Predicting the Effectiveness of Keyword Queries on Databases

Shiwen Cheng
Department of Computer Science & Engineering
University of California at Riverside
Riverside, CA 92507. USA
schen064@cs.ucr.edu

Arash Termehchy
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801. USA
termehch@uiuc.edu

Vagelis Hristidis
Department of Computer Science & Engineering
University of California at Riverside
Riverside, CA 92507. USA
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 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. Further, our prediction algorithms incur minimal time overhead.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process*

General Terms

Measurement, Performance

Keywords

Query Performance, Keyword Query, (Semi-)Structured Data, Database

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 [2, 10, 5, 19, 16, 12]. 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 [2, 10]. 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 [13]. 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 [22] 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 [21], 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. 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

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.

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 [20]. These queries are usually either under-specified, such as query *carolina* in Table 1, or over-specified, such as query *Movies Klaus Kinski actor good rating* in Table 1.

Table 1: List of difficult queries from both benchmarks.

INEX	SemSearch
ancient rome era	austin texas
Movies Klaus Kinski actor good rating	carolina
true story drugs addiction	earl may
	lynchburg virginia
	san antonio

It is important for a KQI to recognize such queries and warn the user or employ alternative techniques like query reformulation or query suggestions [15]. It may also use techniques such as diversification of its returned ranked list [4]. 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 (such as CPU cycle) and the time of users. 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). As a specific example of how knowledge of the query difficulty may be leveraged, consider Figure 1 which shows the ranking results for query *ancient Rome era* by one of our implemented ranking algorithms (details in Section 7). Our algorithms determine that this is a hard (ambiguous) query, which guides the system to generate query reformulation suggestions. Note that the generation of query reformulation technique is beyond our scope in this paper.

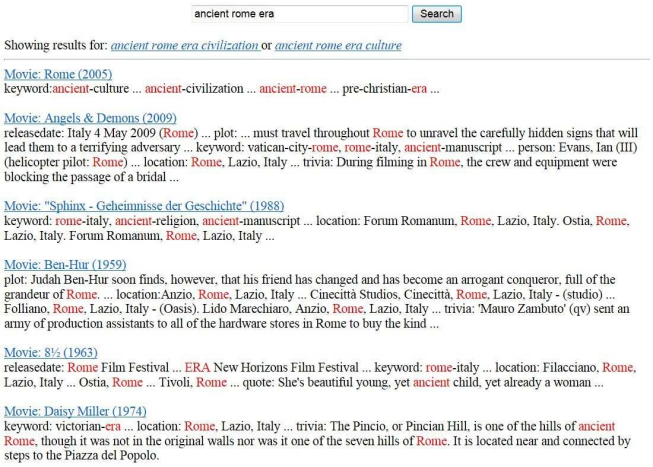


Figure 1: Results for hard query *ancient Rome era* by our system with query suggestions returned.

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 [20, 24]. 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 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.
- 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.
- 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.
- 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 describes the algorithms to compute the SR score. Section 7 contains the experimental results, and Section 8 concludes the paper.

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 [20, 24, 9, 3]. 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

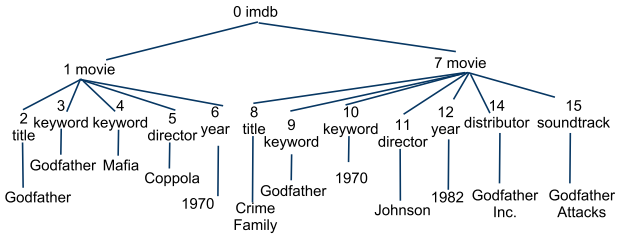


Figure 2: IMDB database fragment

in the query and number of documents that contain at least one query term [9]. 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 [20].

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 [20, 3]. 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 [20]. 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 returned 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 [20].

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 [8]. For instance, different attributes may have different impacts on the degree of the similarity between entities. Assume movies A and B share some terms in their *genre* attributes, which explain the subjects of the movies, in IMDB database. Also, let movies A and C share the same number of terms in their *distributor* attributes, which describe the distribution company of the movies. Given other attributes of A, B, and C do not contain any common term, movies A and B are more likely to be about the same subject and satisfy the same information need than movies A and C. Our empirical results in Section 7 confirms this argument and shows that the straightforward extension of clarity score method predicts difficulties of the queries over databases poorly.

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 [3]. 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 [23]. 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 2 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 2. 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* [17, 6] and *data-centric XML retrieval* [22], 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 [18]. 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 robust-

[1]Some works on keyword search in databases [10] use conjunctive semantics, where all query keywords must appear in a result.

ness 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* [24], 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 [14]. 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 [24] 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. 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 more diverse the candidate answers for the query are 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. If a query matches different attributes in its candidate answers, it will have a more diverse set of potential answers in database. 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. The kind of difficulty introduced by this type of diversity is different from the kind of difficulty created by having a large number of matched entities. Assume that Q_4 and Q_5 have almost equal number of answers. Some of the movies whose titles match Q_5 are related, e.g., there are three documentaries in IMDB whose titles match *taxi driver*, which are about making the well-known movie *taxi driver* directed by *Martin Scorsese*. They may partially or fully satisfy the information need behind Q_5. However, the candidate answers whose *title* attribute match Q_4 and the candidate answers whose *distributor* attribute match Q_4 are not generally related.

3. Each entity set contains the information about a different type of entities. Hence, if a query matches entities from more entity sets, it will have more diverse set of candidate answers. 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. 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_6 has only one possible desired entity set. This kind of diversity poses a new type of difficulty. since the candidate answers of Q_6 are about romantic comedy movies about divorce, they have some similarities to each other. However, movies about divorce and people who get divorced cannot both satisfy information need of query Q_6. Given Q_6 and Q_7 have almost the same number of candidate answers and matching attributes, it is likely that more candidate answers of Q_6 are relevant to its users' information need than the candidate answers to Q_7.

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. One 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 have shown empirically in Section 7.2 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. 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:

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

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 [7]. 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 [10, 12], 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 [24], 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 [11]. 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 $\gamma_A + \gamma_T + \gamma_S = 1$.

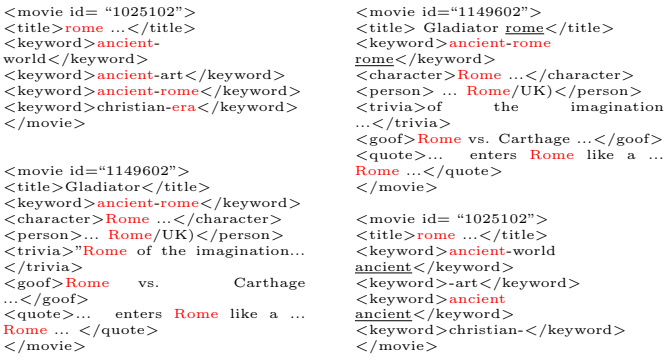
5.4 Examples

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 [12], 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.

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: Figure 3a 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 3a) that do not contain any query keywords due to space consideration. Figure 3b illustrates a corrupted version of the entities shown in Figure 3a. The new keyword instances are underlined. Note that the ordering changed according to the PRSM ranking. The reason is that PRSM believes that *title* is an important attribute for *rome* (for attribute weighing in PRSM see Section 7.1) 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.



(a) Original ranking (b) Corrupted ranking

Figure 3: 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.

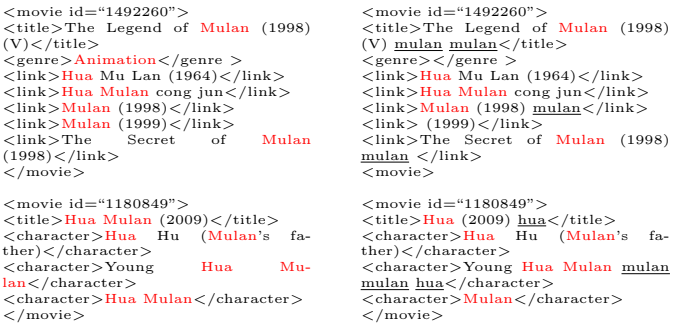
The second and third levels corruptions typically have much smaller probability of adding a word than the first level, because they have much smaller λ; specifically λ_T is the average frequency of the term in attribute T. However, in hard queries like $Q11$, the query terms are frequent in the database, 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 3b, *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*. Figures 4a and 4b 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.



(a) Original ranking (b) Corrupted ranking

Figure 4: Original and corrupted results of $Q9$

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 [24], 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 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.

Limiting the values of K or N are simple ways to decrease the execution time, without much accuracy degradation, as we show in Section 7.

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

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 ← 0; C ← {}; //C caches λ_T, λ_S for keywords in Q
2: FOR i = 1 → N DO
3:    I' ← I; M' ← M; L' ← 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' ← A; //Corrupted versions of A
7:          FOR each keywords w in Q DO
8:             Compute # of w in A' by Equation 10; //If λ_{T,w}, λ_{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 ← SR/N; //AVG score over N rounds
```

The 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.

In order to improve the efficiency of our method, we corrupt only the attribute values that contain at least query keywords. We also use the statistics of original database to re-rank the corrupted results. Moreover, we further limit the value of N. Our empirical results in Section 7 show that these will largely improve the efficiency of our model without much degradation of the prediction quality.

7. EXPERIMENTS

7.1 Experiments 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 [22] discussed in Section 1.The SemSearch data set is a subset of the data set used in Semantic Search 2010 challenge [21]. 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". We have removed the relevance judgment information for the subjects that do not reside in these 20 files.

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

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. 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. Some INEX queries 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 "-". Generally, KQIs over databases return candidate answers that contain all terms in the query [2, 10, 18]. 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 for each query.

Ranking Algorithm: We have evaluated our query performance prediction model using a representative ranking algorithm called PRMS [12]. Many ranking methods for keyword queries over structured data follow a similar heuristic as this algorithm (e.g., [1, 4]). 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 keyword 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:

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

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. 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.

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 index the XML files and implement all algorithms in Java.

7.2 Prediction Quality

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 [13]. Hence, it is reasonable to focus on small values of K for query performance prediction. We conduct our experiments on K=10, 20. Both 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.

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) by 5-fold cross validation. 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 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. Table 3 shows the training of γ_T and γ_S, and the correlation between average precision and SR score on testing sets. It shows that for different training sets, the values of (γ_A, γ_T, γ_S) for the best correlation score are quite stable. In the following results, we set (γ_A, γ_T, γ_S) to (1, 0.9, 0.8) on INEX and (1, 0.1, 0.6) on SemSearch.

Table 3: Training and testing of (γ_A, γ_T, γ_S) under K=20.

	INEX		SemSearch	
training set	(γ_A, γ_T, γ_S)	correlation	(γ_A, γ_T, γ_S)	correlation
1	(1, 0.9, 0.8)	0.689	(1, 0.1, 0.6)	0.744
2	(1, 0.9, 0.8)	0.777	(1, 0.1, 0.6)	0.659
3	(1, 0.9, 0.8)	0.695	(1, 0.1, 0.8)	0.596
4	(1, 0.9, 0.8)	0.799	(1, 0.1, 0.6)	0.702
5	(1, 0.8, 0.3)	0.540	(1, 0.1, 0.6)	0.597

Figures 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 Figure 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 Figure 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. Figure 6 shows that the SR scores of these queries are very small, which confirms our model.

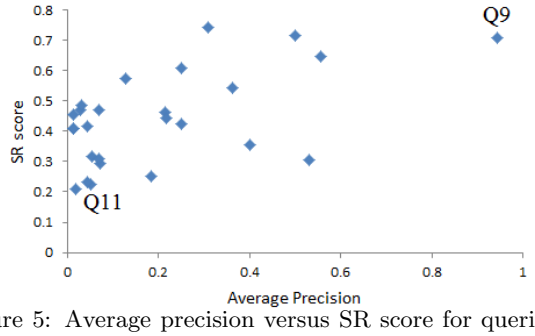


Figure 5: Average precision versus SR score for queries on INEX using PRMS, K=20.

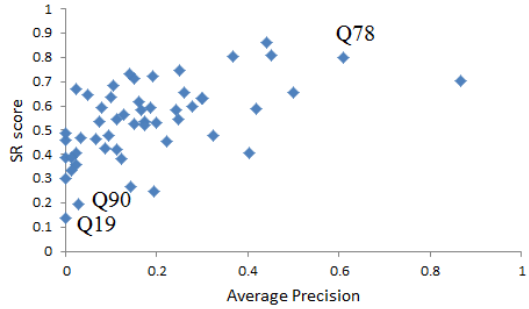


Figure 6: Average precision versus SR score for queries on SemSearch using PRMS, K=20.

Table 4: Correlation of average precision and difficulty measurements for different methods for K=10.

Method	SR	URM	CR	iAA	iAES	iAE	iAS
INEX	0.471	0.247	0.379	0.299	n/a	0.111	0.143
SemSearch	0.486	0.093	0.091	0.066	0.052	0.040	-0.043

Table 5: Correlation of average precision and difficulty measurements for different methods for K=20.

Method	SR	URM	CR	iAA	iAES	iAE	iAS
INEX	0.556	0.311	0.391	0.370	n/a	0.255	0.292
SemSearch	0.564	0.177	0.09	0.082	0.068	0.056	-0.046

Baseline Prediction Methods

Clarity score (CR) [20] and Unstructured Robustness Method

(URM) [24] are two popular query difficulty prediction techniques over text documents. The prediction quality (i.e. the correlation between average precisions of queries and the measurement) of clarity score and URM are 0.21 - 0.51 and 0.30 - 0.61, respectively [24]. We use these methods as well as prevalence of query keywords as baseline query difficulty prediction algorithms in databases.

URM and CR: Our goal in this experiment is to find how accurately URM can predict the effectiveness of queries over a database. 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 [13]. Similar to URM, we implement CR by treating each entity in database as a text document. We have used similar parameters as [20, 24] to compute the CR for queries over our data sets.

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 measurements. We also multiply these three measurements (to avoid normalization issues that summation would have) and create another baseline measurement, denoted as $AS(Q)$. Intuitively, if these measurements 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: Table 4 and 5 shows the prediction accuracy (correlation between average precision and each measurement) for SR, URM, CR, iAA(Q), iAES(Q), iAE(Q), and iAS(Q) methods over both datasets for K=10 and 20, respectively. These results are based on all queries in the query workloads without distinguishing between training and testing sets as in Table 3. The *n/a* value appears in the table because all query keywords in our query workloads occur in both entity sets in the INEX dataset. The correlation values for SR Algorithm are significantly higher than the correlation values of URM and CR on both datasets. This shows that our prediction model is more effective than URM and CR over databases. Measurement iAA provides a more accurate prediction than all other methods over INEX. 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) measurements over both data sets. Hence, SR effectively considers all causes of the difficulties for queries over databases.

7.3 Efficiency

As mentioned in Section 6, we propose techniques to improve the efficiency of our prediction model. Using these techniques, the time spent on the calculation of SR score is 1 and 1.1 second with correlation score of over 0.51 and 0.495 for INEX and SemSearch, respectively.

8. CONCLUSION

We introduced the novel problem of predicting the effectiveness of keyword queries over DBs. 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. Our extensive experiments show that the algorithms predict the difficulty of a query with relatively low errors and negligible time overhead.

9. ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation grants IIS-1216032 and IIS-1216007.

10. REFERENCES

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

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

[3] K. Collins-Thompson and P. N. Bennett. Predicting Query Performance via Classification. In *ECIR*, 2010.

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

[5] R. Fagin, B. Kimelfeld, Y. Li, S. Raghavan, and S. Vaithyanathan. Understanding Queries in a Search Database System. In *PODS*, 2010.

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

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

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

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

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

[11] S. M. Katz. Estimation of Probabilistic from Sparse Data for the Language Model Component of a Speech Recognizer. *IEEE Trans. Signal Process.*, 35(3):400–401, 1987.

[12] J. Kim, X. Xue, and B. Croft. A Probabilistic Retrieval Model for Semistructured Data. In *ECIR*, 2009.

[13] C. Manning, P. Raghavan, and H. Schutze. *An Introduction to Information Retrieval*. 2008.

[14] E. Mittendorf and P. Schauble. Measuring the Effects of Data Corruption on Information Retrieval. In *SDAIR*, 1996.

[15] A. Nandi and H. V. Jagadish. Assisted Querying Using Instant-Response Interfaces. In *SIGMOD*, 2007.

[16] L. Qin, J. X. Yu, and L. Chang. Keyword Search in Databases: The Power of RDBMS. In *SIGMOD*, 2009.

[17] N. Sarkas, S. Paparizos, and P. Tsaparas. Structured Annotations of Web Queries. In *SIGMOD*, 2010.

[18] A. Termehchy, M. Winslett, and Y. Chodpathumwan. How Schema Independent Are Schema Free Query Interfaces? In *ICDE*, 2011.

[19] M. Theobald, R. Schenkel, and G. Weikum. The TopX DB&IR Engine. In *SIGMOD*, 2007.

[20] S. C. Townsend, Y. Zhou, and B. Croft. Predicting Query Performance. In *SIGIR*, 2002.

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

[22] 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*. 2011.

[23] E. Yom-Tov, S. Fine, D. Carmel, and A. Darlow. Learning to Estimate Query Difficulty. In *SIGIR*, 2005.

[24] Y. Zhou and B. Croft. Ranking Robustness: A Novel Framework to Predict Query Performance. In *CIKM*, 2006.