Efficient Query Construction for Large Scale Data

Elena Demidova
L3S Research Center
Leibniz Universität Hannover
Hannover, Germany
demidova@L3S.de

Xuan Zhou*
DEKE Lab, MOE
Renmin University of China
Beijing, China
zhou.xuan@outlook.com

Wolfgang Nejdl
L3S Research Center
Leibniz Universität Hannover
Hannover, Germany
nejdl@L3S.de

ABSTRACT

In recent years, a number of open databases have emerged on the Web, providing Web users with platforms to collaboratively create structured information. As these databases are intended to accommodate heterogeneous information and knowledge, they usually comprise a very large schema and billions of instances. Browsing and searching data on such a scale is not an easy task for a Web user. In this context, interactive query construction offers an intuitive interface for novice users to retrieve information from databases neither requiring any knowledge of structured query languages, nor any prior knowledge of the database schema. However, the existing mechanisms do not scale well on large scale datasets. This paper presents a set of techniques to boost the scalability of interactive query construction, from the perspective of both, user interaction cost and performance. We connect an abstract ontology layer to the database schema to shorten the process of user-computer interaction. We also introduce a search mechanism to enable efficient exploration of query interpretation spaces over large scale data. Extensive experiments show that our approach scales well on Freebase - an open database containing more than 7,000 relational tables in more than 100 domains.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Query formulation

Keywords

Query construction, Freebase, Ontology

1. INTRODUCTION

With the prevalence of Web 2.0, a number of open databases have emerged on the Web, attempting to provide a platform for users to collaboratively create and maintain structured information. A typical example is Freebase[1], which currently contains more than 22 million entities and 350 million facts from more than 100 domains, organized in 7,500 tables. Other examples include DBpedia[2], WikiTaxonomy[3], and Probase[4], whose sizes have already reached the magnitude of several GBs. Databases of this kind are intended to accommodate heterogeneous information and knowledge. It is natural that each of these datasets contains a very large schema and a large volume of data. For a typical Web user, information seeking over such large and heterogeneous database is a challenge.

The technology of interactive query construction [7] enables novice users to interactively create structured queries and retrieve desired information from a database without knowing any structured query language or studying the database schema a-priori. The existing approaches of interactive query construction work well for small or medium sized databases of a particular domain, such as IMDB[5] and Lyrics [16], which contain around 20 tables [7]. However, we found that existing approaches fail to scale on a heterogeneous Freebase database composed of several thousand tables. To this extent, the FreeQ system presented in this paper enables us to scale interactive query construction over a very large database.

The interface of interactive query construction combines the usability of keyword queries with the expressiveness of structured queries. It enables a user to start with a keyword query and refine it into a structured query by interacting with the system. Through interaction, the user can provide additional information to disambiguate the semantics of the keyword query, and finally determine the structured expression reflecting her informational need. The resulting structured queries offer enhanced expressiveness to retrieve results with complex semantics, including collective results, e.g. *"All films starring Tom Hanks"*, or results involving more than one entity, e.g. *"The role of Tom Hanks in the film The Terminal"*. In this way, interactive query construction opens the world of structured queries to unskilled users, who are not familiar with structured query languages, without actually requiring them to learn such query language. This interface is also a useful tool for expert users, who want to explore data organized in an unfamiliar and complex database schema.

Interactive query construction is especially useful for seeking information on large scale, where keyword queries become increasingly ambiguous. For instance, in Freebase, the phrase *"Jack London"* occurs in more than sixty attributes and can be matched with the entities of types *author*, *film*, *olympic athlete*, *tourist attraction*, *astronomical discovery*, and others. As a result, there can be a large number of plausible answers to the keyword query *"Jack London"*, such as author Jack London, asteroid "2625 Jack London", a British

*Corresponding author: zhou.xuan@outlook.com
[1]www.freebase.com

[2]www.dbpedia.org
[3]http://www.h-its.org/english/research/nlp/index.php
[4]http://research.microsoft.com/en-us/projects/probase
[5]www.imdb.com

athlete John Edward “Jack” London, and the Jack London District in Oakland, California. To find the desired information through a keyword search interface, a user may have to scan through a long list of search results. In contrast, an interactive query construction interface suggests interaction options, such as *“Jack London is a book author”* for the user to clarify her intent. By clicking the correct options, the user helps the system to form structured queries to precisely retrieve the desired information.

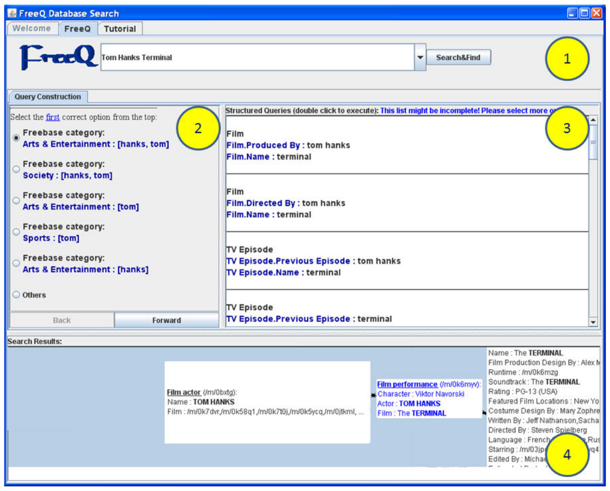
There are two main reasons why existing approaches fail to scale on heterogeneous databases composed of several thousand tables. First, when the database schema is very big, the interaction options generated by the existing schemes are usually not informative enough. As a result, a user may have to go through a laborious interaction procedure to construct the desired query. For example, as the phrase *“Jack London”* appears in more than sixty Freebase attributes, this phrase can be interpreted into more than sixty meanings. Using the existing interaction schemes, in the worst case, the user may have to respond to each of the sixty interpretations to finally clarify her intent. For a more complex keyword query, the procedure of interaction can become unacceptably long. Second, the interpretation space of a keyword query on a very large database is usually too big to be materialized completely. The existing approaches to database keyword search usually rely on an entirely materialized interpretation space to retrieve top-k structured queries and interaction options. These approaches become infeasible in face of large scale databases. Thus we need new methods to explore the interpretation spaces of keyword queries.

The FreeQ system presented in this paper enables scalable interactive query construction over a very large database. First, we propose to connect a hierarchical ontology to the database schema. Using the general concepts in the ontology, we can form more informative interaction options that enable more efficient query construction. The hierarchical ontology can be a manually constructed one, such as the domain set of Freebase. It can also be a generic external ontology, such as YAGO [21]. We conducted both theoretical and experimental studies to evaluate how a hierarchical ontology can speedup interactive query construction. Second, to explore the interpretation space of keyword queries, we design a scheme that is able to efficiently generate top-k structured queries and the optimal interaction options without the complete knowledge of the search space. Finally, we conducted extensive experiments on Freebase demonstrating the effectiveness and the efficiency of our approach.

In summary, we have made the following contributions: (i) A new type of interaction options based on ontologies to enable scalable interactive query construction, and a theoretical justification about the effectiveness of these options; (ii) A scheme to enable efficient generation of top-k structured queries and interaction options, without the complete knowledge of the query interpretation space; (iii) An experimental study on Freebase to verify the effectiveness and efficiency of the proposed approach; (iv) To the best of our knowledge, this is the first attempt to enable effective keyword-based query construction on such a large scale database as Freebase, considering that most existing work on database keyword search uses only test sets of small schemas, such as DBLP, IMDB, etc.

2. PROBLEM DEFINITION

A user interface for interactive query construction is presented in Figure 1. This interface is composed of four parts: (1) an input field for keyword queries, (2) a query construction panel for presenting interaction options, (3) a query window for presenting structured queries, and (4) a result window for query results. Suppose a user, whose name is Alice, issues a keyword query to the



Tom Hanks Terminal

Select the first correct option from the top:

- Freebase category: Arts & Entertainment : [hanks, tom]
- Freebase category: Society : [hanks, tom]
- Freebase category: Arts & Entertainment : [tom]
- Freebase category: Sports : [tom]
- Freebase category: Arts & Entertainment : [hanks]
- Others

Structured Queries (double click to execute): This list might be incomplete! Please select more o

Film
Film.Produced By : tom hanks
Film.Name : terminal

Film
Film.Directed By : tom hanks
Film.Name : terminal

TV Episode
TV Episode.Previous Episode : tom hanks
TV Episode.Name : terminal

TV Episode
TV Episode.Previous Episode : terminal

Film actor (/m/0bxtg):
Name : TOM HANKS
Film : /m/0k7dvr,/m/0k58q1,/m/0k7t0j,/m/0k5ycq,/m/0jtkml, ...

Film performance (/m/0k6myv):
Character : Viktor Navorski
Actor : TOM HANKS
Film : The TERMINAL

Name : The TERMINAL
Film Production Design By : Alex M
Runtime : /m/0k6mzg
Soundtrack : The TERMINAL
Rating : PG-13 (USA)
Featured Film Locations : New Yo
Costume Design By : Mary Zophre
Written By : Jeff Nathanson,Sacha
Directed By : Steven Spielberg
Language : French
Starring : /m/03jp
Edited By : Michae

Figure 1: FreeQ GUI. Its components include: (1) an input field for keyword queries, (2) a query construction panel, (3) top-k structured queries, and (4) query results.

system. The system first tries to guess Alice’s intent and generates the top-k most likely structured queries in the query window (3). If one of the top-k structured queries matches Alice’s intent, she can click the query to obtain the results (4). If no query in the top-k list makes sense to Alice, she can interact with the query construction panel (2) to construct the desired structured query. Whenever Alice clicks on an interaction option in the query construction panel, the structured queries in the query window (3) are refined, such that only the queries complying with Alice’s selection are preserved. Simultaneously, a new set of query construction options is presented in the query construction panel (2). The interaction continues until Alice obtains the desired structured query and results.

In this section, we introduce the basic model for enabling such interface for interactive query construction. We also elaborate on the challenges posed by large scale databases.

2.1 User Interaction Scheme

We start the discussion of the process of interactive query construction with the definition of the key concepts of this process. First of all, we model the schema of a database as a graph.

Definition 2.1. *A* **schema graph** *is a graph $G = (V,E)$, where each vertex $v \in V$ represents a relational table and each edge $e \in E$ represents a foreign key relationship. In the graph, each node v is associated with a set of attributes, denoted by $A(v)$, where the i^{th} attribute is represented by $v.a_i \in A(v)$.* $\Box$

We use the number of vertices to represent the size of a schema graph. Using the structures in a schema graph, we can create structured queries.

Definition 2.2. *Given a schema graph $G = (V,E)$, a* **structured query** *is an edge preserving map $G' = (V',E')$, such that there is a function $L : V' \to V$ which satisfies: for each vertex $v' \in V'$ in the structured query, there is a vertex $L(v') \in V$ in the schema graph such that v' and $L(v')$ represent the same relational table, and for each edge $\{v'_1, v'_2\} \in E'$ in the structured query, there is an edge $\{L(v'_1), L(v'_2)\} \in E$ in the schema graph.*

In addition, each vertex v' in the structured query can be associated with a number of predicates. Each predicate is in the form $v'.a_i$ op c_i, where $v'.a_i$ is an attribute of v', op is a comparison operator, and c_i is a constant. $\Box$

For instance, given a film database, a query looking for all the actors who have collaborated with Tom Hanks can be expressed as:

Q_1= {structure:$actor_1 \bowtie acts \bowtie film_1 \bowtie acts \bowtie actor_2$, predicates:$actor_1.name = $"*Tom Hanks*"}.

It is worth mentioning that each table in the database occurs only once in the schema graph. In contrast, a table can occur multiple times in a structured query.

In the process of interactive query construction, users express their informational needs as keyword queries.

DEFINITION 2.3. *A* **keyword query** *is a bag of terms $K = \{k_1, k_2, ..., k_n\}$, where duplicates are allowed.* □

In Definition 2.3, a term is a normalized class of tokens that is included in the system's dictionary. For token normalization, state-of-the-art Information Retrieval techniques such as case folding and word segmentation can be applied [18].

The main function of FreeQ is to translate a user's keyword query into the intended structured query. We call the structured query resulting from such translation a **query interpretation**. We say that a query interpretation is **complete** if this query interpretation contains all keywords from the initial user query. Otherwise we talk about **partial** query interpretation. We call the set of all complete query interpretations of a keyword query K an **interpretation space** of K.

For instance, the keyword query *"Tom Hanks Film"* seeking the movies starring Tom Hanks can be interpreted to:

Q_2={structure:$actor \bowtie acts \bowtie film$, predicates:$actor.name = $"*Tom Hanks*"}.

In this interpretation, keywords *"Tom Hanks"* are mapped to the constant of a predicate, and *"Film"* is mapped to a table name. The following query is a partial interpretation of *"Tom Hanks Film"*, where only *"Tom Hanks"* is interpreted:

O_1 ={structure: $actor$, predicates: $actor.name = $"*Tom Hanks*"}.

In the process of query construction we interpret user's keywords and generate query construction options (QCOs) to assess the meaning of the keywords intended by the user. We can do that in two ways: First, we can interpret keywords very specifically as a part of a structured query (as performed in [7]). We refer to these QCOs as **query-based QCOs**. For instance, O_1 can be used as a QCO, which indicates that *"Tom Hanks"* should be interpreted as an actor's name. Second, we can also assess the general meaning of the keywords and interpret them as a generic concept representing a class of structured queries. For example, we can interpret *"Tom Hanks"* as a more generic class person, which is a superclass of actor:

O_2 ={structure: $person$, predicates: $person.name = $"*Tom Hanks*"}.

To enable efficient user interaction over large database schema, in this paper we introduce ontology-based QCOs. In a generic object-relational database, a table can be regarded as an entity type, and an attribute of the table can be regarded as a property. Using a hierarchical ontology, a set of entity types can be abstracted into a superclass, and a set of properties can be abstracted into a super-property. For instance, entity types *painter* and *musician* can be abstracted into *artist*, and their properties *painting* and *music* can be abstracted into *work*. Using these superclasses and super-properties, we can create general QCOs that subsume larger proportions of a query interpretation space than query-based QCOs.

DEFINITION 2.4. *Given a schema graph $G = (V,E)$, we use $sv \vdash v$ to denote that sv is a superclass of $v \in V$ and $se \vdash e$ to denote that se is a super-property of $e \in E$. Superclass and super-properties are partial order relationships.* □

With the concepts of superclass and super-property, we define ontology-based query interpretations and ontology-based QCOs.

DEFINITION 2.5. *Let $K = \{k_1, k_2, ..., k_n\}$ be a keyword query. Let $Q = (V,E)$ be a query interpretation of K. Let Q^o be isomorphic to Q, where the isomorphism function is $f(.)$ (that applies to the predicates too). Q^o is an* **ontology-based interpretation** *of K, iff $f(.)$ satisfies: (1) for all $v \in V$, $f(v) \vdash v$; (2) for all $e \in E$, $f(e) \vdash e$; (3) for all attributes $v.a_i \in A(v)$ in the predicates, $f(v.a_i) \vdash v.a_i$. We say that Q^o is a* **super-interpretation** *of Q.* □

When we use ontology-based interpretations as QCOs, we call them **ontology-based QCOs**. In summary, QCOs generated by our system can be either query-based or ontology-based QCOs.

DEFINITION 2.6. *A* **Query Construction Option** *(QCO) is a mapping from a subset of keyword query $K' \subseteq K$ to either:*

- *a structured query Q (a query-based QCO), or*
- *an ontology-based interpretation of K' (an ontology-based QCO).* □

In the interaction process the user is supposed to select the options that subsume her intended query interpretation.

DEFINITION 2.7. *Given a QCO O and a QCO O', we say that O* **subsumes** *O', if either:*

- *O is a subgraph of O', or*
- *O is a super-interpretation of O'.*

Subsumption relationship is transitive, i.e. if O **subsumes** *O' and O'* **subsumes** *O'', then O* **subsumes** *O''.* □

Certainly, as O_1 is a subgraph of the interpretation Q_2, O_1 subsumes Q_2. O_2 is an ontological interpretation of *"Tom Hanks"*, and it is a super-interpretation of O_1. As subsumption relationship is transitive, both O_2 and O_1 subsume Q_2, which is a complete interpretation of the query *"Tom Hanks Film"*.

With the above concepts, the conceptual process of interactive query construction can be modeled as follows:

0. Given a database whose schema is $G = (V,E)$, a user issues a keyword query $K = \{k_1, k_2, ..., k_n\}$.
1. **Initialization**: Let ζ be an interpretation space of K based on G.
2. **Top-*k* Generation**: The system retrieves the top-k interpretations from ζ, and presents them to the user. If the user finds the intended interpretation in the top-k, the query construction process terminates. Otherwise, the process continues with Step 3.
3. **QCO Generation**: The system generates a QCO O and lets the user decide whether O subsumes the intended query interpretation of K.

4. **Post Interaction**: If the user indicates that O subsumes the intended interpretation of K, then the system removes all the interpretations that cannot be subsumed by O from ζ. Otherwise, the system removes all the interpretations subsumed by O from ζ. Go back to Step 2.

Each iteration of the process requires one round of interaction with the user. As the interaction goes on, the interpretation space ζ keep shrinking. Because ζ is finite, the process guarantees to terminate at a certain point. Nevertheless, we would like the process to be short, so that users can obtain desired information as early as possible. The efficiency of query construction can be measured naturally by the number of iterations of the process. We call this measure interaction cost.

DEFINITION 2.8. *Given a process of interactive query construction, its* **interaction cost** *is the number of iterations it has been executed, which is equivalent to the number of QCOs evaluated by the user.* □

2.2 Challenges

When a database, and especially its schema graph, becomes big, it is difficult for the existing approaches to realize an efficient query construction process. This is due to the following two limitations:

PROBLEM 1. *Inefficient query-based QCOs:*

To minimize the interaction cost, the query construction process needs to shrink the query interpretation space quickly. In other words, the evaluation of each QCO should be able to remove a significant proportion of the interpretation space. Therefore, we desire the proportion of query interpretations subsumed by each QCO to fall in a certain range. This proportion should not be too small, as in this case the denial of a QCO could not reduce the interpretation space effectively. It should not be too big either, as in this case the acceptance of a QCO could not reduce the interpretation space effectively.

When the schema graph is big, a keyword can have a large number of occurrences spread across the database, resulting in a vast number of partial interpretations (query-based QCOs). The proportion of the interpretation space subsumed by each query-based QCO will be very small. As a result, query construction processes using only query-based QCOs, such as [7], cannot be efficient. Apart from query-based QCOs, we need more general QCOs to enable efficient query construction.

PROBLEM 2. *Very large query interpretation space:*

When the schema graph becomes big, it is no longer feasible to materialize the interpretation space of a complex keyword query entirely. On the one hand, with an increasing size of a schema graph, the number of its subgraphs grows very sharply. On the other hand, the occurrences of keywords are more numerous in a larger database. As a result, the number of the possible interpretations of a keyword query grows quickly with an increasing size of the schema.

The existing approaches to interactive query construction [7], [8], as well as the state-of-the-art approaches to schema-based database keyword search [2], [12], [16], [17] rely on an entirely materialized query interpretation space. These approaches can hardly work with a big schema, as it is infeasible to generate all the query interpretations at the query time. Therefore, we need a new mechanism which can enable efficient identification of the most efficient QCOs and the top-k most probable query interpretations, without the knowledge of the complete interpretation space.

Compared with the most recent approach to interactive query construction [7], in this paper we addressed the problems listed above and made the following contributions:

- We propose novel ontology-based QCOs that enable efficient query construction process over large scale data. Using these QCOs we can enable efficient query construction for ambiguous queries that have many different interpretations in a large database.
- We develop procedures and algorithms for incremental materialization of query interpretation spaces over large scale data that enables us to perform efficient query construction without materialization of the complete query interpretation space.

We present our solutions to each of these problems in Sections 3 and 4, respectively.

3. NOVEL ONTOLOGY-BASED QCOS

As pointed out by Problem 1 in Section 2.2, to minimize the interaction cost, the QCOs presented to the user need to shrink the query interpretation space quickly. In a large scale database, each single keyword can have numerous occurrences. As a result, the query-based QCOs utilized by the existing approaches [7] become inefficient in reducing interpretation spaces. In this section, we introduce a novel type of QCOs, called ontology-based QCOs. An ontology-based QCO can subsume a wider proportion of an interpretation space, such that it is usually more efficient than a query-based QCO. Intuitively, if the user provides feedback on an ontology-based QCO, we get an implicit user's feedback on multiple partial interpretations (i.e. query-based QCOs) subsumed by this ontology-based QCO within a single user interaction. Thus a query construction process using ontology-based QCOs requires less steps. In this section, we justify the efficiency of ontology-based QCOs from the perspective of information theory.

3.1 Creation of Ontology-based QCOs

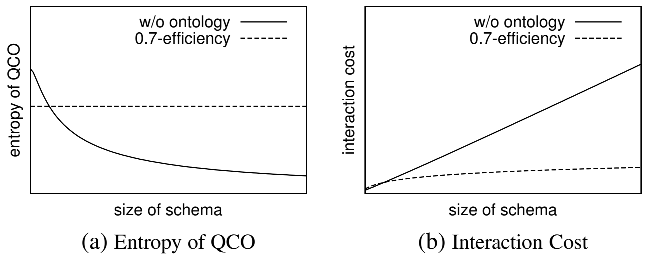
Ontology-based QCOs can be created based on a hierarchical ontology or taxonomy. In order to create ontology-based QCOs, we need an ontology on top of the database schema, which defines superclasses and super-properties. This ontology can be a manually defined one, such as the domain hierarchy of Freebase. Alternatively, we can utilize external ontologies, such as e.g. WordNet [9], and YAGO [21], by mapping the elements of the database schema to the concepts in these ontologies. In this case state-of-the-art schema matching techniques can be used (see [6] for details).

With ontology-based QCOs, we can enable more efficient query construction, especially when confronted with a big database schema. To illustrate a query construction process using ontology-based QCOs, we consider the query *"Emperor Album"*, which intends to retrieve the albums of the artist Emperor from Freebase. To create the ontology-based QCOs, we make use of the domain hierarchy of Freebase. This hierarchy groups together Freebase tables such as *artist*, *album*, and *monarch* in the domains e.g. *music* and *royalty*, and further organizes these domains into the categories such as *Arts & Entertainment* and *Society*.

For this particular query, if we use only query-based QCOs (as performed by [7]), our system requires a user to interact with 74 QCOs to identify the intended interpretation. Using ontology-based QCOs, the user only needs to interact with the 10 QCOs listed in Table 1. The keyword *"album"* is not very ambiguous, as it occurs mostly in the domain of *music*. To disambiguate this keyword, FreeQ does not utilize any ontology-based QCOs. In contrast, the

QCOs (bold ones are ontology-based QCOs)	User's feed-back
Table *artist* (domain *music*): "album"	×
Table *release* (domain *music*): "album"	×
Table *recording contribution* (domain *music*): "album"	×
Table *album* (domain *music*): "album"	✓
Domain *royalty* (Category *Society*): "emperor"	×
Category *Arts & Entertainment*: "emperor"	✓
Domain *fictional universe* (Arts & Ent.): "emperor"	×
Domain *opera* (Arts & Ent.): "emperor"	×
Domain *media common* (Arts & Ent.): "emperor"	×
Query: **album** $\bowtie$ **artist** "emperor"$\subset$artist.name	✓

Table 1: A Query Construction Example for the Query *"Emperor Album"* using Ontology-based QCOs



(a) Entropy of QCO (b) Interaction Cost

Figure 2: Efficiency of QCO and Interaction Cost vs. Schema Size

keyword *"emperor"* is very ambiguous. *"Emperor"* occurs in 221 attributes of Freebase, which are spread across multiple categories and domains. To disambiguate *"emperor"*, it is much faster if we use ontology-based QCOs. With ontology-based QCOs, we manage to first restrict the meaning of *"emperor"* to the category of *Arts & Entertainment*. Within this category, the exact meaning of *"emperor"* can be identified easily.

In what follows, we analyze how ontology-based QCOs achieve such efficiency.

3.2 A Measure of QCO Efficiency

As depicted in Section 2.2, in each round of interactive query construction, FreeQ needs to select one QCO to present to the user. For a keyword query, there is usually a large number of available QCOs. In principle, FreeQ should always select the most efficient QCO that can minimize the final interaction cost. The efficiency of a QCO can be quantified using information theory.

Let ζ denote the interpretation space of a keyword query K. Then, the uncertainty of K's interpretation can be measured by the entropy $H(\zeta)$, which can be computed as:

$$H(\zeta) = -\sum_{I \in \zeta} P(I) \times log_2 P(I), \tag{1}$$

where $P(I)$ denotes the probability that the interpretation I is the interpretation intended by the user.

The process of query construction is the process of reducing the uncertainty of K's interpretation. After one round of interaction with the user, FreeQ obtains the knowledge of one QCO, say O. Then, the uncertainty is reduced to $H(\zeta|O)$, i.e., the conditional entropy of ζ given O. The difference between $H(\zeta)$ and $H(\zeta|O)$ is known as the expected *information gain* provided by O, denoted by:

$$IG(O) = H(\zeta) - H(\zeta|O). \tag{2}$$

To minimize the interaction cost, we need maximize the information gain of each QCO presented to the user.

Obviously, the knowledge about ζ contains the knowledge of any O. In other words, the information gain provided by O is exactly the entropy of O. Therefore, we have:

$$IG(O) = H(\zeta) - H(\zeta|O) = H(O). \tag{3}$$

In turn, the entropy of O can be calculated as:

$$H(O) = -P(O) log_2 P(O) - P(\neg O) log_2 P(\neg O), \tag{4}$$

where $P(O)$ is the probability that O is accepted by the user. Let $\zeta(O)$ denote the complete set of query interpretations subsumed by O. Then, $P(O)$ can be computed as:

$$P(O) = \sum_{I \in \zeta(O)} P(I). \tag{5}$$

To summarize, the efficiency of a QCO can be measured by its entropy. Therefore, FreeQ is supposed to select the QCO with the highest entropy in each round of user interaction.

3.3 Effects of Ontology-based QCOs

As discussed in Section 3.2, to achieve fast query construction, in each round of interaction, FreeQ is supposed to present the user with an efficient QCO, i.e., a QCO whose entropy is sufficiently high. The question is whether such QCOs would be available at all. To answer this question, we first propose the following measure to quantify the efficiency of an entire query construction process.

DEFINITION 3.1. *During an interactive query construction process, if we can ensure with a high probability that the entropy of each QCO presented to the user is larger than* ε *(*$0 < \varepsilon \leq 1$*), we say that the query construction process is* **ε-efficient**.

According to Definition 3.1, to minimize the interaction cost, we should maximize the lower bound (ε) of the efficiency of the QCOs presented to the user. We can show that query-based QCOs, i.e. the QCOs used in the existing approaches [7], cannot guarantee a lower bound. In contrast, if we have an ontology with a sufficient number of concepts of diverse generality, by using ontology-based QCOs, we can achieve ε-efficient query construction with a good lower bound ε.

To simplify our analysis, we assume: (1) the number of possible interpretations of a keyword grows linearly with the size of the schema graph; (2) the number of the possible interpretations of an entire keyword query increases polynomially; (3) all the complete query interpretations are equally probable.

Let the size of the schema be x. According to (1), the number of the query-based QCOs for a keyword can be expressed as $\alpha \times x$, where α is a constant. Then the entropy of a query-based QCO for each keyword will be $H(\frac{1}{\alpha \times x})$, which can be plotted as the solid curve in Figure 2a. We can see that when the database schema grows, the efficiency of each query-based QCO will decrease. This efficiency can drop to a very small value when the database schema is big.

According to (2), the size of an interpretation space can be modeled as $\beta \times x^{\gamma}$, where β and γ are constants. Based on (3), the most efficient query-based QCO will be an interpretation for a single keyword, whose entropy can be modeled as: $H(\frac{1}{\alpha \times x})$. Therefore, the average interaction cost can be calculated as the entropy of the whole interpretation space divided by the maximum entropy of an QCO, i.e. $log_2(\beta \times x^{\gamma})/H(\frac{1}{\alpha \times x})$, which can be plotted as the solid curve in Figure 2b. As we can see, if we use only query-based QCOs, the interaction cost can increase quickly with the size of the database schema.

In contrast, if we can achieve a 0.7-efficient query construction process, that is, the entropy of each QCO be no less than 0.7 (as illustrated by the dashed line in Figure 2a), the growth of the interaction cost can be significantly reduced (as illustrated by the dashed line in Figure 2b). This is achievable, if we utilize ontology-based QCOs.

The concepts in an ontology normally have a variety of generality. There are very specific concepts that can subsume small sets of entity types, such as *artist* and *book*. There are also very general concepts that can subsume larger proportions of entity types, such as *person* and *artifact*. As a result, no matter how big the query interpretation space is, it is always possible to find suitable concepts to form QCOs that can subsume a certain proportion of the interpretation space that would yield big entropies. As a simple analysis, we assume that the probability of a random ontology-based QCO is a random value between 0 and 1. Then, within the set of N ontology-based QCOs, the probability that we can find a QCO whose entropy is larger than ε is:

$$P_{\exists o:H(o)>\varepsilon} = 1 - \left(\frac{H^{-1}(\varepsilon)}{0.5}\right)^N, \tag{6}$$

where $H^{-1}()$ is the inverse function of the binary entropy. In this function, no matter how big ε is, we can always find a big enough N, such that the resulting probability is close to 1. (As N is an exponent in the formula, it normally does not need to be very big.) In other words, as long as there is a rich ontology with a sufficient number of concepts of diverse generality, we can achieve ε-efficiency for interactive query construction.

4. INCREMENTAL MATERIALIZATION OF QUERY INTERPRETATION SPACES

To perform query construction, FreeQ is required to quickly generate the most efficient QCOs and the most probable complete query interpretations. As mentioned in Problem 2, Section 2.2, the existing approaches to interactive query construction and schema-based database keyword search in general require a complete materialization of the query interpretation space [2], [12], [16], [17], [7]. For a large scale dataset, the interpretation space of a keyword query is usually very big, such that it is no longer feasible to materialize this space entirely. To this end, we develop new algorithms that can generate top-k most probable query interpretations and QCOs without the knowledge of the entire interpretation space.

4.1 Query Hierarchy for the QCO Generation

To enable efficient generation of QCOs and query interpretations, we organize the QCOs of a keyword query in a query hierarchy based on their subsumption relationships. Using the query hierarchy, we can materialize the interpretation space of a keyword query step-by-step by following the subsumption relationships of the QCOs. During the progress of the materialization, a lot of QCOs and interpretations can be eliminated based on the information provided by user interaction. Such progressive materialization is much less costly than the generation of an entire interpretation space.

Figure 3 illustrates a query hierarchy, in which the arrows represent the reversed subsumption relationships. The more general the QCOs, the lower their positions in the query hierarchy. The most general QCOs are the single-node QCOs, i.e. the QCOs involving only one entity type such as O_2, O_1, and O_3. These QCOs are located at the bottom of the hierarchy. The top level of the query hierarchy consists of the complete query interpretations (e.g. Q_3, and Q_4), which constitute the interpretation space of a keyword query. The entire query hierarchy looks like an upside-down trapezoid, as there are much more complete query interpretations than single-node QCOs.

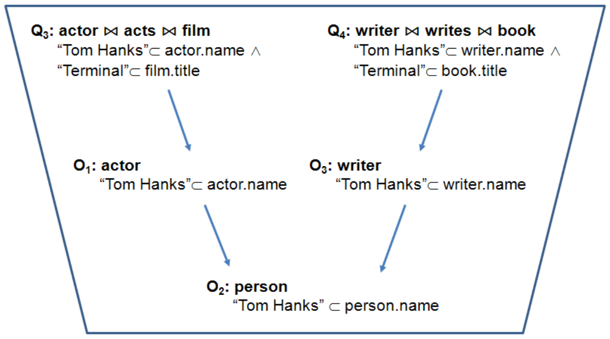

Q_3: actor ⋈ acts ⋈ film
"Tom Hanks"⊂ actor.name ∧
"Terminal"⊂ film.title
Q_4: writer ⋈ writes ⋈ book
"Tom Hanks"⊂ writer.name ∧
"Terminal"⊂ book.title
O_1: actor
"Tom Hanks"⊂ actor.name
O_3: writer
"Tom Hanks"⊂ writer.name
O_2: person
"Tom Hanks" ⊂ person.name

Figure 3: An Example of a Query Hierarchy for the Query "Tom Hanks Terminal" (the arrows represent reversed subsumption relationships, e.g. O_2 subsumes O_1).

As mentioned previously, it is infeasible to instantiate the complete query hierarchy at the query time. Given a big schema, the top levels of the query hierarchy can even be too big to be accommodated in the main memory. Therefore, FreeQ chooses to instantiate the query hierarchy incrementally throughout the process of query construction. The query construction process of FreeQ conforms to the generic process defined in Section 2.1, except that it does not materialize the entire query interpretation space. This process works as follows:

- **1. Initialization:** Upon receiving a keyword query, FreeQ identifies database attributes and schema elements containing keyword occurrences using an inverted index. Based on the keyword occurrences and an ontology, it generates the most general QCOs. As a result, the bottom of the query hierarchy is instantiated. As the bottom of the query hierarchy is small, its instantiation does not incur much cost.

- **2. Top-*k* Generation:** To generate the top-k complete query interpretations, FreeQ performs a **top-*k* depth first traversal (DF-*k*)** of the query hierarchy from the bottom up. All the QCOs and interpretations encountered during the DF-k are instantiated. DF-k enables FreeQ to reach the top of the query hierarchy by instantiating the minimal number of QCOs. The first k complete interpretations encountered by the DF-k are presented to the user as the top-k interpretations.

- **3. QCO Generation:** FreeQ evaluates the QCOs in the instantiated part of the query hierarchy and selects the QCO with the highest entropy. This QCO is presented to the user.

- **4. Post Interaction:** After the user provides feedback on the QCO, FreeQ truncates the query hierarchy according to the user's selection. If the user has denied the QCO, all the QCOs or query interpretations subsumed by the denied QCO are removed from the query hierarchy. If the user has accepted a QCO, only the QCOs and interpretations subsumed by the accepted QCO are preserved. After the truncation, the query hierarchy becomes smaller. If the size of the instantiated part of the hierarchy falls below a threshold, FreeQ perform a **top-*k* breadth first traversal (BF-*k*)** of the query hierarchy (from the bottom up), during which more QCOs are instantiated. The BF-k stops as the size of the instantiated part reaches a predefined upper bound.

As we can see, as the process of the interactive query construction goes on, the instantiated part of the query hierarchy remains stable in size. On the one hand, this part is truncated as the user reveals more information in the interaction process. On the other hand, the instantiated part is continuously expanded by the BF-*k* and DF-*k* to ensure that its size is sufficient to generate efficient QCOs and high quality top-*k* interpretations.

4.2 Algorithms for Efficient Top-*k* Generation

FreeQ obtains top-*k* query interpretations and query construction options by traversing the query hierarchy. To make interactive query construction smooth to the user, it is important to ensure the efficiency of the hierarchy traversal. Although in the related work the basic Breadth First (BF) and Depth First path search (DF) algorithms are typically used to materialize interpretation spaces of keyword queries [2], [12], [11], [16], these algorithms not scale on large database schemas.

The basic BF and DF path search applied in the state-of-the-art database keyword search seek to connect occurrences of the user's keywords in the database schema to materialize query interpretation spaces. For example, to materialize query interpretation space, Discover [12] explores the schema graph in a BF-manner starting from all tables containing occurrences of a keyword $k_i \in K$. The time complexity of this approach can be computed as $T_{k_i} * avg(E_t)^{r*(K-1)}$, where T_{k_i} is the number of tables containing keyword k_i, $avg(E_t)$ is an average fan-out of a table in the schema graph, K is the number of terms in the keyword query, and r is the maximum length of the path within the schema graph explored to connect occurrences of two keywords. The time complexity of this approach increases exponentially with the number of user's keywords and the length of the explored path. This makes materialization of the entire query interpretation space infeasible for longer user queries.

To address these limitations, we propose several important adaptations of the basic BF and DF path search algorithms, to enable efficient generation of the top-*k* query interpretations and QCOs over the large database schemas. We refer to these algorithms as BF-*k* and DF-*k*, respectively. Both BF-*k* and DF-*k* traverse the query hierarchy by following the subsumption relationships. Each traversal step expands a QCO to one of the QCOs it subsumes, e.g. expanding O_2 to O_1, or O_1 to Q_3 in Figure 3. Therefore, it is crucial to make the QCO expansion efficient.

In principle, a QCO can be expanded in two different ways. **The first type of QCO expansion** is to replace an entity type or a property of the QCO with its subclass or sub-property. Such expansion does not add any additional keywords to the QCO. For example, the expansion of O_2 to O_1 in Figure 3 belongs to the first type. This type of expansion requires only the knowledge of the ontology structure. Once we have indexed the ontology, such that its subclasses or sub-properties can be retrieved quickly, this type of expansion will be very efficient. **The second type of QCO expansion** is to find the paths in the schema graph to connect a QCO and an additional keyword occurrence. For instance, to expand O_1 to Q_3 in Figure 3, we need to identify the path "$actor \bowtie acts \bowtie film$" to connect O_1 and the keyword occurrence $film.title : terminal$.

To identify the paths for connecting a QCO with a keyword occurrence efficiently, FreeQ pre-indexes all the paths leading to keyword occurrences in the schema graph starting from every table in the schema graph. The index is constructed at the beginning of the query construction, as the user issues a keyword query. To ensure scalability of the indexing, we restrict the maximal size of the path connecting two keyword nodes in the structured queries that can be constructed by FreeQ and perform bi-directional search. The time complexity of the indexing procedure of FreeQ can be computed as $T * avg(E_t)^{r/2}$, where T is the number of tables in the database schema, $avg(E_t)$ is an average fan-out of a table in the schema graph, and $r/2$ is the half of the maximum length of the explored path. As we can observe, the time complexity of this approach is significantly reduced compared to Discover [12] (discussed above) and does not depend on the size of the keyword query.

To maximize the quality of the top-*k* query interpretations and QCOs generated by FreeQ, we always start the expansion with the QCOs that are most likely to be intended by the user. This applies to both DF-*k* and BF-*k*. As a result, the top-*k* interpretations generated by DF-*k* will be more likely to meet the user's requirements, and the QCOs instantiated by BF-*k* will be more likely to have high entropy. To ensure a short response time of FreeQ, we also enforce a time limit on both BF-*k* and DF-*k*. That is, we stop BF-*k* and DF-*k* once the time limit has been reached, even though the top-*k* interpretations may have not been completely generated, or the number of the instantiated QCOs has not yet reached a predefined upper bound. We demonstrate through the experiments that the resulting hierarchy traversal procedure is effective and efficient.

4.3 Probability Estimation for QCOs

The QCOs in the query hierarchy are all the candidates to be presented to the user. According to Section 3.2, the most efficient QCO is the QCO with the maximum entropy. To identify this QCO, we need to estimate the probabilities of the QCOs. These probabilities cannot be computed by Equation 5, as we do not have the entire query interpretation space materialized. Instead, we consider the frontier of the instantiated part of the query hierarchy. This frontier contains the most specific query-based QCOs (i.e. partial interpretations) that have been materialized so far. We treat all the QCOs in the frontier as samples, such that each QCO in the frontier represents the complete queries that can be derived from this QCO by further expansion of the hierarchy. For each sample represented by s, we compare this sample against each candidate QCO, say o. The comparison can end up with one of the following conclusions: (1) o subsumes s; (2) s conflicts with o (e.g. one keyword is being interpreted into two different meanings); (3) none of the above. If the conclusion is (3), s does not provide any helpful information. With (1) or (2), we can know that either $\zeta(s) \subset \zeta(o)$, or $\zeta(s) \cap \zeta(o) = \varnothing$, respectively. By applying the maximum likelihood principle, we can estimate the probability $P(o)$ using $P(s)$, that is,

$$P(o) = \frac{\sum_{\zeta(s)\subset\zeta(o)} P(s)}{\sum_{\zeta(s)\subset\zeta(o)} P(s) + \sum_{\zeta(s)\cap\zeta(o)=\varnothing} P(s)}, \qquad (7)$$

where $P(o)$ and $P(s)$ are the probabilities that o and s are accepted by the user, respectively. With $P(o)$, we can obtain the entropy of o based on Equation 5. Therefore, the problem of finding the QCO with the maximum entropy is reduced to the estimation of the probabilities of (partial) query interpretations $P(s)$. FreeQ applies the probabilistic model in [7] to estimate these probabilities.

5. EXPERIMENTAL EVALUATION

We have implemented FreeQ, an interactive query construction system based on the mechanisms proposed in this paper. To evaluate the performance of FreeQ, we conducted extensive experiments. Our experiments include two parts. First, we evaluated the impact of ontology-based QCOs on the efficiency of query construction processes. Then, we evaluated the time efficiency of the proposed algorithms in handling large scale databases.

#Key-words	Avg. #nodes	Examples of keyword queries
1	1.00	location, book, event, disease, election
2	1.43	emperor album, hockey team, alpine skier
3	1.92	artist lived vancouver, founding figure kagyu
4	2.40	olympic athletes table tennis
5	2.11	canada hockey 2010 winter olympics
6	1.48	2011 san francisco international film festival
7	1.29	fictional character created by edgar allan poe
8	1.13	school type university of puerto rico at ponce

Table 2: Complexity of Keyword Queries

5.1 Dataset and Queries

Our experiments were performed on a Freebase dataset downloaded in June 2011 [10]. This dataset contains approximately 7,500 tables with more than 20 million entities in about 100 domains. We imported the dataset into a MySQL database and indexed the data and the schema using Apache Solr[6]. FreeQ was implemented as a client-server Java application. For our experiments, we used two cores and 10GB of memory on a server, which was equipped with 8x Quad-Core AMD Opteron 2.7GHz processors and 256GB of memory.

Our query set was based on the user-defined views of Freebase. Freebase allows users to create views using a dedicated query language called MQL. Each of the views is given a descriptive title in a natural language. Some examples of the view titles are: *"Lionel Richie discography"*, *"Directing Award: U.S. Dramatic - Winning Films"*, and *"TV Celebrities on Twitter"*. For evaluation, we can regard the title of each view as a keyword query and the MQL definition of the view as the structural interpretation. Then we can study how FreeQ can assist users to construct the structural interpretation from the keyword query. The ground truth is a plausible mapping between the keyword query and the structural interpretation and can be automatically established through a projection program. For our experiments, we randomly selected a set of 615 keyword queries (views). The number of keywords in each query ranges from 1 to 8 keywords. The structural interpretations of the keyword queries are of different complexity. We measure the complexity of an interpretation by the number of keyword nodes it contains. (A keyword node is a table containing at least one keyword occurrence.) The complexity of the queries in our query log ranges from 1 to 3. Table 2 presents the average complexity of queries with different number of keywords. As we can observe, the relatively complex queries mostly contain 3-5 keywords.

5.2 Effectiveness of Ontology-based QCOs

Our first set of experiments was intended to evaluate the effectiveness of the ontology-based QCOs. To this extent, we perform two sets of experiments: First, we compare our approach to that in [7], while using different ontologies to generate QCOs. Second, we compare our approach against that of ranking using different ranking functions.

For each query in our query set, its correct interpretation is already known. Thus, the correctness of each QCO generated by FreeQ can be determined without any user intervention. In [7] we evaluated accessibility of the user interface of incremental query construction in a user study, and demonstrated that the users could make the right choice of QCOs using this interface. In this paper we evaluate how big the interaction cost of query construction is, given that the user makes the right choices. In our experiments, we let the computer interact with FreeQ automatically. In this interaction, the computer will always accept the QCO presented by the system if this QCO subsumes the correct query interpretation and reject the QCO otherwise.

[6]https://lucene.apache.org/solr

Notation	Description	Size
NoOntology	no ontology is used, representing the baseline approach [7]	zero
Freebase	the ontology of Freebase, consisting of domains and categories	medium
YAGO	an external ontology YAGO	big

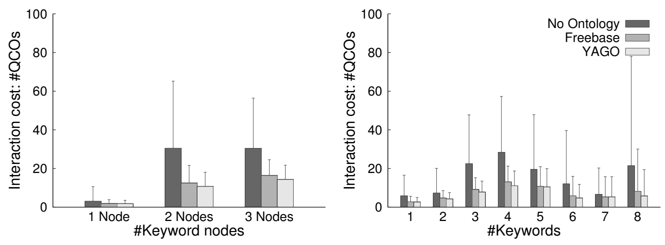
Table 3: Ontologies of Different Size

To evaluate the effectiveness of the ontology-based QCOs, we ran the experiments in a number of scenarios, in which different ontologies were used. The ontologies are of different size and complexity, as summarized in Table 3. *NoOntology* represents the baseline approach [7], where no ontology is used. *Freebase* represents the ontology based on the domains and categories given in the Freebase website. *YAGO* represents an external ontology known as YAGO. To associate each table of Freebase with a suitable YAGO category we used YAGO+F mapping described in [6].

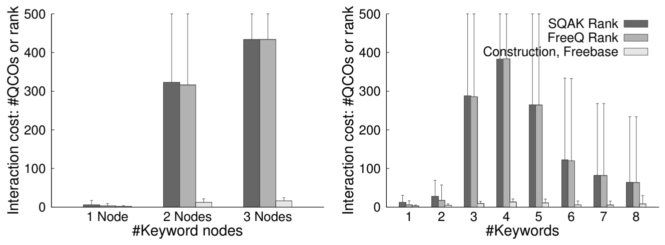
In the experiments, we measured the interaction cost of query construction for the queries with different complexity in different scenarios. According to Definition 2.8, interaction cost is the number of the QCOs a user needs to evaluate to construct a structured query. The results for the 615 test queries are presented in Figure 4a.

Figure 4a presents the mean and the standard deviation of the interaction cost, with respect to query complexity (the number of keyword nodes) and number of query terms. As we can see, compared to the baseline *NoOntology* approach, the interaction cost can be significantly reduced by using the ontology-based QCOs. With a larger ontology, such as YAGO, the interaction cost tends to be smaller. For example, a 2-node query *"ami suzuki album"* requires 38 QCOs with *NoOntology*, 19 QCOs with *Freebase*, and 10 QCOs with *YAGO*. The benefits of the ontology-based QCOs become stronger with an increasing number of keyword nodes in the structural interpretation too. As we can observe, the majority of the 1-node queries, such as *"university"* and *"football game"*, can still be efficiently answered in the baseline *NoOntology* scenario, where an average cost amounts to 3. With the more complex 2-node, and 3-node queries, such as *"don shirley album", "film performance tom everett", and "location leonardo da vinci lived"*, the interaction cost of *NoOntology* goes up quickly to about 30 and can occasionally exceed 70. By applying the ontology-based QCOs, this cost can be limited to a much smaller range. For example, the average cost for 2-node queries in the *YAGO* scenario is around 10. In addition, we can observe that the interaction cost tends to increase with the number of keywords, while this trend may not hold in all the cases. This is because a longer keyword query does not necessarily imply the more complex structural interpretation (see Table 2).

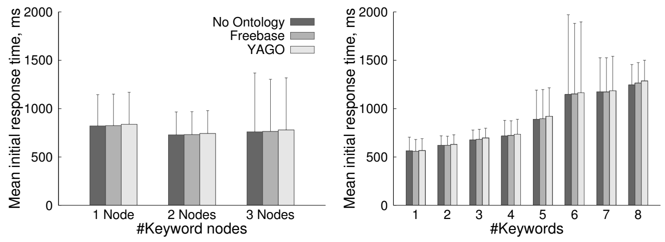
In the next set of experiments we compared the interaction cost of query construction using Freebase ontology against that of query ranking to evaluate the effectiveness of interactive query construction in general. To this end, we generated the top-500 interpretations using the DF-*k* algorithm of FreeQ. We ranked these interpretations using two ranking functions proposed recently: the ranking function of FreeQ described in [7], and SQAK [22]. The interaction cost of ranking corresponds to the number of interpretations a user needs to evaluate before the intended interpretation is identified, which is exactly the rank of the intended interpretation. If the intended interpretation was not contained within the top-500, we set its rank to 500. Figure 4b presents the results.



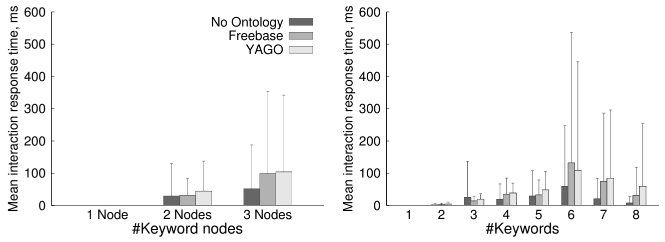
(a) Interaction Cost of Query Construction



(b) Interaction Cost of Ranking vs. Query Construction



(c) Mean Initial Response Time, ms



(d) Mean Interaction Response Time, ms

Figure 4: Effectiveness and Efficiency of Query Construction for Queries with Different Complexity

As we can observe, the interaction cost of both ranking and construction for the simplest queries (with 1-2 keywords or 1 keyword node) is acceptable. For the keyword queries that are relatively complex, ranking become ineffective. This is because the query interpretation space on Freebase is too big, such that there can always be a large number of non-intended interpretations that receive good ranks.

5.3 Performance of the System

Existing schema-based approaches to keyword search and interactive query construction in databases typically rely on the completely materialized interpretation space of a keyword query [2, 12, 7]. We have tried to run a number of existing algorithms, including [12], and [7] on Freebase. However, as the schema of Freebase is much bigger than the schemas these algorithms were designed for, they could not finish in a reasonable time. In contrast, the BF-*k* and DF-*k* approach allows FreeQ to explore the query interpretation space smartly. Thus it can achieve reasonable response time on a large scale dataset.

To assess the time efficiency of FreeQ, we measured its response time in two phases: (1) The *initial response time* when a user submits a query, and (2) the *interaction response time* when a user interacts with the query construction panel.

When a user issues a keyword query, FreeQ will perform a series of pre-processing, including the identification of the keyword occurrences in an inverted index and the creation of a bi-directional index for performing BF-*k* and DF-*k* expansions. The mean and the standard deviation of the initial response time for our query set are shown in Figure 4c. As we can see, the mean initialization time stays around 1 second, and its maximum does not exceed 2 seconds. With an increasing number of keywords, the initial response time increases slowly. This is because the time required for the identification of keyword occurrences in an inverted index increases with the number of query terms. This index access time ranges from 70 to 700ms for our query set. We can also see that the initialization time does not increase with the increasing query complexity. As presented in Table 2, a more complex query does not necessarily contain more query terms. The time required for the creation of a bi-directional index for DF-*k* and BF-*k* ranges from 360 to 460ms for our query set. This time mainly depends on the size and the complexity of the schema graph rather than the complexity of queries. We can also observe that the use of different ontologies has limited impact on the initial response time.

We also measured the *interaction response time*, i.e. the time required by FreeQ to present a new set of QCOs and top-*k* structured queries after each user interaction. This time comprises the time consumed by the BF-*k* and DF-*k* algorithms and the QCO selection. Figure 4d presents the results. As we can observe, the interaction response time of FreeQ varies from 1 to 130ms, meaning that in most cases the user would feel that the system is reacting instantaneously [20]. The interaction response time increases with an increasing query complexity. This is expected. As more keywords imply a larger query hierarchy and more QCOs to evaluate, the BF-*k* and DF-*k* algorithms and the process of QCO selection will all be more time consuming. Nevertheless, this increase does not appear steep in the results. Moreover, a larger ontology seems to incur higher interaction time too. This is because a larger ontology enables FreeQ to generate more QCOs to evaluate.

In summary, the interaction response time of our system is always below one second. Its initialization time stays within one second most of the time, except for some long queries, whose initialization time can take up to 2 seconds. While further optimization can be conducted, such performance can already ensure that the user's flow of thought stays uninterrupted [20].

6. RELATED WORK

Recently the problem of structured query generation from user's keywords has attracted a lot of research attention, e.g. [7], [22], [23]. The methods for interpreting keywords evolved from considering attribute values only, to including schema terms and operators [16], [22]. In [15], [1], the author modeled the problem as question guided search.

Demidova et al. proposed a probabilistic incremental query construction model for an interactive user interface [7]. These methods do not provide a sufficient solution to large scale datasets with flat schemas, such as the schema of Freebase. This is because these methods relied only on the database internal statistics to generate query construction options. In large scale databases, such options are not informative enough to efficiently reduce the search space. FreeQ alleviates this problem by using ontologies, such as the domain hierarchy of Freebase, and the YAGO ontology [21], in the option generation process. Furthermore, previous work on incremental query construction relied on a set of query templates generated a-priori. In contrast, FreeQ presents algorithms to generate structured queries and query construction options on the fly over a large scale database.

Database usability is a long-term research issue [13]. One of the early approaches to address this problem was the Query by Example [25]. Recent approaches include NL query interfaces [3], [5], query auto-completion, e.g. [19], and adaptive forms [14]. Some commercial DB products such as Microsoft Access offer visual query builder interfaces. Query graphs in a typical visual query builder interface have to be created starting from scratch. The user has to study the database schema and manually put together pieces of the query graph. In contrast, FreeQ enables users to focus on interpretations of keyword queries, and automatically suggests structured queries, without requiring any prior schema knowledge.

User-driven query disambiguation has been successfully applied in Information Retrieval in the context of faceted search. Faceted search engines, such as the product search engine of Google and the Yippy[7], organize search results into meaningful groups, called facets, by applying some clustering or categorization algorithms. Users can easily shrink the scope of the search by focusing on a small number of facets. Several navigational techniques, e.g. [4, 24] were proposed to support users in finding information in a hierarchy of faceted categories. The interface of FreeQ is similar to a faceted interface, whereas each facet corresponds to a query construction option.

7. CONCLUSION

In this paper, we considered the problem of scaling interactive query construction on a very large dataset, such as Freebase. To achieve this goal, we analyzed the efficiency of query construction options and extended the database schema with an ontology layer to reduce the interaction cost. Furthermore, we developed a set of algorithms to explore the interpretation spaces of keyword queries incrementally, such that top-k query interpretations and query construction options can be generated efficiently. Our results confirm the efficiency of the proposed algorithms and the effectiveness of the ontology layer created using the native taxonomy of Freebase. Furthermore, we have demonstrated that external ontologies, such as YAGO ontology, can be used to further increase the effectiveness of query construction. This is especially meaningful for the portability of the FreeQ system, as it can be applied to databases without any pre-defined ontologies.

[7]http://search.yippy.com/

8. ACKNOWLEDGMENTS

This work is partly funded by the NSFC, Project No. 61272138, the European Commission, grant agreement 270239 (ARCOMEM), BMBF ("Gute Arbeit", Re-SozIT) and KEYSTONE COST Action.

9. REFERENCES

[1] P. Adolphs, M. Theobald, U. Schäfer, H. Uszkoreit, and G. Weikum. Yago-qa: Answering questions by structured knowledge queries. In *ICSC*, 2011.

[2] S. Agrawal. Dbxplorer: A system for keyword-based search over relational databases. In *ICDE*, 2002.

[3] L. Androutsopoulos. Natural language interfaces to databases - an introduction. *Journal of Natural Language Engineering*, 1:29–81, 1995.

[4] S. Basu Roy, H. Wang, G. Das, U. Nambiar, and M. Mohania. Minimum-effort driven dynamic faceted search in structured databases. In *Proceeding of the CIKM '08*.

[5] A. Blum. Microsoft english query 7.5: Automatic extraction of semantics from relational databases and olap cubes. In *VLDB '99*.

[6] E. Demidova, I. Oelze, and W. Nejdl. Yago meets freebase: Combining a large-scale database with an ontology. Technical report, L3S Research Center, Leibniz Universität Hannover, available at: http://iqp.l3s.uni-hannover.de/yagof-TR.pdf, May 2013.

[7] E. Demidova, X. Zhou, and W. Nejdl. A probabilistic scheme for keyword-based incremental query construction. *IEEE Trans. Knowl. Data Eng.*, 24(3):426–439, 2012.

[8] E. Demidova, X. Zhou, G. Zenz, and W. Nejdl. Suits: Faceted user interface for constructing structured queries from keywords. In *DASFAA 2009*, Berlin, Heidelberg, 2009. Springer-Verlag.

[9] e. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, 1998.

[10] Google. Freebase data dumps. `http://download.freebase.com/datadumps/`, 2011.

[11] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB '2003*, pages 850–861. VLDB Endowment, 2003.

[12] V. Hristidis and Y. Papakonstantinou. Discover: keyword search in relational databases. In *VLDB*, 2002.

[13] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. In *SIGMOD*, 2007.

[14] M. Jayapandian and H. V. Jagadish. Expressive query specification through form customization. In *EDBT*, 2008.

[15] A. Kotov and C. Zhai. Towards natural question guided search. In *WWW '10*.

[16] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, 2006.

[17] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: top-k keyword query in relational databases. In *SIGMOD '07*.

[18] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[19] A. Nandi and H. V. Jagadish. Assisted querying using instant-response interfaces. In *SIGMOD*, 2007.

[20] J. Nielsen. *Usability engineering*. Morgan Kaufmann Publishers, San Francisco, Calif., 1993.

[21] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *WWW 2007*. ACM Press, 2007.

[22] E. Tata and G. M. Lohman. Sqak: doing more with keywords. In *SIGMOD*, 2008.

[23] T. Tran, P. Cimiano, S. Rudolph, and R. Studer. Ontology-based interpretation of keywords for semantic search. In *ISWC*, 2007.

[24] P. Wu, Y. Sismanis, and B. Reinwald. Towards keyword-driven analytical processing. In *SIGMOD '07*.

[25] M. M. Zloof. Query by example. In *AFIPS '75: May 19-22, 1975, national computer conference and exposition*. ACM, 1975.