Predicting the Effectiveness of Keyword Queries on Databases

Shiwen Cheng
UC, Riverside
schen064@cs.ucr.edu

Arash Termehchy
University of Illinois, Urbana
termehch@uiuc.edu

Vagelis Hristidis
UC, Riverside
vagelis@cs.ucr.edu

ABSTRACT

Keyword query interfaces (KQIs) for 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 be able 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 devise efficient algorithms to compute the degree of difficulty at query-time, and show that the overhead is very small compared to the query execution time. We evaluate our query difficulty prediction model against two relevance judgment benchmarks for keyword search on databases, INEX and SemSearch. Our study shows that our model predicts the hard queries with high accuracy.

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 databases [4, 13, 15, 9, 18, 26]. 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 [4, 13, 9, 3, 26].

Databases contain entities, and entities contain attributes that take attribute values. Some of the difficulties of answering a keyword 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 keyword 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 [20]. 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.

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 [31] where keyword query interfaces are evaluated over the well-known IMDB data set that contains structured information about movies and people in show business. Keyword queries were provided by participants of the workshop. Another effort is the series of Semantic Search Challenges (SemSearch) at Semantic Search Workshop [29, 30], 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 keyword queries are taken from Yahoo! keyword query log. Users have provided relevance judgments for both benchmarks.

A widely accepted metric to measure the ranking quality of a ranking method over a set of queries is Mean Average Precision (MAP), which is a number between zero and one, and captures both the precision and recall of a ranking. The larger this number is the higher the ranking quality of a search method is over the set of keyword queries. The MAP of the best performing method(s) in the last data-centric track in INEX Workshop and Semantic Search Challenge for keyword queries are about 0.36 and 0.2, respectively. The lower MAP values of methods in Semantic Search Challenge are mainly due to the larger size and more heterogeneity of its data set.

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 *may the force be with you* over the IMDB data set. Users would like to see the information about the movies where the phrase "may the force be with you" is cited. For this query, all methods return rankings of considerably lower qualities than their average ranking qualities over all queries. Hence, there are some queries that are much more difficult than others. Moreover, no matter which ranking method is used, we cannot deliver a reasonable ranking for these queries. Such a trend has been also observed for keyword queries over text document collections [32, 28].

It is important for a KQI to recognize such queries and warn the user or employ alternative techniques like query reformulation or query suggestions [22]. It may also use techniques such as diversification of its returned ranked list [33, 6]. On the other hand, if a KQI would employ these techniques for queries with high-quality results, it may hurt their quality and/or waste computational resources. Hence, it is important that a KQI distinguishes difficult from easy queries and act upon them accordingly (the latter is out of the scope of this work).

To the best of our knowledge, there has not been any work on predicting or analyzing the difficulties of keyword queries over databases. Researchers have proposed some methods to detect difficult queries over plain text document collections [28, 37, 25]. 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 experimentally show that direct adaptations of these techniques are ineffective for structured data.

In this paper, we analyze the characteristics of difficult keyword 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 keyword queries over databases. We also analyze the reasons that make a keyword query difficult to answer by KQIs.
- We propose the *Structured Robustness (SR)* score, which measures the difficulty of a keyword 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.
- We introduce efficient algorithms to compute the SR score, given that such a measure is only useful when it can be computed with a small cost overhead compared to the query execution cost. Our most efficient algorithm accurately estimates the SR score by combining the two seemingly independent steps (corruption and re-ranking) into a single step.
- 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.

In the remainder of the paper, Section 2 discusses related work and Section 3 presents basic definitions. Section 4 explains the ranking robustness principle and analyzes the properties of difficult queries over databases. Section 5 presents concrete methods to compute the SR score in (semi-) structured data. Section 6 provides efficient algorithms to compute the SR score. Section 7 contains the experimental results, and Section 8 concludes the paper.

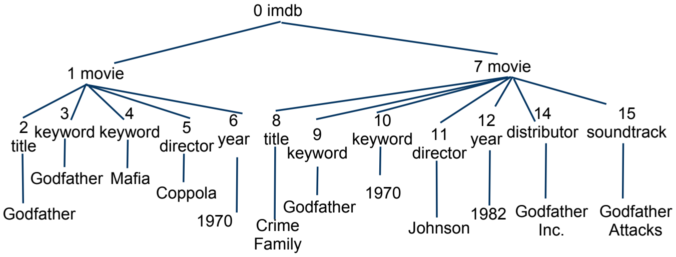


Figure 1: IMDB database fragment

2. RELATED WORK

In this section we present an overview of the works on predicting the query difficulty in free text collections and explain why they generally cannot be applied to our setting. Researchers have proposed methods to predict hard queries over unstructured text documents [28, 32, 37, 12, 25, 5, 34]. Some methods use the statistical properties of the terms in the query to predict its difficulty. Examples of such statistical characteristics are average inverse document frequency of the terms in the query and number of documents that contain at least one query term [12]. The common idea behind these methods is that the more discriminative the query terms are, the easier the query will be. Empirical evaluations indicate that these methods have limited prediction accuracies [28, 11].

A popular approach called *clarity score* argues that an easy query is sufficiently distinctive to separate the user's desired documents from other documents in the collection [28, 11, 5]. Hence, its top ranked answers belong to very few topics that are very likely to be the desired topics. On the other hand, the top ranked documents of a difficult query describe various topics, which many of them are irrelevant to the user's information need. Consider a set of documents that contain the information about different types of news. The top ranked documents of query *European financial crises* are mainly about financial news, but the top ranked answers for query *European crises* may describe several topics such as political, financial, and social news. The latter is more difficult than the former. Researchers have shown that this method provides a better estimation of the difficulty of a query for text documents than clues such as number of terms in the query or inverse document frequencies of its terms [28, 11]. In order to measure the number of topics in the top ranked document of a given query, some systems compare the probability distribution of terms in the top ranked documents with the probability distribution of terms in the whole collection. If these probability distributions are relatively similar, the top ranked documents contain the information about almost as many topics as the whole collection, thus, the query is difficult [28, 11, 25].

Each text document normally contains information about very few topics. If a text document is relevant to a query, we can assume that almost all its topics are relevant to the query. However, each tuple in a rich database, like IMDB, covers a large number of aspects about an entity. If an entity is relevant to a query, it is not clear which aspects of the entity satisfy the information need behind the query. For instance, assume that the movie rooted at node 1 in Figure 1 is relevant to query Q_1. This movie can be categorized in categories such as *movies about Godfather*, *movies directed by Johnson*, *movies produced by Godfather*, and so on. It is not reasonable to assume users are equally interested in all categories of a relevant entity. Moreover, a term might represent different pieces of information if it appears in different attributes or entity sets. For example, term *Godfather* in *keyword* attribute of a movie conveys a different concept from its occurrence in *distributor* attribute in the IMDB database. Therefore, a straightforward extension of the clarity score will poorly predict the difficulty of queries over databases. One way is to address this issue is to compute the probability distributions of terms over top ranked entities and all entities in the database as the weighted linear combination of the probability distributions of terms over the attributes values of these entities. The attribute values of the attributes that describe more interesting and popular aspects of an entity will be assigned larger weights. However, it is very hard to find such weights for the attributes without any domain knowledge about the database and its users.

Some systems use a pre-computed set of topics and assign each document to at least one topic in the set in order to compute the clarity score [5]. They compare the probability distribution of topics in the top ranked documents with the probability distribution of topics of the whole collection to predict the degree of the difficulty of the query. One requires domain knowledge about the data sets and its users to create a set of useful topics for the tuples in the database. We like to find an effective and domain independent approach to predict the difficulties of queries.

Some methods use machine learning techniques to learn the properties of difficult queries and predict them [35, 34]. They have similar limitations as the other approaches when applied to structured data. Moreover, their applications depend on the amount and quality of the training data. Sufficient and high quality training data is not normally available for many databases.

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. Figure 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 Figure 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* [23, 7] and *data-centric XML retrieval* [31], 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.

A *keyword query* is a set $Q = \{q_1 \cdots q_{|Q|}\}$ 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 7.2 on the *Unstructured Robustness Method* [37], which is a direct adaptation of the Ranking Robustness Principle for unstructured data.

4.1 Background: Unstructured Data

Researchers have shown that the effectiveness of a document retrieval system is positively correlated with its ranking stability in the presence of noise in the data [2]. Further, Mittendorf has shown that if a text retrieval method effectively ranks the text documents in a collection, it will also perform well over the version of the collection that contains some errors such as repeated terms [21]. More formally, given scoring function g, query Q, document collection C, and its corrupted version C', let $L(g, Q, C)$ and $L(g, Q, C')$ be the ranked list of candidate answers returned by g for Q over C and C', respectively. Given documents $D_1, D_2 \in L(q, Q, C)$, where $g(D_1, Q, C) > g(D_2, Q, C)$, Mittendorf has shown that the probability that $g(D_1, Q, C') > g(D_2, Q, C')$, i.e. D_1 and D_2 have the same relative positions in $L(g, Q, C)$ and $L(g, Q, C')$, is proportional to $g(D_1, Q, C) - g(D_2, Q, C)$. Hence, the larger the value of $g(D_1, Q, C) - g(D_2, Q, C)$ is, the more similar their rankings over original and corrupted collections will be. If the values of $g(D_i, Q, C) - g(D_j, Q, C)$ for most documents D_i and D_j in the collection are relatively large, the scoring function is able to effectively differentiate the relevant set of documents from the non-relevant ones. As we discussed in Section2, query Q will be easy. Thus, the degree of the difficulty of a query is positively correlated with the robustness of its ranking over the

[1] Some works on keyword search in databases [1, 13] use conjunctive semantics, where all query keywords must appear in a result.

original and the corrupted versions of the collection. We call this observation the *Ranking Robustness Principle*.

Zhou and Croft [37] 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 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. The 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. Specifically, this approach finds many difficult queries that are classified incorrectly as easy queries using methods based on the similarities between probability distributions. This result is especially important for ranking over structured data. As we explained in Section 2, It is generally hard to define an effective and domain independent similarity function between attributes in databases. Hence, we can use Ranking Robustness Principle as a domain independent proxy metric to measure the degree of the difficulties of queries. Researchers have taken similar approach to evaluate the quality of clustering [16, 24]. The stability of a clustering method has been shown to provide an effective metric for the clustering quality.

4.2 Properties of Hard Queries on Structured Data

The difficulties of answering a query over a database are three-fold:

1. KQI has to find the desired attribute value that is referred by the terms in the query. For example, there are more than one person called *John Ford* in the IMDB data set. Hence, the keyword search method must resolve the desired *John Ford* for the Q_2: *John Ford*.

2. Keyword queries do not specify the attributes for their keywords, unlike queries in languages like SQL. Hence, keyword query interface must disambiguate the keywords in the query. For instance, consider keyword query Q_3: *Ford* over the IMDB data set. There are many directors such as *John Ford*, actors such as *Harrison Ford*, and companies such as *Ford Production* that contain the keywords in Q_3. A keyword query interface must recognize the attribute that is implicitly referred in Q_3.

3. The third challenge is to find the desired entity sets that satisfy the information need behind the query. For instance, IMDB contains both the information about movies and the people involved in making the movies. A user that submits Q_2 may like to see the biographies written about *John Ford* or some movies directed by him. The keyword search system must identify and rank these options.

The aforementioned observations show that we can use the statistical properties of the query terms in the data set to predict the difficulty of the query. Any metric that measures the difficulty of a keyword query must reflect the aforementioned factors that cause difficulty. Intuitively, an idea is to count the number of possible attributes, entities, and entity sets that contain the query terms and use them to predict the difficulty of the query. The larger this value is the more difficult the query will be (we consider this baseline in Section 7.2). However, the type of the distribution of query terms over attributes and entity sets may also impact the degree of 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 keyword query interface can leverage this property and rank books higher than movies when answering query Q_4: *database* over DB_1. However, it will be much harder to decide the desired entity set in DB_2 for Q_4. Hence, a 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. Section 5.1 discusses the role of the structure and content of the database in the corruption process, and presents the robustness computation formula given corrupted database instances. Section 5.2 provides the details of how we generate corrupted instances of the database. Section 5.3 suggests methods to compute the parameters of our model. In Section 5.4 we show real examples of how our method corrupts the database and predicts the difficulty of queries.

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 $DB', DB'', \cdots$ to get ranked lists L and $L', L'', \cdots$, respectively. The less similar L is to $L', L'', \cdots$, 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 $X_a = (X_{a,1}, \cdots, X_{a,V})$, 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:

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

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

Random variable $X_\mathcal{A} = (X_1, \cdots, X_{|\mathcal{A}|})$ 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:

$$f_{X_\mathcal{A}}(\vec{x}) = f_{X_\mathcal{A}}(\vec{x_1}, \cdots, \vec{x_{|\mathcal{A}|}}) = Pr(X_1 = \vec{x_1}, \cdots, X_{|\mathcal{A}|} = \vec{x_{|\mathcal{A}|}}) \quad (2)$$

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 [8]. 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:

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

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 [13, 15, 26], 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 [37], 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 [10]. 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 model attribute value A_i whose attribute is T_t and entity set is S_s is:

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

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:

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

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:

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

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 [14]. 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:

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

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 [15], called *PRSM*, which we explain in more detail in Section 6. Given query Q, PRSM computes the relevance score of entity E based on the weighted linear combination of the relevance scores the attribute values of E.

Q11: Figure 2a 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 Figure 2a) that do not contain any query keywords due to space consideration. Figure 2b illustrates a corrupted version of the entities shown in Figure 2a. The new keyword instances are shown with underline. Note that the ordering changed according to the PRSM ranking. The reason is that PRSM believes that *title* is an important attribute (for attribute weighing in PRSM see Section 6) 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.

```
<movie id= “1025102”>
<title>rome ...</title>
<keyword>ancient-
culture</keyword>
<keyword>ancient-
civilization</keyword>
<keyword>ancient-rome</keyword>
<keyword>christian-era</keyword>
</movie>

<movie id=“1149602”>
<title>Gladiator</title>
<keyword>ancient-rome</keyword>
<character>Rome ...</character>
<person>... Rome/UK)</person>
<trivia>”Rome of the imagination...
</trivia>
<goof>Rome vs. Carthage ...</goof>
<quote>... enters Rome like a ...
Rome ... </quote>
</movie>
```

(a) Original ranking

```
<movie id=“1149602”>
<title> Gladiator rome</title>
<keyword>ancient-rome
rome</keyword>
<character>Rome ...</character>
<person> ... Rome/UK)</person>
<trivia>of the imagination
...</trivia>
<goof>Rome vs. Carthage ...</goof>
<quote>... enters Rome like a ...
Rome ...</quote>
</movie>

<movie id= “1025102”>
<title>rome ...</title>
<keyword>ancient-culture
ancient</keyword>
<keyword>-civilization</keyword>
<keyword>ancient
ancient</keyword>
<keyword>christian-</keyword>
</movie>
```

(b) Corrupted ranking

Figure 2: Original and corrupted results of $Q11$

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 level corruptions typically have much smaller probability of adding a word than the first level, because they have much smaller λ; specifically λ_T is 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 Figure 2b, *rome* is added by the first level of corruption, whereas in the *trivia* attribute *rome* is removed by the first level of corruption.

Example of calculation of $\lambda_{t,j}$ for term t =godfather and attribute T_j =keyword in Equation 8: Assuming that *godfather* occurs in attribute *keyword* 10 times in total, and total number of attribute values under attribute *keyword* is 10,000, $\lambda_{t,j}$ = 10/10000. Then, since γ_T = 0.9, the probability that *godfather* occurs k times in a corrupted *keyword* attribute is $\frac{0.9e^{-\frac{1}{1000}}(\frac{1}{1000})^k}{k!}$.

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*. Figures 3a and 3b 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 in reasonable, 0.0007 for *movie* entity set, but the noise generated in this level alone is not considerable.

```
<movie id="1492260">
<title>The Legend of Mulan (1998)
(V)</title>
<genre>Animation</genre >
<link>Hua Mu Lan (1964)</link>
<link>Hua Mulan cong jun</link>
<link>Mulan (1998)</link>
<link>Mulan (1999)</link>
<link>The Secret of Mulan
(1998)</link>
</movie>

<movie id="1180849">
<title>Hua Mulan (2009)</title>
<character>Hua Hu (Mulan's fa-
ther)</character>
<character>Young Hua Mu-
lan</character>
<character>Hua Mulan</character>
</movie>
```

(a) Original ranking

```
<movie id="1492260">
<title>The Legend of Mulan (1998)
(V) mulan mulan</title>
<genre></genre >
<link>Hua Mu Lan (1964)</link>
<link>Hua Mulan cong jun</link>
<link>Mulan (1998) mulan</link>
<link> (1999)</link>
<link>The Secret of Mulan (1998)
mulan </link>
<movie>

<movie id="1180849">
<title>Hua (2009) hua</title>
<character>Hua Hu (Mulan's fa-
ther)</character>
<character>Young Hua Mulan mulan
mulan hua</character>
<character>Mulan</character>
</movie>
```

(b) Corrupted ranking

Figure 3: Original and corrupted results of $Q9$

6. EFFICIENT STRUCTURED ROBUSTNESS SCORE COMPUTATION

A key requirement for this work to be useful in practice is that the structure robustness score should be computed with very small time overhead compared to the query execution time. In this section we present a suite of optimization and approximation techniques for that.

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 database is so inefficient as to be impractical. Hence, similar to [37], 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 database. In addition to the time savings, our empirical results in Section 7.2 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 7.2.

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 7.2.

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

6.2 Exact 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 database. Some examples of such statistics are the number of occurrences of a query term in all attributes values of the database or total number of attribute values in each attribute and entity set. These global statistics are stored in M (metadata) and I (inverted indexes) in the SR Algorithm pseudocode. SR Algorithm generates the noise in the database 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 database, except to lookup some statistics. Hence, it adds a relatively marginal overhead on the query processing time. 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 7.3, 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 15 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 databases are relatively slow. SR Algorithm has to re-rank the top K entities N times which is time consuming.

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

```
Input: Query $Q$, Top-$K$ list $L$ of $Q$ by ranking function $g$, Metadata $M$, Inverted indexes $I$, corruption iterations $N$.
Output: $SR$ score for $Q$.
 1: $SR \leftarrow 0$; $C \leftarrow \{\}$; //$C$ caches $\lambda_T$, $\lambda_S$ 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 $\lambda_{A,w}$ in $A$; Get $\lambda_{T,w}$, $\lambda_{S,w}$ from $C$;
 9:         IF $\lambda_{A,w} == 0$ and $\lambda_{T,w}$ not in $C$ THEN
10:           Compute $\lambda_{T,w}$ by $I, M$ and cache it to $C$;
11:         IF $\lambda_{T,w} == 0$ and $\lambda_{S,w}$ not in $C$ THEN
12:           Compute $\lambda_{S,w}$ by $I, M$ and cache it to $C$;
13:         IF $\lambda_{A,w}$, $\lambda_{T,w}$ and $\lambda_{S,w}$ all equal to 0 THEN
14:           NEXT;
15:         Compute # of $w$ in $A'$ by Equation 10;
16:         IF # of $w$ varies in $A'$ and $A$ THEN
17:           Update $A'$, $M'$ and entry of $w$ in $I'$;
18:       Add $A'$ to $R'$;
19:     Add $R'$ to $L'$;
20:   Rank $L'$ using $g$, which returns $L$, based on $I'$, $M'$;
21:   $SR$ += $Sim(L, L')$; //$Sim$ computes Spearman correlation
22: RETURN $SR \leftarrow SR/N$; //AVG score over $N$ rounds
```

6.3 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 relatively much smaller than the number of all attribute values in each entity.

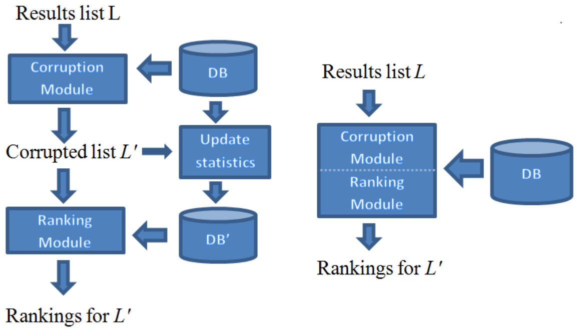
Hence, we can significantly decrease the time spent on corruption if we corrupt only the attribute values that contain query terms. We add a check before Line 7 in SR Algorithm to test if A contains any term in Q. Hence, we skip the loop in Line 7. The second and third levels of corruption (on attributes, 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 database statistics do not change much.

Figure 4a 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 high percentage of the robustness calculation time on the loop that re-ranks the corrupted results (Line 20 in SR Algorithm), by taking into account the updated global statistics. Since the value of K (e.g., 10, 20 or 50) is much smaller than the number of entities in the database, the top K entities constitute a very small portion of the database. Thus, the global statistics largely remain unchanged or change very little. Hence, we can use the global statistics of the original version of the database 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* is illustrated in Figure 4b.



(a) SR Algorithm (b) SGS-Approx

Figure 4: Execution flows of SR Algorithm and SGS-Approx

We use the ranking algorithm proposed in [15], 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 keyword 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:

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

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 database 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 databases that use query term statistics to score entities.

Combination of QAO-Approx and SGS-Approx: QAO-Approx and SGS-Approx improve the performance of robustness calculation by approximating different parts of the corruption and re-ranking process. Hence, we can combine these two algorithms to further improve the performance of the query difficulty predication. Figure 5 depicts the relationship between these two algorithms and SR Algorithm.

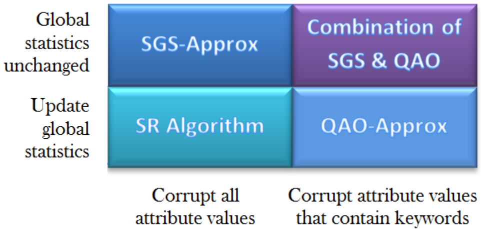

Global statistics unchanged
Update global statistics
SGS-Approx
Combination of SGS & QAO
SR Algorithm
QAO-Approx
Corrupt all attribute values
Corrupt attribute values that contain keywords

Figure 5: Approximation algorithms versus SR Algorithm

7. EXPERIMENTS

7.1 Experiments Setting

Data sets: Table 1 shows the characteristics of two data sets used in our experiments. The INEX data set is from the INEX 2010 Data Centric Track [31]. It is a subset of the IMDB dataset. 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*, *year*, *actor*, *director* etc. The *person* entity set contains attributes like *name*, *nickname*, *movie*. The SemSearch data set is a subset of the data set used in Semantic Search 2010 challenge [29]. 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.

Table 1: 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

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 relevant 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 data set.

28 query topics are provided in the INEX 2010 Data Centric Track. However, no relevant judgments are provided for two of them (Q3 and Q13). Thus, we use the 26 queries that have relevance judgment in our experiments on INEX. Some query topics contain characters "+" and "-" to indicate the conjunctive and exclusive conditions. In our experiments, we don't use these conditions and remove the keywords after character "-". Generally, keyword query interfaces over databases return candidate answers that contain all terms in the query [4, 13, 9, 18, 26]. 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 $|Q|$ terms in query Q. If they are not sufficient, we retrieve the entities that contain at least $|Q| - 1$ keywords, and so on until we get 1000 candidate answers for each query.

Effectiveness Metrics: We have used the widely used effectiveness metrics Average Precision and Mean Average Precision (MAP) to measure the quality of rankings delivered by ranking algorithms for a query and a set of queries over a database, respectively [20].

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 [15] and IR-Style [13]. Many other algorithms are extensions of these two methods (e.g., [19, 17, 6]).

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 0.1 on the INEX dataset and 0.2 for the SemSearch dataset.

IR-Style: We use a variation of the ranking model proposed in [13] for relational data model, referred as IR-Style ranking. Given a keyword query, IR-Style returns a minimal join tree that connects the tuples from different tables in the database 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 [13] 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 [36] 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.

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

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.

7.2 Quality results

In this section, we evaluate the effectiveness of the query quality prediction model computed using SR Algorithm. We use Pearson'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 [20]. Hence, it is reasonable to focus on small values of K for query performance prediction. 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). We have achieved the best prediction quality using K=20 for both datasets with different combination of γ_A, γ_T, and γ_S which we will introduce later. Hence, relatively smaller values for K provide not only efficient prediction (short execution time), but also reasonable effectiveness (prediction quality). Since the low ranked answers are generally not relevant, their scores are considerably smaller than the top ranked answers. As discussed in Section 4.1, these low ranked answers have a low chance of replacing the top ranked answers in the ranked list for the corrupted versions of the data. Hence, if the value of K is not too small, we achieve a reasonable prediction quality.

Impact of the Values for γ_A, γ_T, and γ_S: We denote the coefficients combination in Equation 10 as (γ_A, γ_T, γ_S) for brevity. 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. We computed the SR score for γ_T and γ_S from 0 to 3 (we only show results up to 1 below because for more than 1 the correlation was weaker) with step 0.1 for different values of K. Figures 6 and 7 show the correlation between average precision and SR score for various (γ_A, γ_T, γ_S) with K=20, for INEX and SemSearch, respectively. The highest correlation is achieved by (1, 0.1, 0.6) on SemSearch, and (1, 0.9, 0.8) on INEX. On both datasets the second and third levels of corruption improve the correlation over 15% compared to (1, 0, 0).

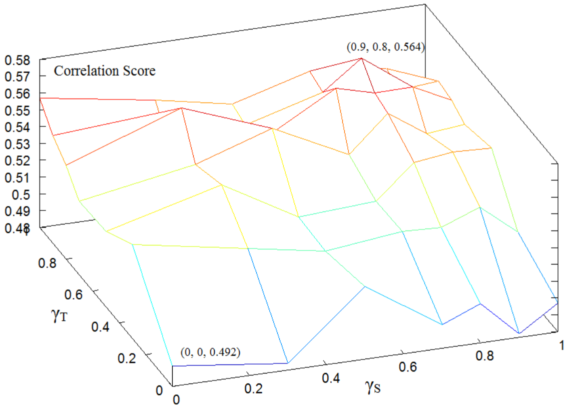


Figure 6: Effect of (γ_A, γ_T, γ_S) on correlation score for the INEX dataset using PRMS and K=20.

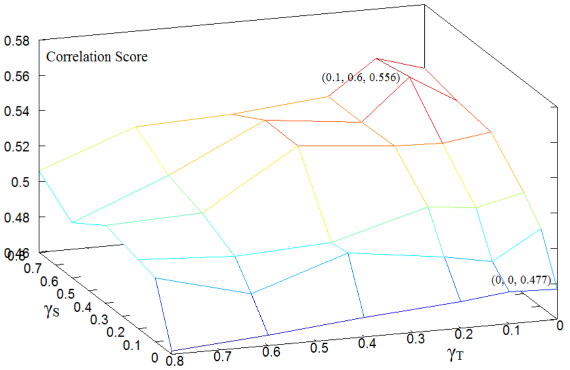


Figure 7: Effect of (γ_A, γ_T, γ_S) on correlation score for the SemSearch dataset using PRMS and K=20.

Figures 8 and 9 depict the plot of average precision and SR score for the queries in INEX and SemSearch, respectively. In Figure 8, we see that $Q9$ is easy (has high average precision) and $Q11$ is relatively hard, as discussed above. As shown in Figure 9, 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. Figure 9 shows that the SR scores of these queries are very small, which confirms our model.

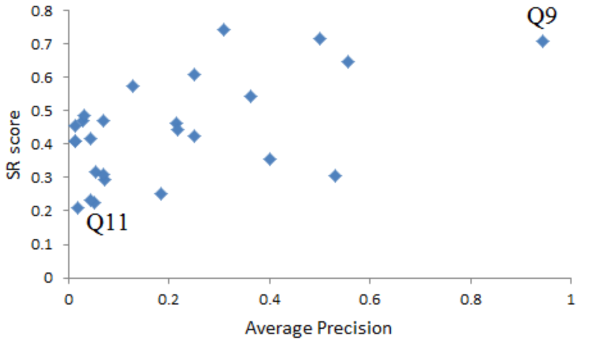


Figure 8: Average precision versus SR score for queries on INEX using PRMS, K=20, and (γ_A, γ_T, γ_S) = (1, 0.9, 0.8).

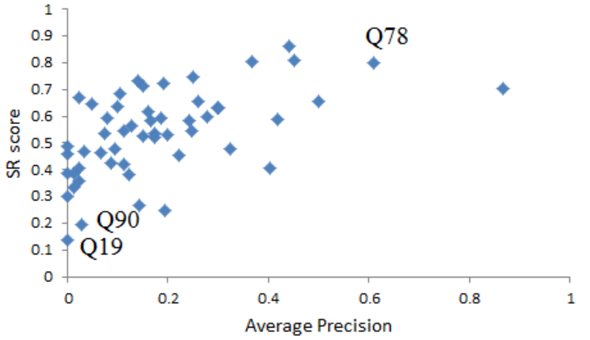


Figure 9: Average precision versus SR score for queries on SemSearch using PRMS, K=20, and (γ_A, γ_T, γ_S) = (1, 0.1, 0.6).

Baseline Prediction Methods

Unstructured Robustness Method (URM): We use the idea in [37] as the baseline query performance prediction algorithm. Our goal in this experiment is to find how accurately URM can predict the effectiveness of queries over a given database DB. Since the work in [37] is for text documents, we concatenate the XML elements and tags of each entity into a text document. We 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 terms (see calculation of μ_j in Section 6 for details). Hence, PRMS becomes a language model retrieval method [20].

Prevalence of Query Keywords: As we argued in Section 4.2, if the query keywords appear in many attribute values, attributes or entity sets, it is harder for a ranking algorithm to locate the desired entities. Given keyword query Q, we compute the average number of attributes ($AvgA(Q)$), average number of entity sets ($AvgES(Q)$), and the average number of entities ($AvgE(Q)$) where each keyword in Q occurs. We consider each of the three as an individual baseline, and also consider their combination. We combine them by multiplying the three (to avoid normalization issues that summation would have); we refer to the product as $AS(Q)$. Intuitively, if these metrics have higher value, the query Q should be harder, and hence the average precision of Q should be lower, Thus we use the inverse of these values, denoted as $iAvgA(Q)$, $iAvgES(Q)$ and $iAvgE(Q)$ and $iAS(Q)$ respectively.

Table 2 shows the prediction accuracy (correlation between average precision and each measure) for SR Algorithm, URM, $iAvgA(Q)$, $iAvgES(Q)$ and $iAvgE(Q)$ and $iAS(Q)$ for different values of K. We use (γ_A, γ_T, γ_S)= (1, 0.1, 0.6) on SemSearch and (1, 0.9, 0.8) on INEX for SR Algorithm. The correlation values for SR Algorithm are significant higher than the correlation values of URM on both datasets. This shows that our prediction model is more effective than URM over databases. The other baselines provide better prediction quality than URM over the INEX dataset. This indicates that the main cause of the difficulties for the queries over the INEX dataset is to find their desired attributes. Since all query keywords in our query workloads occur in both entity sets in the IMDB dataset, we put n/a on the correlation score between $iAvgES(Q)$ and average precision.

IR-style ranking algorithm: The best value of MAP for the IR-Style ranking algorithm over INEX is 0.105 for K=20, which is very low. Note that we tried both Equation 12 as well as the vector space model originally used in [13]. 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 3 shows Pearson's correlation of SR score with the average precision for different values of K, for N=250 and (γ_A, γ_T, γ_S) = (1, 0.1, 0.6). Figure 10 plots SR score against the average precision when K=20. On SemSearch, we also tried IR-Style ranking algorithm using vector space model ranking, but it achieves very low MAP (0.08) so we do not report results.

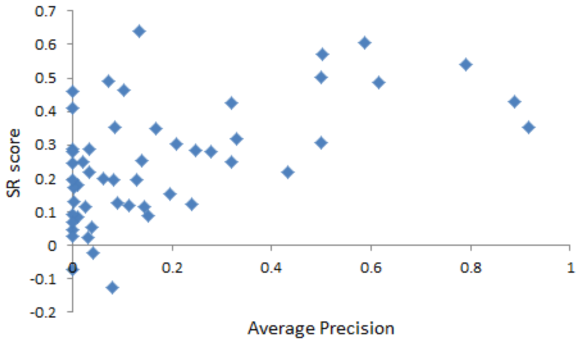


Figure 10: Average precision versus SR score using IR-Style over SemSearch with K=20 and (γ_A, γ_T, γ_S) = (1, 0.1, 0.6).

Table 2: Pearson's correlation of average precision against: SR score computed by SR Algorithm, URM, iAvgA, iAvgES, iAvgE and iAS.

K	10						20						50					
Method	SR	URM	iAvgA	iAvgES	iAvgE	iAS	SR	URM	iAvgA	iAvgES	iAvgE	iAS	SR	URM	iAvgA	iAvgES	iAvgE	iAS
INEX	0.471	0.247	0.299	n/a	0.111	0.143	0.556	0.311	0.370	n/a	0.255	0.292	0.398	0.274	0.431	n/a	0.354	0.399
SemSearch	0.486	0.093	0.066	0.052	0.040	-0.043	0.564	0.177	0.082	0.068	0.056	-0.046	0.449	0.151	0.0915	0.078	0.064	-0.038

Table 3: Effect of K on Pearson's correlation of average precision and SR score using IR-Style ranking on SemSearch

K	10	20	50
Correlation score	0.565	0.544	0.539

Table 4: Average query processing time and average robustness computation for K=20, N=250 on SemSearch and N=300 on INEX.

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

7.3 Performance Study

In this section we study the efficiency of our SR score computation algorithms. We use PRMS as the ranking algorithm for our experiments. We first report the performance results for SR Algorithm, and then for the approximation 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 PRSM for the queries in our query workloads. These times are presented in Table 4 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 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. This does not happen for SemSearch. From Table 1, we see that SemSearch contains far more entity sets and attributes than INEX.

QAO-Approx: Figures 11a and 12a 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 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 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: Figures 11b and 12b depict the results of applying SGS-Approx on INEX and SemSearch, respectively. Since re-ranking is done on-the-fly during corruption, SR-time is reported as corruption time only. As shown in Figure 11b, 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.

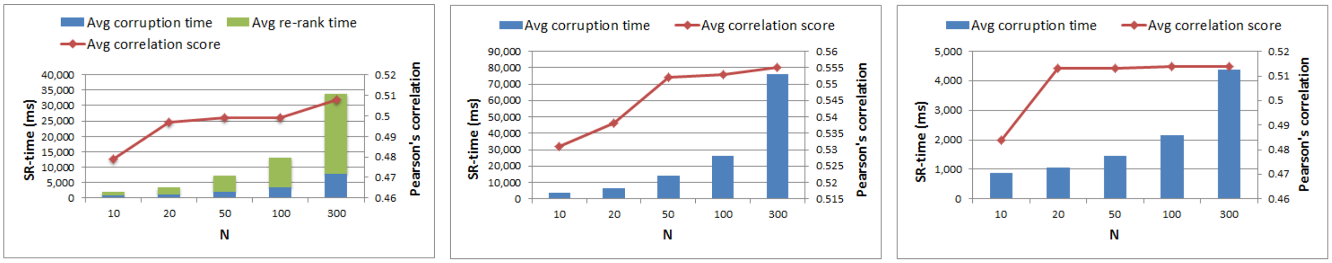
Combination of QAO-Approx and SGS-Approx: As noted in Section 6, we can combine QAO-Approx and SGS-Approx algorithms to achieve better performance. Figures 11c and 12c present the results of the combined algorithm for INEX and SemSearch databases, respectively. Since we use SGS-Approx, the SR-time consists 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, we observe that the combined algorithm delivers the best balance of improvement in efficiency and decrease in effectiveness for both datasets. On both datasets, the combined algorithm can achieve high prediction accuracy (correlation score about 0.5) with SR-time around 1 second. On INEX when the value of N is set to 20, the correlation is 0.513 and the time is decreased to about 1 second using the combined algorithm. For SR Algorithm on INEX, when N decreases to 10, correlation score is 0.537, but SR-time is over 9.8 seconds which is not ideal. On SemSearch, if we use the combined algorithm, the correlation score is 0.495 and SR-time is about 1.1 seconds when N=50. However, to achieve the similar efficiency, SGS-Approx needs to decrease N to 10, with SR-time is 1.2 seconds and correlation is 0.49. And SR Algorithm spends over 2.2 and 6 seconds of SR-time when N is set to 10 and 50 respectively. Thus, the combined algorithm is the best choice to predict the difficulties of queries efficiently and effectively.

8. CONCLUSION AND FUTURE WORK

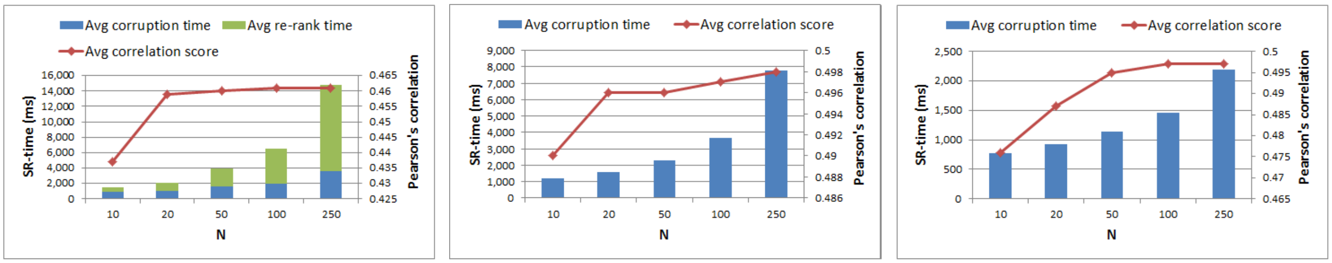
In this paper, we introduced the novel problem of predicting the effectiveness of keyword queries over databases. We showed that the current prediction methods for queries over unstructured data sources cannot be effectively used to solve this problem. We set forth a principled framework to measure the degree of the difficulty of a keyword query over a database, 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.

An interesting future work is to extend this framework to estimate the query difficulty on other top K ranking problems in databases such as ranking underspecified SQL statements. Further, we plan to experiment with ranking functions that may not fall under the two function classes we used in this paper.



(a) QAO-Approx (b) SGS-Approx (c) Combination of QAO and SGS

Figure 11: Approximations on INEX.



(a) QAO-Approx (b) SGS-Approx (c) Combination of QAO and SGS

Figure 12: Approximations on SemSearch.

9. REFERENCES

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

[2] A.Singhal, G.Salton, and C. Buckley. Length Normalization in Degraded Text Collections. In *Symposium on Document Analysis and Information Retrieval*, pages 149–162, 1996.

[3] Z. Bao, T. W. Ling, B. Chen, and J. Lu. Effective XML Keyword Search with Relevance Oriented Ranking. In *ICDE*, pages 517–528, 2009.

[4] G. Bhalotoa, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in databases using BANKS. In *ICDE*, pages 431–440, 2002.

[5] K. Collins-Thompson and P. N. Bennett. Predicting Query Performance via Classification. In *ECIR*, pages 140–152, 2010.

[6] E. Demidova, P. Fankhauser, X. Zhou, and W. Nejdl. DivQ: Diversification for Keyword Search over Structured Databases. In *SIGIR*, pages 331–338, 2010.

[7] V. Ganti, Y. He, and D. Xin. Keyword++: A Framework to Improve Keyword Search Over Entity Databases. *PVLDB*, 3:711–722, 2010.

[8] J. Gibbons and S.Chakraborty. *Nonparametric Statistical Inference*. Marcel Dekker, New York, 1992.

[9] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. In *SIGMOD*, pages 16–27, 2003.

[10] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2011.

[11] C. Hauff, V. Murdock, and R. Baeza-Yates. Improved Query Difficulty Prediction for the Web. In *CIKM*, 2008.

[12] B. He and I. Ounis. Query performance prediction. *Inf. Syst.*, 31:585–594, November 2006.

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

[14] S. M. Katz. Estimation of Probabilistic from Sparse Data for the Language Model Component of a Speech Recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401, 1987.

[15] J. Kim, X. Xue, and B. Croft. A Probabilistic Retrieval Model for Semistructured Data. In *ECIR*, pages 228–239, 2009.

[16] T. Lange, V. Roth, M. Braun, and J. Buhmann. Stability-Based Validation of Clustering Solutions. *Neural Computation*, 16(6):1299–1323, 2004.

[17] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective Keyword Search in Relational Databases. In *SIGMOD*, pages 563–574, 2006.

[18] Z. Liu and Y. Chen. Reasoning and Identifying Relevant Matches for XML Keyword Search. In *VLDB*, volume 1, pages 921–932, 2008.

[19] Y. Luo, X. Lin, W. Wang, and X. Zhou. SPARK: Top-k Keyword Query in Relational Databases. In *SIGMOD*, pages 115–126, 2007.

[20] C. Manning, P. Raghavan, and H. Schutze. *An Introduction to Information Retrieval*. Cambridge University Press, 2008.

[21] E. Mittendorf and P. Schauble. Measuring the Effects of Data Corruption on Information Retrieval. In *SDAIR*, pages 179–189, 1996.

[22] A. Nandi and H. V. Jagadish. Assisted Querying Using Instant-Response Interfaces. In *SIGMOD*, pages 1156–1158, 2007.

[23] N. Sarkas, S. Paparizos, and P. Tsaparas. Structured

Annotations of Web Queries. In *SIGMOD*, pages 771–782, 2010.

[24] O. Shamir and N. Tishby. Cluster Stability for Finite Samples. In *NIPS*, 2007.

[25] A. Shtok, O. Kurland, and D. Carmel. Using Statistical Decision Theory and Relevance Models for Query-Performance Prediction. In *SIGIR*, pages 259–266, 2010.

[26] A. Termehchy and M. Winslett. Using Structural Information in XML Keyword Search Effectively. *TODS*, 36(1):4:1–4:39, 2011.

[27] A. Termehchy, M. Winslett, and Y. Chodpathumwan. How Schema Independent Are Schema Free Query Interfaces? In *ICDE*, pages 649–660, 2011.

[28] S. C. Townsend, Y. Zhou, and B. Croft. Predicting Query Performance. In *SIGIR*, pages 299–306, 2002.

[29] T. Tran, P. Mika, H. Wang, and M. Grobelnik. Semsearch'10: the 3th semantic search workshop. In *WWW*, 2010.

[30] T. Tran, P. Mika, H. Wang, and M. Grobelnik. Semsearch'11: the 4th semantic search workshop. In *WWW*, 2011.

[31] A. Trotman and Q. Wang. Overview of the INEX 2010 Data Centric Track. In *Comparative Evaluation of Focused Retrieval*, volume 6932 of *Lecture Notes in Computer Science*, pages 171–181. Springer Berlin / Heidelberg, 2011.

[32] E. M. Voorhees. Overview of the TREC 2003 Robust Retrieval Track. In *TREC*, 2004.

[33] J. Wang and J. Zhu. Portfolio Theory of Information Retrieval. In *SIGIR*, pages 115–122, 2009.

[34] X. Xing, Y. Zhang, and M. Han. Query Difficulty Prediction for Contextual Image Retrieval. In *Advances in Information Retrieval*, volume 5993, pages 581–585. 2010.

[35] 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 *SIGIR*, pages 512–519, 2005.

[36] C. Zhai and J. Lafferty. A Study of Smoothing Methods for Language Models Applied to Ad Hoc Information Retrieval. In *SIGIR*, pages 334–342, 2001.

[37] Y. Zhou and B. Croft. Ranking Robustness: A Novel Framework to Predict Query Performance. In *CIKM*, pages 567–574, 2006.