estimations, we can generate noisy versions of a database according to Equation 6.

5.3 Smoothing The Noise Generation Model

Equation 6 overestimates the frequency of the terms of the original database in the noisy versions of the database. For example, assume a bibliographic database of computer science publications that contains attribute T_2 = *abstract* which constitutes the abstract of a paper. Apparently, many abstracts contain term w_2 = *algorithm*, therefore, this term will appear very frequently with high probability in f_{T_2,w_2} model. On the other hand, a term such as w_3 = *Dirichlet* is very likely to have very low frequency in f_{T_2,w_3} model. Let attribute value A_2 be of attribute *abstract* in the bibliographic DB that contains both w_2 and w_3. Most likely, term *algorithm* will appear more frequently than *Dirichlet* in A_2. Hence, the mean for f_{A_2,w_2} will be also larger than the mean of f_{A_2,w_3}. Thus, a mixture model of f_{T_2,w_2} and f_{A_2,w_2} will have much larger mean than a mixture model of f_{T_2,w_3} and f_{A_2,w_3}. The same phenomenon occurs if a term is relatively frequent in an entity set. Hence, a mixture model such as Equation 6 overestimates the frequency of the terms that are relatively frequent in an attribute or entity set. Researchers have faced a similar issue in language model smoothing for speech recognition [30]. We use a similar approach to resolve this issue. If term w appear in attribute value A, we use only the first term in Equation 6 to model the frequency of w in the noisy version of database. Otherwise, we use the second or third terms if w belongs to T and S, respectively. Hence, the noise generation model is:

$$\hat{f}_{X_{a,j}}(x_{a,j}) = \begin{cases} \gamma_A f_{X_{a,j}}(x_{a,j}) & \text{if } w_j \in A_a \\ \gamma_T f_{Y_{t,j}}(x_{t,j}) & \text{if } w_j \notin A_a,\ w_j \in T_t \\ \gamma_S f_{Z_{s,j}}(x_{s,j}) & \text{if } w_j \notin A_a, T_t,\ w_j \in S_s\ , \end{cases} \quad (10)$$

where we remove the condition $\gamma_A + \gamma_T + \gamma_S = 1$.

5.4 Examples

We illustrate the corruption process and the relationship between the robustness of the ranking of a query and its difficulty using INEX queries $Q9$: *mulan hua animation* and $Q11$: *ancient rome era*, over the IMDB dataset. We set $\gamma_A = 1$, $\gamma_T = 0.9$, $\gamma_S = 0.8$ in Equation 10. We use the XML ranking method proposed in [4], called *PRMS*, which we explain in more detail in Section 6. Given query Q, PRMS computes the relevance score of entity E based on the weighted linear combination of the relevance scores the attribute values of E.

Example of calculation of $\lambda_{t,j}$ for term t = ancient and attribute T_j = plot in Equation 8: In the IMDB dataset, *ancient* occurs in attribute *plot* 2132 times in total, and total number of attribute values under attribute *plot* is 184,731, $\lambda_{t,j} = 2132/184731$ which is 0.0115. Then, since $\gamma_T = 0.9$, the probability that *ancient* occurs k times in a corrupted *plot* attribute is $\frac{0.9e^{-0.0115}(0.0115)^k}{k}$.

Q11: Fig. 2(a) depicts two of the top results (ranked as 1^{st} and 12^{nd} respectively) for query $Q11$ over IMDB. We omit most attributes (shown as elements in XML lingo in Fig. 2(a)) that do not contain any query keywords due to space consideration. Fig. 2(b) illustrates a corrupted version of the entities shown in Fig. 2(a). The new keyword instances are underlined. Note that the ordering changed according to the PRMS ranking. The reason is that PRMS believes that *title* is an important attribute for *rome* (for attribute weighing in PRMS see Section 8.1) and hence having a query keyword (*rome*) there is important. However, after corruption, query word *rome* also appears in the *title* of the other entity, which now ranks higher, because it contains the query words in more attributes.

Word *rome* was added to the *title* attribute of the originally second result through the second level (attribute-based, second branch in Equation 10) of corruption, because *rome* appears in the *title* attribute of other entities in the database. If no *title* attribute contained *rome*, then it could have been added through the third level corruption (entity set-based, third branch in Equation 10) since it appears in attribute values of other *movie* entities.

The second and third levels corruptions typically have much smaller probability of adding a word than the first level, because they have much smaller λ; specifically λ_T is

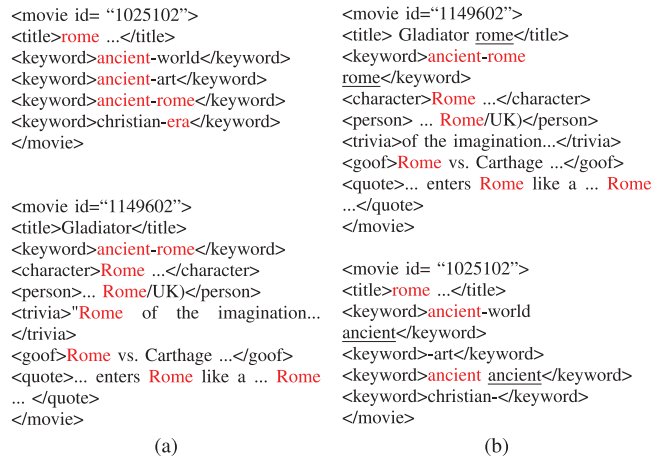

Fig. 2. Original and corrupted results of $Q11$: (a) original ranking. (b) corrupted ranking.

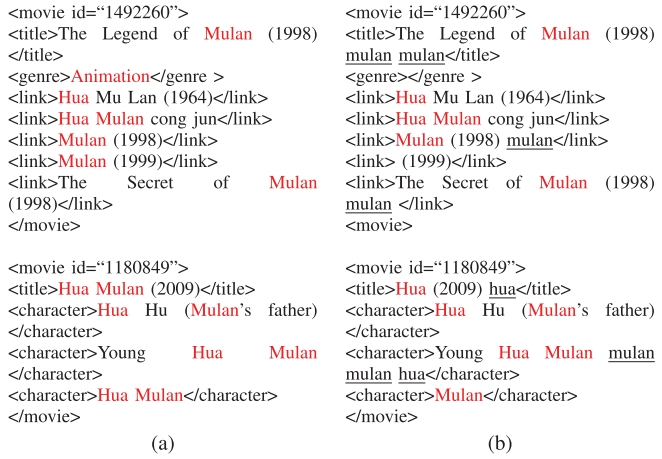

Fig. 3. Original and corrupted results of $Q9$: (a) original ranking. (b) corrupted ranking.

the average frequency of the term in attribute T. However, in hard queries like $Q11$, the query terms are frequent in the database, and also appear in various entities and attributes, and hence λ_T and λ_S are larger.

In the first *keyword* attribute of the top result in Fig. 2, *rome* is added by the first level of corruption, whereas in the *trivia* attribute *rome* is removed by the first level of corruption.

To summarize, $Q11$ is *difficult* because its keywords are spread over a large number of attribute values, attributes and entities in the original database, and also most of the top results have a similar number of occurrences of the keywords. Thus, when the corruption process adds even a small number of query keywords to the attribute values of the entities in the original database, it drastically changes the ranking positions of these entities.

Q9: $Q9$ (*mulan hua animation*) is an *easy* query because most its keywords are quite infrequent in the database. Only term *animation* is relatively frequent in the IMDB dataset, but almost all its occurrences are in attribute *genre*. Fig. 3(a) and (b) present two ordered top answers for $Q9$ over the original and corrupted versions of IMDB, respectively. The two results are originally ranked as 4^{th} and 10^{th}. The attribute values of these two entities contain many query keywords in the original database. Hence, adding and/or removing some query keyword instances in these results, does not considerably change their relevance score and they preserve their ordering after corruption.

Since keywords *mulan* and *hua* appear in a small number of attribute values and attributes, the value of λ for these terms in the second and the third level of corruption is very small. Similarly, since keyword *animation* only appears in the *genre* attribute, the value of λ for all other attributes (second level corruption) is zero. The value of λ for *animation* in the third level is reasonable, 0.0007 for *movie* entity set, but the noise generated in this level alone is not considerable.

6 Efficient Computation of SR Score

A key requirement for this work to be useful in practice is that the computation of the SR score incurs a minimal time overhead compared to the query execution time. In this section we present efficient SR score computation techniques.

6.1 Basic Estimation Techniques

Top-K results: Generally, the basic information units in structured data sets, attribute values, are much shorter than text documents. Thus, a structured data set contains a larger number of information units than an unstructured data set of the same size. For instance, each XML document in the INEX data centric collection constitutes hundreds of elements with textual contents. Hence, computing Equation 3 for a large DB is so inefficient as to be impractical. Hence, similar to [13], we corrupt only the top-K entity results of the original data set. We re-rank these results and shift them up to be the top-K answers for the corrupted versions of DB. In addition to the time savings, our empirical results in Section 8.2 show that relatively small values for K predict the difficulty of queries better than large values. For instance, we found that $K = 20$ delivers the best performance prediction quality in our datasets. We discuss the impact of different values of K in the query difficulty prediction quality more in Section 8.2.

Number of corruption iterations (N): Computing the expectation in Equation 3 for all possible values of $\vec{x}$ is very inefficient. Hence, we estimate the expectation using $N > 0$ samples over $M(|\mathcal{A}|\times V)$. That is, we use N corrupted copies of the data. Obviously, smaller N is preferred for the sake of efficiency. However, if we choose very small values for N the corruption model becomes unstable. We further analyze how to choose the value of N in Section 8.2.

We can limit the values of K or N in any of the algorithms described below.

6.2 Structured Robustness Algorithm

Algorithm 1 shows the Structured Robustness Algorithm (SR Algorithm), which computes the exact SR score based on the top K result entities. Each ranking algorithm uses some statistics about query terms or attributes values over the whole content of DB. Some examples of such statistics are the number of occurrences of a query term in all attributes values of the DB or total number of attribute values in each attribute and entity set. These global statistics

Algorithm 1 $CorruptTopResults(Q, L, M, I, N)$

Input: Query Q, Top-K result list L of Q by ranking function g, Metadata M, Inverted indexes I, Number of corruption iteration N.
Output: SR score for Q.
1: $SR \leftarrow 0$; $C \leftarrow \{\}$; //C caches λ_T, λ_S for keywords in Q
2: **FOR** $i = 1 \rightarrow N$ **DO**
3: $I' \leftarrow I$; $M' \leftarrow M$; $L' \leftarrow L$; //Corrupted copy of I, M and L
4: **FOR** each result R in L **DO**
5: **FOR** each attribute value A in R **DO**
6: $A' \leftarrow A$; //Corrupted versions of A
7: **FOR** each keywords w in Q **DO**
8: Compute # of w in A' by Equation 10; //If $\lambda_{T,w}$, $\lambda_{S,w}$ needed but not in C, calculate and cache them
9: **IF** # of w varies in A' and A **THEN**
10: Update A', M' and entry of w in I';
11: Add A' to R';
12: Add R' to L';
13: Rank L' using g, which returns L, based on I', M';
14: SR += $Sim(L, L')$; //Sim computes Spearman correlation
15: **RETURN** $SR \leftarrow SR/N$; //AVG score over N rounds

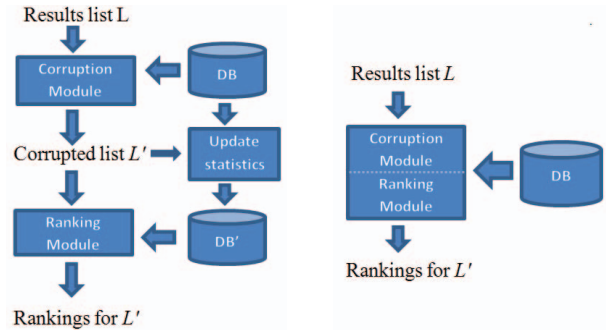

Fig. 4. Execution flows of SR Algorithm and SGS-Approx: (a) SR Algorithm. (b) SGS-Approx.

are stored in M (metadata) and I (inverted indexes) in the SR Algorithm pseudocode.

SR Algorithm generates the noise in the DB on-the-fly during query processing. Since it corrupts only the top K entities, which are anyways returned by the ranking module, it does not perform any extra I/O access to the DB, except to lookup some statistics. Moreover, it uses the information which is already computed and stored in inverted indexes and does not require any extra index.

Nevertheless, our empirical results, reported in Section 8.2, show that SR Algorithm increases the query processing time considerably. Some of the reasons for SR Algorithm inefficiency are the following: First, Line 5 in SR Algorithm loops every attribute value in each top-K result and tests whether it must be corrupted. As noted before, one entity may have hundreds of attribute values. We must note that the attribute values that do not contain any query term still must be corrupted (Line 8-10 in SR Algorithm) for the second and third levels of corruption defined in Equation 10. This is because their attributes or entity sets may contain some query keywords. This will largely increase the number of attribute values to be corrupted. For instance, for IMDB which has only two entity sets, SR Algorithm corrupts all attribute values in the top-K results for all query keywords. Second, ranking algorithms for DBs are relatively slow. SR Algorithm has to re-rank the top K entities N times which is time consuming.

7 Approximation Algorithms

In this section, we propose approximation algorithms to improve the efficiency of SR Algorithm. Our methods are independent of the underlying ranking algorithm.

Query-specific Attribute values Only Approximation (QAO-Approx): QAO-Approx corrupts only the attribute values that match at least one query term. This approximation algorithm leverages the following observations:

Observation 1. *The noise in the attribute values that contain query terms dominates the corruption effect.*

Observation 2. *The number of attribute values that contain at least one query term is much smaller than the number of all attribute values in each entity.*

Hence, we can significantly decrease the time spent on corruption if we corrupt only the attribute values that contain query terms. We add a check before Line 7 in SR Algorithm to test if A contains any term in Q. Hence, we skip the loop in Line 7. The second and third levels of corruption (on attributes, entity sets, respectively) corrupt a smaller number of attribute values so the time spent on corruption becomes shorter.

Static Global Stats Approximation (SGS-Approx): *SGS-Approx* uses the following observation:

Observation 3. *Given that only the top-K result entities are corrupted, the global DB statistics do not change much.*

Fig. 4(a) shows the execution flow of SR Algorithm. Once we get the ranked list of top K entities for Q, the corruption module produces corrupted entities and updates the global statistics of DB. Then, SR Algorithm passes the corrupted results and updated global statistics to the ranking module to compute the corrupted ranking list.

SR Algorithm spends a large portion of the robustness calculation time on the loop that re-ranks the corrupted results (Line 13 in SR Algorithm), by taking into account the updated global statistics. Since the value of K (e.g., 10 or 20) is much smaller than the number of entities in the DB, the top K entities constitute a very small portion of the DB. Thus, the global statistics largely remain unchanged or change very little. Hence, we use the global statistics of the original version of the DB to re-rank the corrupted entities. If we refrain from updating the global statistics, we can combine the corruption and ranking module together. This way re-ranking is done on-the-fly during corruption. *SGS-Approx* algorithm is illustrated in Fig. 4(b).

We use the ranking algorithm proposed in [4], called *PRMS*, to better illustrate the details of our approximation algorithm. PRMS employs a language model approach to search over structured data. It computes the language model of each attribute value smoothed by the language model of its attribute. It assigns each attribute a query keyword-specific weight, which specifies its contribution in the ranking score. It computes the keyword-specific weight $\mu_j(q)$ for attribute values whose attributes are T_j and query q as $\mu_j(q) = \frac{P(q|T_j)}{\sum_{T \in DB} P(q|T)}$. The ranking score of entity E for query Q, $P(Q|E)$ is:

TABLE 2
INEX and SemSearch Datasets Characteristics

	INEX	SemSearch
Size	9.85 GB	9.64 GB
Number of Entities	4,418,081	7,170,445
Number of Entity Sets	2	419,610
Number of Attributes	77	7,869,986
Number of Attribute values	113,603,013	114,056,158

$$P(Q|E) = \prod_{q \in Q} P(q|E) = \prod_{q \in Q} \sum_{j=1}^{n} [\mu_j(q)((1-\lambda)P(q|A_j) + \lambda P(q|T_j))], \quad (11)$$

where A_j is an attribute value of E, T_j is the attribute of A_j, $0 \leq \lambda \leq 1$ is the smoothing parameter for the language model of A_j, and n is the number of attribute values in E. If we ignore the change of global statistics of DB, then μ_j and $P(q|T_j)$ parts will not change when calculating the score of corrupted version of E, E', for q. Hence, the score of E' will depend only on $P(q|A'_j)$, where A'_j is the corrupted version of A_j. We compute the value of $P(q|A'_j)$ using only the information of A'_j as (# of q in A'_j / # of words in A'_j). SGS-Approx uses the global statistics of the original DB to compute μ_j and $P(q|T_j)$ in order to calculate the value of $P(q|E)$. It re-uses them to compute the score of the corrupted versions of E. However, SR Algorithm has to finish all corruption on all attribute values in top results to update the global statistics and re-rank the corrupted results. Similarly, we can modify other keyword query ranking algorithms over DBs that use query term statistics to score entities.

Combination of QAO-Approx and SGS-Approx: QAO-Approx and SGS-Approx improve the efficiency of robustness calculation by approximating different parts of the corruption and re-ranking process. Hence, we combine these two algorithms to further improve the efficiency of the query difficulty predication.

8 Experiments

8.1 Experimental Setting

Data sets: Table 2 shows the characteristics of two data sets used in our experiments. The INEX data set is from the INEX 2010 Data Centric Track [8] discussed in Section 1. The INEX data set contains two entity sets: *movie* and *person*. Each entity in the *movie* entity set represents one movie with attributes like *title*, *keywords*, and *year*. The *person* entity set contains attributes like *name*, *nickname*, and *biography*. The SemSearch data set is a subset of the data set used in Semantic Search 2010 challenge [9]. The original data set contains 116 files with about one billion RDF triplets. Since the size of this data set is extremely large, it takes a very long time to index and run queries over this data set. Hence, we have used a subset of the original data set in our experiments. We first removed duplicate RDF triplets. Then, for each file in SemSearch data set, we calculated the total number of distinct query terms in SemSearch query workload in the file. We selected the 20, out of the 116, files that contain the largest number of query keywords for our experiments. We converted each distinct RDF subject in this data set to an entity whose identifier is the subject identifier. The RDF properties are mapped to attributes in our model. The values of RDF properties that end with substring "#type" indicates the type of a subject. Hence, we set the entity set of each entity to the concatenation of the values of RDF properties of its RDF subject that end with substring "#type". If the subject of an entity does not have any property that ends with substring "#type", we set its entity set to "UndefinedType". We have added the values of other RDF properties for the subject as attributes of its entity. We stored the information about each entity in a separate XML file. We have removed the relevance judgment information for the subjects that do not reside in these 20 files. The sizes of the two data sets are quite close; however, SemSearch is more heterogeneous than INEX as it contains a larger number of attributes and entity sets. The size of both data sets are about 10GB, which is reasonably large for highly structured data sets, especially given that most empirical studies on keyword query processing over databases have been conducted on much smaller datasets [2]–[4].

Query Workloads: Since we use a subset of the dataset from SemSearch, some queries in its query workload may not contain enough candidate answers. We picked the 55 queries from the 92 in the query workload that have at least 50 candidate answers in our dataset. Because the number of entries for each query in the relevance judgment file has also been reduced, we discarded another two queries (Q6 and Q92) without any relevant answers in our dataset, according to the relevance judgment file. Hence, our experiments is done using 53 queries (2, 4, 5, 11-12, 14-17, 19-29, 31, 33-34, 37-39, 41-42, 45, 47, 49, 52-54, 56-58, 60, 65, 68, 71, 73-74, 76, 78, 80-83, 88-91) from the SemSearch query workload. 26 query topics are provided with relevance judgments in the INEX 2010 Data Centric Track. Some query topics contain characters "+" and "−" to indicate the conjunctive and exclusive conditions. In our experiments, we do not use these conditions and remove the keywords after character "−". Some searching systems use these operators to improve search quality. Similar to other efforts in predicting query difficulty, we left supporting these operators to the future work. Generally, KQIs over DBs return candidate answers that contain all terms in the query [1], [6], [27]. However, queries in the INEX query workload are relatively long (normally over four distinct keywords). If we retrieve only the entities that contain all query terms, there will not be sufficient number of (in some cases none) candidate answers for many queries in the data. Hence, for every query Q, we use the following procedure to get at least 1,000 candidate answers for each query. First, we retrieve the entities that contain $|Q|$ distinct terms in query Q. If they are not sufficient, we retrieve the entities that contain at least $|Q| - 1$ distinct query keywords, and so on until we get 1000 candidate answers.

Ranking Algorithms: To evaluate the effectiveness of our model for different ranking algorithms, we have evaluated the query performance prediction model with two representative ranking algorithms: PRMS [4] and IR-Style [1]. Many other algorithms are extensions of these two methods (e.g., [2], [12]).

PRMS: We explained the idea behind PRMS algorithm in Section 6. We adjust parameter λ in PRMS in our experiments to get the best MAP and then use this value of λ for query performance prediction evaluations. Varying λ from 0.1 to 0.9 with 0.1 as the test step, we have found that different values of λ change MAP very slightly on both datasets, and generally smaller λs deliver better MAP. We use $\lambda = 0.1$ on INEX and 0.2 on SemSearch.

IR-Style: We use a variation of the ranking model proposed in [1] for relational data model, referred as IR-Style ranking. Given a query, IR-Style returns a minimal join tree that connects the tuples from different tables in the DB that contain the query terms, called *MTNJT*. However, our datasets are not in relational format and the answers in their relevance judgments files are entities and not *MTNJT*s. Hence, we extend the definition of *MTNJT* as the minimal subtree that connects the attribute values containing the query keywords in an entity. The root of this subtree is the root of the entity in its XML file. If an entity has multiple *MTNJT*s, we choose the one with the maximum score as explained below. Let M be a *MTNJT* tree of entity E and $\mathcal{A}_{\mathcal{M}}$ be the attribute values in M. The score of M for query Q is: $\frac{IRScore(M,Q)}{size(M)}$, where $IRScore(M, Q)$ is the score of M for query Q based on some IR ranking formula. If we use a vector space model ranking formula as in [1] to compute the $IRScore(M, Q)$, we get very low MAP (less than 0.1) for both datasets. Hence, we compute it using a language model ranking formula with Jelink-Mercer smoothing [31] which is shown in equation 12. We set the value of smoothing parameter α to 0.2 as it returns the highest MAP for our datasets.

$$IRScore(M, Q) = \prod_{q \in Q} \sum_{A \in \mathcal{A}_{\mathcal{M}}} ((1 - \alpha)P(q|A) + \alpha P(q|T)). \quad (12)$$

Configuration: We have performed our experiments on an AMD Phenom II X6 2.8 GHz machine with 8 GB of main memory that runs on 64-bit Windows 7. We use Berkeley DB 5.1.25 to store and index the XML files and implement all algorithms in Java.

8.2 Quality Results

In this section, we evaluate the effectiveness of the query quality prediction model computed using SR Algorithm. We use both Pearson's correlation and Spearman's correlation between the SR score and the average precision of a query to evaluate the prediction quality of SR score.

Setting the value of N: Let L and L' be the original and corrupted top-K entities for query Q, respectively. The SR score of Q in each corruption iteration is the Spearman's correlation between L and L'. We corrupt the results N times to get the average SR score for Q. In order to get a stable SR score, the value of N should be sufficiently large, but this increases the computation time of the SR score. We chose the following strategy to find the appropriate value of N: We progressively corrupt L 50 iterations at a time and calculate the average SR score over all iterations. If the last 50 iterations do not change the average SR score over 1%, we terminate. N may vary for different queries in query workloads. Thus, we set it to the maximum number of iterations over all queries. According to our experiments, the value of N varies very slightly for different value of K. Therefore, we set the value of N to 300 on INEX and 250 on SemSearch for all values of K.

Different Values for K: The number of interesting results for a keyword query is normally small [7]. For example, the average number of relevant results is 9.6 for the SemSearch query workload. In this setting, many low ranked answers may not be relevant and have quite close scores, which makes their relative ranking positions very sensitive to noise. If we use large values for K, the SR score will be dominated by the low ranked non-relevant results and the SR score may deem all queries almost equally difficult. Hence, it is reasonable to use small values of K for query performance prediction. We empirically show that our model prefers smaller K on these two datasets. We conduct our experiments on $K = 10$, 20 and 50. All these values deliver reasonable prediction quality (i.e. the robustness of a query is strongly correlated with its effectiveness). However, on both datasets, $K = 10$ and 20 deliver better prediction quality than $K = 50$, given other parameters are the same. For instance, the value of Pearson's correlation on SemSearch is 0.398 for $K = 50$ but 0.471 for $K = 10$ and 0.556 for $K = 20$. We have achieved the best prediction quality using $K = 20$ for both datasets with various combinations of γ_A, γ_T, and γ_S. We present these experiments in details later in this Section.

Training of γ_A, γ_T, and γ_S: We denote the coefficients combination in Equation 10 as (γ_A, γ_T, γ_S) for brevity.

We train (γ_A, γ_T, γ_S) using 5-fold cross validation. We get two settings on each dataset by using Spearman's correlation and Pearson's correlation, respectively, to measure the prediction quality. After some preliminary experiments, we found that large γ_A is effective. Hence, to reduce the number of possible combinations, we fix γ_A as 1, and vary the other two during the training process to find the highest Pearson's correlation between average precision and SR score. We computed the SR score for γ_T and γ_S from 0 to 3 with step 0.1 for different values of K. We found that the value of (γ_A, γ_T, γ_S) that leads to the best correlation score, is quite stable on different training sets. In the rest of the paper, we report the results of Pearson's correlation and Spearman's correlation over INEX using the values of (1, 0.9, 0.8) and (1, 0.3, 0.5) for (γ_A, γ_T, γ_S), respectively. We also present the values of Pearson's correlation and Spearman's correlation on SemSearch that are achieved by the setting (γ_A, γ_T, γ_S) to (1, 0.1, 0.6).

Figs. 5 and 6 depict the plot of average precision and SR score for all queries in our query workload on INEX and SemSearch, respectively. In Fig. 5, we see that $Q9$ is easy (has high average precision) and $Q11$ is relatively hard, as discussed in Section 5.4. As shown in Fig. 6, query $Q78$: *sharp-pc* is easy (has high average precision), because its keywords appear together in few results, which explains its high SR score. On the other hand, $Q19$: *carl lewis* and $Q90$: *university of phoenix* have a very low average precision as their keywords appear in many attributes and entity sets. Fig. 6 shows that the SR scores of these queries are very small, which confirms our model.

Baseline Prediction Methods: We use Clarity score (CR) [10], Unstructured Robustness Method (URM) [13],

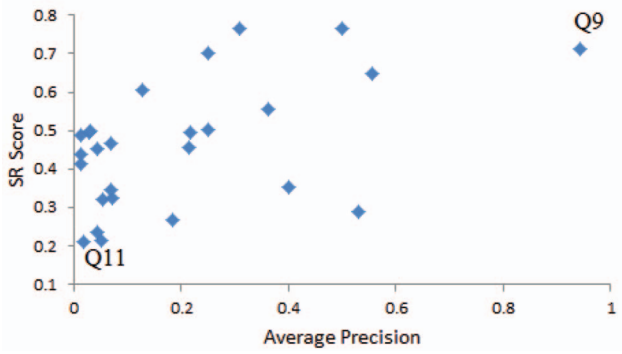

Fig. 5. Average precision versus SR score for queries on INEX using PRMS, $K = 20$ and $(\gamma_A, \gamma_T, \gamma_S) = (1, 0.3, 0.5)$.

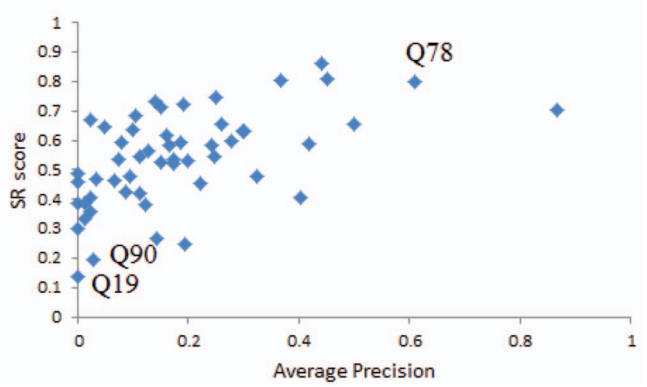

Fig. 6. Average precision versus SR score for queries on SemSearch using PRMS, $K = 20$ and $(\gamma_A, \gamma_T, \gamma_S) = (1, 0.1, 0.6)$.

Weighted Information Gain (WIG) [17], normalized-query-commitment (NQC) [16], and prevalence of query keywords as baseline query difficulty prediction algorithms in databases. CR, URM are two popular post-retrieval query difficulty prediction techniques over text documents. WIG and NQC are also post-retrieval predictors that have been proposed recently and are shown to achieve better query difficulty prediction accuracy than CR and URM [16], [17].

To implement CR, URM, WIG, and NQC, we concatenate the XML elements and tags of each entity into a text document and assume all entities (now text documents) belong to one entity set. The values of all μ_j in PRMS ranking formula are set to 1 for every query term. Hence, PRMS becomes a language model retrieval method for text documents [7]. We have separately trained the parameters of these method on each dataset using the whole query workload as the training data to get the optimal settings for these methods.

URM and CR: For URM on both datasets, we corrupted each ranking list 1000 times such that robustness score becomes stable. For CR, we trained three parameters: the number of results (k), the vocabulary size (v) used in computing query language model, and the background language model smoothing factor (λ). We report the results for CR using $k = 100$ and $\lambda = 0.7$ for INEX and $k = 500$ and $\lambda = 0.3$ for SemSearch. We have use the whole vocabulary to compute the query language model in both datasets.

WIG and NQC: In order to make a reasonable comparison, we have used the implementations of WIG and NQC from [16]. We trained the number of of top results, k, for both methods . For WIG, we set $k = 5$ on SemSearch and $k = 10$ on INEX. For NQC, we set $k = 10$ on SemSearch and $k = 150$ on INEX. We think smaller k is prefered on SemSearch for both methods because its query workload has smaller average number of relevant results per query.

Prevalence of Query Keywords: As we argued in Section 4.2, if the query keywords appear in many entities, attributes, or entity sets, it is harder for a ranking algorithm to locate the desired entities. Given query Q, we compute the average number of attributes ($AA(Q)$), average number of entity sets ($AES(Q)$), and the average number of entities ($AE(Q)$) where each keyword in Q occurs. We consider each of these three values as an individual baseline difficulty prediction metric. We also multiply these three metrics (to avoid normalization issues that summation would have) and create another baseline metric, denoted as $AS(Q)$. Intuitively, if these metrics for query Q have higher values, Q must be harder and have lower average precision. Thus, we use the inverse of these values, denoted as $iAA(Q)$, $iAES(Q)$, $iAE(Q)$, and $iAS(Q)$, respectively.

Comparison to Baseline Methods: Tables 3 and 4 show Pearson's and Spearman's correlation values between average precision and the metrics for SR, NQC, WIG, URM, CR, iAA(Q), iAES(Q), iAE(Q), and iAS(Q) methods for different values of K over both datasets, respectively. These results are based on all queries in the query workloads without distinguishing between training and testing sets. The *n/a* value appears in Table 3 because all query keywords in our query workloads occur in both entity sets in INEX.

The Pearson's and Spearman's correlation scores for SR Algorithm are significantly higher than those of URM, CR and WIG on both datasets for both cases of $K = 10$ and $K = 20$.

TABLE 3
Pearson's Correlation of Average Precision Against Each Metric

K	10									20								
Method	SR	WIG	NQC	URM	CR	iAA	iAES	iAE	iAS	SR	WIG	NQC	URM	CR	iAA	iAES	iAE	iAS
INEX	**0.486**	0.176	0.302	0.093	0.266	0.299	n/a	0.111	0.143	**0.564**	0.187	0.262	0.177	0.257	0.370	n/a	0.255	0.292
SemSearch	**0.471**	0.107	-0.083	0.247	0.111	0.066	0.052	0.040	-0.043	**0.556**	0.109	-0.079	0.311	0.119	0.082	0.068	0.056	-0.046

TABLE 4
Spearman's Correlation of Average Precision Against Each Metric

K	10									20								
Method	SR	WIG	NQC	URM	CR	iAA	iAES	iAE	iAS	SR	WIG	NQC	URM	CR	iAA	iAES	iAE	iAS
INEX	0.303	0.242	0.381	0.196	0.199	**0.409**	n/a	-0.167	0.187	**0.475**	0.218	0.319	0.270	0.202	0.448	n/a	-0.154	0.174
SemSearch	**0.519**	0.270	0.287	-0.012	0.182	0.334	0.282	0.289	0.306	**0.576**	0.253	0.271	0.074	0.179	0.348	0.302	0.310	0.326

TABLE 5
Effect of K on Correlation of Average Precision and SR Score Using IR-Style Ranking on SemSearch

K	10	20
Pearson's correlation score	0.565	0.544
Spearman's correlation score	0.502	0.507

TABLE 6
Average Query Processing Time and Average Robustness Computation for $K = 20$

	Avg Q-time (ms)	Avg SR-time (ms)
INEX (N=250)	24,177	(88,271 + 29,585)
SemSearch (N=300)	46,726	(11,121 + 12,110)

SR Algorithm delivers a higher Pearson's and Spearman's correlation than NQC over both data sets for both values cases of $K = 20$, and higher Pearson's correlation over both data sets for the case of $K = 10$. SR algorithm also provides a higher Pearson's correlation than NQC over SemSearch for $K = 10$. This shows that our prediction model is generally more effective than other methods over databases. Especially, the large improvement over URM confirms that our corruption model better captures the properties of difficult queries on databases. iAA provides a more accurate prediction than all other baseline methods over INEX but slightly worse than NQC in terms of Spearman's correlation for $K = 10$. This indicates that one of the main causes of the difficulties for the queries over the INEX dataset is to find their desired attributes, which confirms our analysis in Section 4.2. SR also delivers far better prediction qualities than iAA(Q), iAES(Q), iAE(Q), and iAS(Q) metrics over both data sets. Hence, SR effectively considers all causes of the difficulties for queries over databases.

IR-style ranking algorithm: The best value of MAP for the IR-Style ranking algorithm over INEX is 0.134 for $K = 20$, which is very low. Note that we tried both Equation 12 as well as the vector space model originally used in [1]. Thus, we do not study the quality performance prediction for IR-Style ranking algorithm over INEX. On the other hand, the IR-Style ranking algorithm using Equation 12 delivers larger MAP value than PRMS on the SemSearch dataset. Hence, we only present results on SemSearch. Table 5 shows Pearson's correlation of SR score with the average precision for different values of K, for $N = 250$ and $(\gamma_A, \gamma_T, \gamma_S) = (1, 0.1, 0.6)$. Fig. 7 plots SR score against the average precision when $K = 20$.

Discussion: Without combining with other predictors, all state-of-the-art predictors on text collections achieve linear/non-linear correlation between average precision and prediction metrics up to 0.65 depending on corpus and query workload [10], [13]–[17], [20], [24], [25], [32]. As the first work in query difficulty prediction on database, we believe our prediction quality is reasonably high.

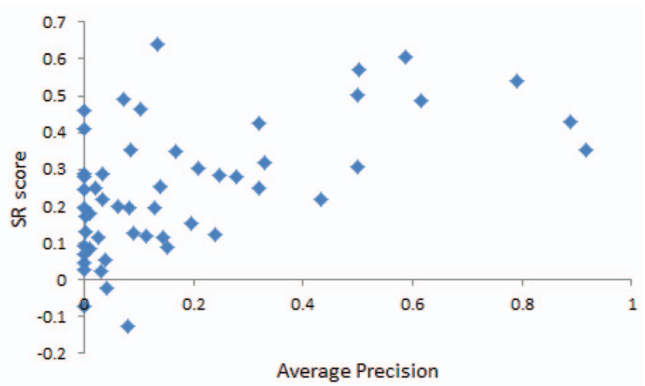

Fig. 7. Average precision versus SR score using IR-Style over SemSearch with $K = 20$.

Impact of database schema complexity: On one hand, increasing the complexity of the schema (e.g., increasing nesting or number attributes) makes it harder to locate the user-desired results. On the other hand, a richer structure may improve the quality of search if the system is able to locate the right attribute types, e.g., when the keywords only appear in a single attribute type. For the same reasons we believe there is no general rule on the effect of the schema complexity on the effectiveness of SR score.

Using SR Algorithm: Similar to other difficulty metrics, given the result of SR Algorithm for an input query, a KQI may apply a thresholding approach to categorize an input query as "easy" or "hard" [10]. This thresholding approach defines a reasonable threshold t for a query difficulty metric. If the difficulty metric of the query is below t, it will be considered a "hard" query, and the KQI will apply further treatments like the ones discussed in Section 1 to it. One may apply the kernel density estimation technique proposed in [10] to find the value of t for a database. This technique computes the SR score for a large number of syntactic keyword queries that are randomly sampled from the database. It then sets the value of t to the SR score that is estimated to be less than the SR values of 80% of queries. Readers can find more information on the justification and implementation of this approach in [10].

8.3 Performance Study

In this section we study the efficiency of our SR score computation algorithms.

SR Algorithm: We report the average computation time of SR score (SR-time) using SR Algorithm and compare it to the average query processing time (Q-time) using PRMS for the queries in our query workloads. These times are presented in Table 6 for $K = 20$. SR-time mainly consists of two parts: the time spent on corrupting K results and the time to re-rank the K corrupted results. We have reported SR-time using (corruption time + re-rank time) format. We see that SR Algorithm incurs a considerable time overhead on the query processing. This overhead is higher for queries over the INEX dataset, because there are only two entity sets, (*person* and *movie*), in the INEX dataset, and all query keywords in the query load occur in both entity sets. Hence, according to Equation 10, every attribute value in top K entities will be corrupted due to the third level of corruption. Since SemSearch contains far more entity sets and attributes than INEX, this process does not happen for SemSearch.

QAO-Approx: Figs. 8(a) and 9(a) show the results of using QAO-Approx on INEX and SemSearch, respectively. We measure the prediction effectiveness for smaller values of N using average correlation score. The QAO-Approx

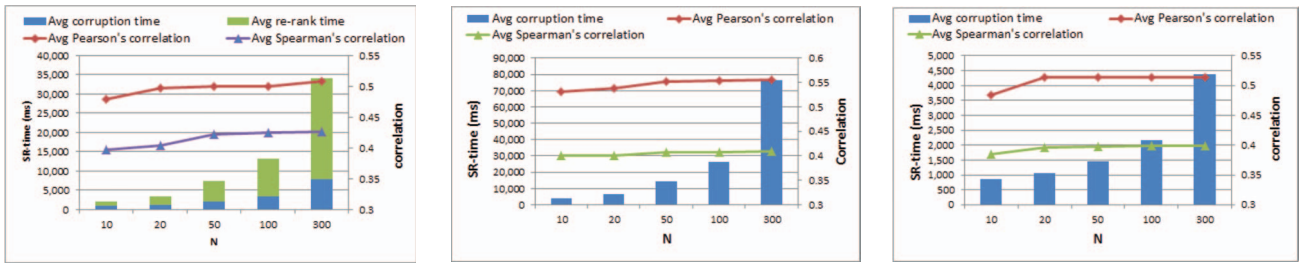

Fig. 8. Approximations on INEX: (a) QAO-Approx. (b) SGS-Approx. (c) Combination of QAO and SGS.

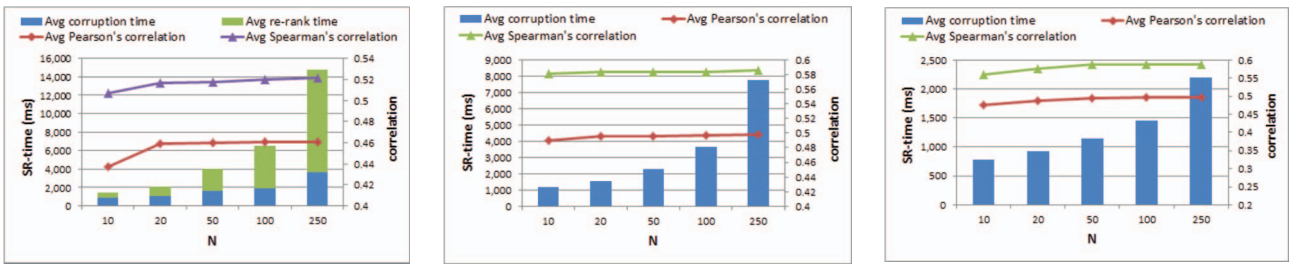

Fig. 9. Approximations on SemSearch: (a) QAO-Approx. (b) SGS-Approx. (c) Combination of QAO and SGS.

algorithm delivers acceptable correlation scores and the corruption times of about 2 seconds for $N = 10$ on INEX and $N = 20$ on SemSearch. Comparing to the results of SR Algorithm for $N = 250$ on SemSearch and $N = 300$ on INEX, the Pearson's correlation score drops, because less noise is added by second and third level corruption. These results show the importance of these two levels of corruption.

SGS-Approx: Figs. 8(b) and 9(b) depict the results of applying SGS-Approx on INEX and SemSearch, respectively. Since re-ranking is done on-the-fly during the corruption, SR-time is reported as corruption time only. As shown in Fig. 8(b), the efficiency improvement on the INEX dataset is slightly worse than QAO-Approx, but the quality (correlation score) remains high. SGS-Approx outperforms QAO-Apporx in terms of both efficiency and effectiveness on the SemSearch dataset.

Combination of QAO-Approx and SGS-Approx: As noted in Section 6, we can combine QAO-Approx and SGS-Approx algorithms to achieve better performance. Figs. 8(c) and 9(c) present the results of the combined algorithm for INEX and SemSearch databases, respectively. Since we use SGS-Approx, the SR-time consists only of corruption time. Our results show that the combination of two algorithms works more efficiently than either of them with the same value for N.

Summary of the Performance Results: According to our performance study of QAO-Approx, SGS-Approx, and the combined algorithm over both datasets, the combined algorithm delivers the best balance of improvement in efficiency and reduction in effectiveness for both datasets. On both datasets, the combined algorithm achieves high prediction accuracy (the Pearson's correlation score about 0.5) with SR-time around 1 second. Using the combined algorithm over INEX when the value of N is set to 20, the the Pearson's and Spearman's correlation scores are 0.513 an 0.396 respectively and the time decreases to about 1 second. For SR Algorithm on INEX, when N decreases to 10, the Pearson's correlation is 0.537, but SR-time is over 9.8 seconds, which is not ideal. If we use the combined algorithm on SemSearch, the Pearson's and Spearman's correlation scores are 0.495 and 0.587 respectively and SR-time is about 1.1 seconds when $N = 50$. However, to achieve a similar running time, SGS-Approx needs to decrease N to 10, with the SR-time of 1.2 seconds, the Pearson's correlation of 0.49 and the Spearman's correlation of 0.581. Thus, the combined algorithm is the best choice to predict the difficulties of queries both efficiently and effectively.

Discussion: The time to compute the SR score only depends on the top-K results, since only the top-K results are corrupted and reranked (see Section 6). Increasing the data set size will only increase the query processing time, which is not the focus of this paper.

The complexity of data schema could have impact on the efficiency of our model. A simpler schema may not mean shorter SR computation time, since more attribute values need to be corrupted, since more attribute values of the same attribute type of interest exists. The latter is supported by the longer corruption times incurred by INEX, which has simpler schema than SemSearch, as shown in Table 6.

9 Conclusion

We introduced the novel problem of predicting the effectiveness of keyword queries over DBs. We showed that the current prediction methods for queries over unstructured data sources cannot be effectively used to solve this problem. We set forth a principled framework and proposed novel algorithms to measure the degree of the difficulty of a query over a DB, using the ranking robustness principle. Based on our framework, we propose novel

algorithms that efficiently predict the effectiveness of a keyword query. Our extensive experiments show that the algorithms predict the difficulty of a query with relatively low errors and negligible time overheads.

Acknowledgments

A. Termehchy is supported by US NSF grants 0938071, 076532, and 0938064 and a Yahoo! Key Scientific Challenges Award. V. Hristidis is supported by US NSF grants IIS-1216032 and IIS-1216007.

References

[1] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient IR-style keyword search over relational databases," in *Proc. 29th VLDB Conf.*, Berlin, Germany, 2003, pp. 850–861.
[2] Y. Luo, X. Lin, W. Wang, and X. Zhou, "SPARK: Top-k keyword query in relational databases," in *Proc. 2007 ACM SIGMOD*, Beijing, China, pp. 115–126.
[3] V. Ganti, Y. He, and D. Xin, "Keyword++: A framework to improve keyword search over entity databases," in *Proc. VLDB Endowment*, Singapore, Sept. 2010, vol. 3, no. 1–2, pp. 711–722.
[4] J. Kim, X. Xue, and B. Croft, "A probabilistic retrieval model for semistructured data," in *Proc. ECIR*, Tolouse, France, 2009, pp. 228–239.
[5] N. Sarkas, S. Paparizos, and P. Tsaparas, "Structured annotations of web queries," in *Proc. 2010 ACM SIGMOD Int. Conf. Manage. Data*, Indianapolis, IN, USA, pp. 771–782.
[6] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword searching and browsing in databases using BANKS," in *Proc. 18th ICDE*, San Jose, CA, USA, 2002, pp. 431–440.
[7] C. Manning, P. Raghavan, and H. Schütze, *An Introduction to Information Retrieval*. New York, NY: Cambridge University Press, 2008.
[8] A. Trotman and Q. Wang, "Overview of the INEX 2010 data centric track," in *9th Int. Workshop INEX 2010*, Vugh, The Netherlands, pp. 1–32,
[9] T. Tran, P. Mika, H. Wang, and M. Grobelnik, "Semsearch Ś10," in *Proc. 3rd Int. WWW Conf.*, Raleigh, NC, USA, 2010.
[10] S. C. Townsend, Y. Zhou, and B. Croft, "Predicting query performance," in *Proc. SIGIR '02*, Tampere, Finland, pp. 299–306.
[11] A. Nandi and H. V. Jagadish, "Assisted querying using instant-response interfaces," in *Proc. SIGMOD 07*, Beijing, China, pp. 1156–1158.
[12] E. Demidova, P. Fankhauser, X. Zhou, and W. Nejdl, "DivQ: Diversification for keyword search over structured databases," in *Proc. SIGIR' 10*, Geneva, Switzerland, pp. 331–338.
[13] Y. Zhou and B. Croft, "Ranking robustness: A novel framework to predict query performance," in *Proc. 15th ACM Int. CIKM*, Geneva, Switzerland, 2006, pp. 567–574.
[14] B. He and I. Ounis, "Query performance prediction," *Inf. Syst.*, vol. 31, no. 7, pp. 585–594, Nov. 2006.
[15] K. Collins-Thompson and P. N. Bennett, "Predicting query performance via classification," in *Proc. 32nd ECIR*, Milton Keynes, U.K., 2010, pp. 140–152.
[16] A. Shtok, O. Kurland, and D. Carmel, "Predicting query performance by query-drift estimation," in *Proc. 2nd ICTIR*, Heidelberg, Germany, 2009, pp. 305–312.
[17] Y. Zhou and W. B. Croft, "Query performance prediction in web search environments," in *Proc. 30th Annu. Int. ACM SIGIR*, New York, NY, USA, 2007, pp. 543–550.
[18] Y. Zhao, F. Scholer, and Y. Tsegay, "Effective pre-retrieval query performance prediction using similarity and variability evidence," in *Proc. 30th ECIR*, Berlin, Germany, 2008, pp. 52–64.
[19] C. Hauff, L. Azzopardi, and D. Hiemstra, "The combination and evaluation of query performance prediction methods," in *Proc. 31st ECIR*, Toulouse, France, 2009, pp. 301–312.
[20] C. Hauff, V. Murdock, and R. Baeza-Yates, "Improved query difficulty prediction for the Web," in *Proc. 17th CIKM*, Napa Valley, CA, USA, 2008, pp. 439–448.
[21] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*. San Francisco, CA: Morgan Kaufmann, 2011.
[22] E. Yom-Tov, S. Fine, D. Carmel, and A. Darlow, "Learning to estimate query difficulty: Including applications to missing content detection and distributed information retrieval," in *Proc. 28th Annu. Int. ACM SIGIR Conf. Research Development Information Retrieval*, Salvador, Brazil, 2005, pp. 512–519.
[23] J. A. Aslam and V. Pavlu, "Query hardness estimation using Jensen-Shannon divergence among multiple scoring functions," in *Proc. 29th ECIR*, Rome, Italy, 2007, pp. 198–209.
[24] O. Kurland, A. Shtok, S. Hummel, F. Raiber, D. Carmel, and O. Rom, "Back to the roots: A probabilistic framework for query-performance prediction," in *Proc. 21st Int. CIKM*, Maui, HI, USA, 2012, pp. 823–832.
[25] O. Kurland, A. Shtok, D. Carmel, and S. Hummel, "A Unified framework for post-retrieval query-performance prediction," in *Proc. 3rd Int. ICTIR*, Bertinoro, Italy, 2011, pp. 15–26.
[26] S. Cheng, A. Termehchy, and V. Hristidis, "Predicting the effectiveness of keyword queries on databases," in *Proc. 21st ACM Int. CIKM*, Maui, HI, 2012, pp. 1213-1222.
[27] A. Termehchy, M. Winslett, and Y. Chodpathumwan, "How schema independent are schema free query interfaces?" in *Proc. IEEE 27th ICDE*, Hannover, Germany, 2011, pp. 649–660.
[28] E. Mittendorf and P. Schauble, "Measuring the effects of data corruption on information retrieval," in *Proc. SDAIR 96 Conf.*, Las Vegas, NV, USA, 1996, pp. 179–189.
[29] J. D. Gibbons and S. Chakraborty, *Nonparametric Statistical Inference*. New York, NY: Marcel Dekker, 1992.
[30] S. M. Katz, "Estimation of probabilistic from sparse data for the language model component of a speech recognizer," *IEEE Trans. Signal Process.*, vol. 35, no. 3, pp. 400–401, Mar. 1987.
[31] C. Zhai and J. Lafferty, "A study of smoothing methods for language models applied to ad hoc information retrieval," in *Proc. 24th Annu. Int. ACM SIGIR Conf. Research Development Information Retrieval*, New Orleans, LA, USA, 2001, pp. 334–342.
[32] C. Hauff, L. Azzopardi, D. Hiemstra, and F. Jong, "Query performance prediction: Evaluation contrasted with effectiveness," in *Proc. 32nd ECIR*, Milton Keynes, U.K., 2010, pp. 204–216.

Shiwen Cheng is a Ph.D. candidate in the Computer Science and Engineering Department at UC Riverside. He received the B.Sc. degree in computer science from Huazhong University of Science and Technology, China, in 2007. His current research interests include information retrieval and databases.

Arash Termehchy is an Assistant Professor in the School of Electrical Engineering & Computer Science at Oregon State University, Corvallis, OR, USA. His current research interests include the areas of database systems and data mining. He is the recipient of the Best Student Paper Award of ICDE 2011, Best Papers Selection of ICDE 2011, the Yahoo! Key Scientific Challenges Award in 2011–2012, and Feng Chen Memorial Award in 2012. He received the Ph.D. degree from the University of Illinois at Urbana-Champaign.

Vagelis Hristidis is an Associate Professor of Computer Science at UC Riverside. His current research interests include databases, information retrieval, and particularly the intersection of these two areas. His key achievements include the US NSF CAREER Award, a Google Research Award, an IBM Award, and a Kauffmann Entrepreneurship Award.