A Probabilistic Scheme for Keyword-Based Incremental Query Construction

Elena Demidova, Xuan Zhou, and Wolfgang Nejdl, *Member*, *IEEE Computer Society*

Abstract—Databases enable users to precisely express their informational needs using structured queries. However, database query construction is a laborious and error-prone process, which cannot be performed well by most end users. Keyword search alleviates the usability problem at the price of query expressiveness. As keyword search algorithms do not differentiate between the possible informational needs represented by a keyword query, users may not receive adequate results. This paper presents IQ^P—a novel approach to bridge the gap between usability of keyword search and expressiveness of database queries. IQ^P enables a user to start with an arbitrary keyword query and incrementally refine it into a structured query through an interactive interface. The enabling techniques of IQ^P include: 1) a probabilistic framework for incremental query construction; 2) a probabilistic model to assess the possible informational needs represented by a keyword query; 3) an algorithm to obtain the optimal query construction process. This paper presents the detailed design of IQ^P, and demonstrates its effectiveness and scalability through experiments over real-world data and a user study.

Index Terms—Query formulation, search process.

1 Introduction

Structured queries are a powerful tool to precisely describe a user's informational need and retrieve the intended information from a database. However, manual creation of a structured query is a labor-intensive and error-prone task. This task requires exact knowledge of the database schema as well as proficiency in a query language, which are typically beyond the expertise of end users. Keyword search on the other hand can be performed efficiently by novice users, as it requires neither a-priori schema knowledge nor query construction skills. However, keyword search lacks expressiveness to precisely describe a user's informational need, and may return irrelevant or incomplete results.

To take advantage of both, i.e., expressiveness of structured queries and usability of keyword search, some recent approaches [18], [32], [33], [37] translate a keyword query into a ranked list of structured queries, such that the user can select the query that best represents her informational need. Such a query ranking approach has two limitations. First, as each keyword can potentially occur in any textual attribute of a database, the number of structural interpretations grows sharply with the complexity of the database schema and the length of the keyword query. With a complex database and a lengthy keyword query, it is computationally infeasible to materialize and sort all possible structural query interpretations at runtime. Second, even a theoretically optimal ranking algorithm can, at best, rank the most common query interpretations highest, whereas the users with less frequent informational needs may not receive adequate results. For example, if the majority of users who issued the keyword query "London" were interested in a city guide of London, the results referring to Jack London as a book author will receive a low rank. If a ranking function fails to place the user intended structured query within the top results, the user will need to examine all interpretations prior to the intended one. This process is tedious and error prone.

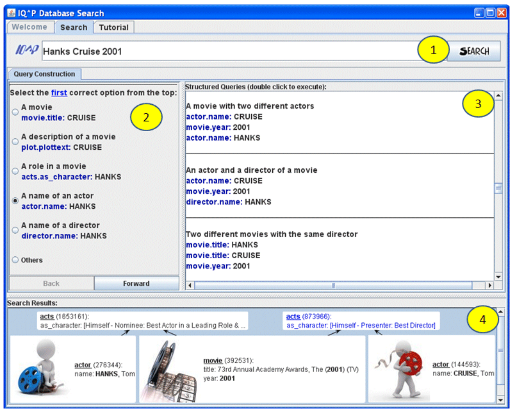
In this paper, we present IQ^P,[1] a novel system that aims at bridging the gap between usability of keyword search and expressiveness of database queries. Using IQ^P, a user can benefit from both, a conventional ranking interface and a more controllable query construction interface. The former allows the user to immediately identify the most common interpretation of her query. The latter enables the user to clarify her search intent step by step, which is especially helpful when the intended query interpretation does not receive a good rank. For instance, if a user issues a keyword query "London Age," IQ^P would ask a number of questions, such as "Is London a person?" or "Are you looking for a city's age?." Based on the user's responses, IQ^P is able to accurately identify the structured query representing the user's informational need. Using IQ^P, users are able to construct structured queries efficiently, without necessarily knowing the database schema or mastering a query language.

Our IQ^P system consists of three components: 1) a framework that formally defines the process of incremental query construction, which does not require any a-priori schema knowledge or proficiency in a query language; 2) a probabilistic model to estimate the probabilities of structural query interpretations, so as to identify meaningful items for users to interact with; 3) an algorithm for generating the

• *E. Demidova and W. Nejdl are with the L3S Research Center and Leibniz Universität Hannover, Forschungszentrum L3S, Appelstraße 9a, Hannover 30167, Germany. E-mail: {demidova, nejdl}@L3S.de.*
• *X. Zhou is with the DEKE Lab, Renmin University of China, Beijing 100872, China. E-mail: xuan.zhou.mail@gmail.com.*

Manuscript received 11 Nov. 2009; revised 20 May 2010; accepted 27 Aug. 2010; published online 7 Feb. 2011.
Recommended for acceptance by J. Haritsa.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2009-11-0773.
Digital Object Identifier no. 10.1109/TKDE.2011.40.

1. http://iqp.l3s.uni-hannover.de.



Hanks Cruise 2001

Select the first correct option from the top:

- ○ A movie movie.title: CRUISE
- ○ A description of a movie plot.plottext: CRUISE
- ○ A role in a movie acts.as_character: HANKS
- ● A name of an actor actor.name: HANKS
- ○ A name of a director director.name: HANKS
- ○ Others

Structured Queries (double click to execute):

A movie with two different actors
actor.name: CRUISE
movie.year: 2001
actor.name: HANKS

An actor and a director of a movie
actor.name: CRUISE
movie.year: 2001
director.name: HANKS

Two different movies with the same director
movie.title: HANKS
movie.title: CRUISE
movie.year: 2001

Search Results:

acts (1653161): as_character: [Himself - Nominee: Best Actor in a Leading Role & ...

acts (873966): as_character: [Himself - Presenter: Best Director]

actor (276344): name: HANKS, Tom

movie (392531): title: 73rd Annual Academy Awards, The (2001) (TV) year: 2001

actor (144593): name: CRUISE, Tom

Fig. 1. IQ^P User interface.[2]

optimal query construction plan (QCP), which enables a user to obtain the intended structured query with a minimal number of interactions. In this paper, we present the detailed design of these components. We also show the effectiveness of our system through a user study and extensive experiments using real-world data sets.

In our preliminary paper [10], we did not address the scalability issue of IQ^P. In this paper, we address the scalability problem by utilizing the concept of query hierarchy, which enables IQ^P to incrementally materialize the query interpretation space. We performed simulations to demonstrate the scalability of the resulting system.

The rest of the paper is organized as follows: Section 2 gives an overview of the IQ^P system. Section 3 presents the conceptual framework for incremental query construction. Section 4 introduces a probabilistic model for assessing the likelihood of a structural query interpretation. Section 5 presents algorithms for generating (near-) optimal query construction plans. Our experiment results are reported in Section 6. Section 7 summarizes related work. Finally, Section 8 provides a conclusion.

2 Overview of IQ^P

IQ^P query construction interface is designed to enhance the ranking centric approaches to database keyword search, such as [18], [32], [33], [37], by giving users control over the query disambiguation process. Fig. 1 illustrates the user interface of IQ^P.

The user interface of IQ^P consists of

1. A search field to input keyword queries,
2. A query construction window to present query construction options,
3. A query window listing structured queries, and
4. A result window for presenting search results.

When a user issues a keyword query, IQ^P provides the user with a ranked list of structured queries (as interpretations of the keyword query) and the corresponding results, which are presented in the query and result windows, respectively. If the user identifies the desired structured query, she can double click the query to retrieve its results. If the user cannot immediately find the desired structured queries or results, she can resort to the query construction window, which enables the user to refine the space of structured queries.

For instance, to find the movies acted by Tom Hanks and Tom Cruise in 2001, Alice issues a keyword query "Hanks Cruise 2001" to IQ^P. Without knowing the exact informational need of Alice, IQ^P translates the keyword query into a number of structured queries, which give different interpretations to Alice's keywords. For example, one possible structured query searches for the movies of 2001 acted by Hanks and entitled "Cruise," whereas another query searches for the movies acted by Cruise and directed by Hanks. These possible interpretations and corresponding results are ranked and presented in the query and result windows, respectively (Fig. 1 (3) and (4)). Simultaneously, IQ^P generates a set of query construction options, and presents these options in the query construction window (Fig. 1 (2)).

If the query interpretation desired by Alice is shown in the query window, she can double click the interpretation and obtain the results. In case Alice cannot immediately identify the desired information in the query or result windows, she can use the query construction interface to refine the presented list of queries. For instance, Alice can select a query construction option specifying that "Hanks" is an actor's name. Whenever Alice selects an option, the query and result windows zoom into the structured queries and results satisfying the selected options. At the same time, the set of the query construction options is updated to enable Alice to further clarify her informational need. Interaction between Alice and the system continues, until the desired structured query or results show up. In fact, the query construction options shown in Fig. 1 are structured queries too, though they interpret only a part of Alice's keyword query.

As the number of possible structural interpretations of a keyword query grows sharply with the number of keywords and the size of the database schema, ranking alone is not always sufficient to help users identify desired structured queries or results. While some natural language processing techniques, such as phrasing and tokenization, can be used to restrict the interpretation space, they have limited usage. For instance, although the keywords "Tom" and "Hanks" can be typically interpreted as the name of a single actor, a user may use these keywords to refer to a movie with two different actors named "Tom Cruise" and "Colin Hanks." In contrast, a set of properly selected query construction options enables a user to exponentially reduce the interpretation space and thus obtain the intended structured query quickly. For example, "Hanks" can be either a part of a movie title or an actor's name in a movie database. By selecting the "Hanks is an actor's name" option, Alice implicitly eliminates all structured queries that interpret "Hanks" to a movie title.

IQ^P can present structured queries either in a natural language (using, e.g., techniques such as [19]) or as graphs similar to that of visual query builders. To generate user understandable options, IQ^P requires appropriate naming of the schema elements in a database.

2. Fig. 1 includes the following images: http://de.fotolia.com/id/25660633© Anatoly Maslennikov - Fotolia.com; http://de.fotolia.com/id/10056489© ioannis kounadeas - Fotolia.com; http://de.fotolia.com/id/9974098© XYZproject - Fotolia.com.

In the following, we present the detailed techniques for enabling the above interface.

3 Query Construction Framework

This section presents a conceptual framework for incremental query construction. First, we show how IQ^P translates a keyword query to a set of structured queries of a relational database. Then, we introduce the concept of query construction plan, which can guide a user through an interactive process to obtain the intended query.

3.1 From Keywords to Structured Queries

We first define the concepts of keyword queries and structured queries.

Definition 1 (Keyword query). *A keyword query $K = \{k_1, \ldots, k_m\}$ is a bag of keywords, where duplicates are allowed.*

Some examples of keyword queries are $K_1 =$ "titanic 1997," $K_2 =$ "actor tom hanks," $K_3 =$ "movie hanks 2001," and $K_4 =$ "number of movies with tom hanks."

Definition 2 (Structured query). *A structured query Q in IQ^P is an expression of relational algebra which is composed of tables and the following operators and predicates:*

Operator. The operators for forming structured queries include selection (σ) and natural join ($\bowtie$).

Predicate. A predicate is in the form $\{k_1, \ldots, k_m\} \subset A$, indicating that the bag of keywords $\{k_1, \ldots, k_m\}$ is contained in the value of the attribute A.

Some examples of structured queries are as follows:

$\boldsymbol{\sigma}_{\text{titanic}\in\text{title} \wedge 1997\in\text{year}}(\text{Movie})$	Movie with title "Titanic" which is made in 1997
$\boldsymbol{\sigma}_{\{\text{tom, hanks}\}\subset\text{name}}(\text{Actor})$	Actor whose name is Tom Hanks
$\boldsymbol{\sigma}_{\text{hanks}\in\text{name}}(\text{Actor}) \bowtie \text{Acts} \bowtie \boldsymbol{\sigma}_{2001\in\text{year}}(\text{Movie})$	Movies in 2001 that are acted by Hanks

To construct a structured query from a keyword query, IQ^P first interprets each keyword to an element of a structured query, which is called keyword interpretation.

Definition 3 (Keyword interpretation). *A keyword interpretation is denoted by A_i:k_i, which maps a keyword k_i to an element A_i of a structured query. A_i can refer to a table, an operator, an attribute, or a value in a predicate. A_i is also called an interpretation of k_i.*

In IQ^P, a keyword is interpreted to an element of a structured query, if it matches the name or the value of that element. For example, to translate a keyword query $K_2 =$ "actor tom hanks," we can make the following keyword interpretations: *Actor:actor*, $tom \in name : tom$, and $tom \in name : hanks$, where "actor" is interpreted as a table, and "*tom*" and "*hanks*" are interpreted to values in predicates.

A set of keyword interpretations can be connected to form a structured query. For example, based on the keyword interpretations mentioned above, we can build a structured query $\boldsymbol{\sigma}_{\{\text{tom,hanks}\}\subset\text{name(Actor)}}$. We call a mapping from a keyword query to a structured query a *query interpretation*.

Definition 4 (Query interpretation). *Given a keyword query K, we say that a structured query Q is a query interpretation of K, if and only if there is a set of keyword interpretations $\{A_i : k_i\}$, where $A_i \in Q$ and $k_i \in K$, such that 1) each keyword in K is interpreted as at most one element of Q; 2) given a substructure Q' of Q, if after removing Q' the remaining structure of Q is also a structured query, then there is at least one keyword in K that is interpreted as an element of Q'.*

If $\{A_i : k_i\}$ contains the keyword interpretations for all the keywords in K, we call Q a complete interpretation *of K. Otherwise, we call Q a* partial interpretation *of K.*

The first condition in Definition 4 guarantees that each keyword has a unique interpretation w.r.t. the structured query. This reflects the intuition that users typically assign one specific meaning to a keyword. The second condition ensures that a query interpretation does not contain any redundant parts, i.e., no part can be excluded from a query such that remaining query still contains the same number of keywords. This corresponds to the minimality condition introduced in [15], [32]. For instance, $\boldsymbol{\sigma}_{\{\text{tom,hanks}\}\subset\text{title}}$ (Movie) and $\boldsymbol{\sigma}_{\{\text{tom,hanks}\}\subset\text{name}}(\text{Actor})$ are both valid interpretations of $K_1 =$ "tom hanks," and $\boldsymbol{\sigma}_{\{\text{tom,hanks}\}\subset\text{name}}(\text{Actor}) \bowtie \text{Acts} \bowtie \boldsymbol{\sigma}_{2001\in\text{year}}$ (Movie) is a valid interpretation of $K_2 =$ "movie tom hanks 2001." However, $\boldsymbol{\sigma}_{\{\text{tom,hanks}\}\subset\text{name}}(\text{Actor}) \bowtie$ Acts $\bowtie$ Movie is not a valid interpretation of $K_3 =$ "actor tom hanks," as $\bowtie$ Acts $\bowtie$ Movie does not contain any interpretation of a keyword in K_3 and thus violates condition 2 in Definition 4.

In our query construction framework, each complete interpretation of a keyword query can be a possible structured query desired by the user. Each partial interpretation can be used as a query construction option. As the number of possible complete interpretations of a single keyword query is usually large, the objective of query construction is to quickly identify the interpretation desired by the user.

Definition 5 (Interpretation space). *Given a keyword query K, the* interpretation space *of K is the entire set of complete interpretations of K.*

Our current implementation of IQ^P considered only a subset of operators and predicates in relational algebra. Involvement of other operators, such as projection, has been considered in the related work [23]. It is an interesting direction for our future research. Currently, a structured query created by IQ^P returns the values of all referred attributes which corresponds to "SELECT *" in SQL.

3.2 Query Interpretation Generation

As creation of the entire interpretation space of a keyword query is expensive, IQ^P uses a set of precomputed *query templates* to accelerate this process. A query template is a pattern typically used to issue structured queries.

Definition 6 (Query template). *A query template T is a structured query whose predicates do not contain any keywords, but variables.*

For instance, $T_1 = \sigma_{? \in \text{name}}(\text{Actor}) \bowtie \text{Acts} \bowtie \sigma_{? \in \text{year}}(\text{Movie})$ is a frequently used query template, as users often look for a movie acted by a certain actor in a certain year. Other examples of query templates include $\sigma_{? \in \text{name}}(\text{Director}) \bowtie \text{Directs} \bowtie \sigma_{? \in \text{year}}(\text{Movie})$ and $\sigma_{? \in \text{name}}(\text{Actor}_1) \bowtie \text{Acts}_1 \bowtie \text{Movie} \bowtie Acts_2 \bowtie \sigma_{? \in \text{name}}(\text{Actor}_2)$.

A structured query (query interpretation) is a composition of a query template and a set of keyword interpretations. IQ^P uses the following process to create an interpretation of a keyword query K:

1. Find an interpretation for each keyword in K;
2. Pick an available query template T;
3. Combine T and the keyword interpretations into a query interpretation.

For example, given a query $K_3 =$ "movie hanks 2001," we first find a set of keyword interpretations {Movie:movie, $\sigma_{\text{hanks} \in \text{name}}$: hanks, $\sigma_{2001 \in \text{year}}$: 2001} and a query template $T_1 = \sigma_{? \in \text{name}}(\text{Actor}) \bowtie \text{Acts} \bowtie \sigma_{? \in \text{year}}(\text{Movie})$, and then combine them into a structured query $\sigma_{\{\text{tom, hanks}\} \in \text{name}}(\text{Actor}) \bowtie \text{Acts} \bowtie \sigma_{2001 \in \text{year}}(\text{Movie})$. This process can be repeated to generate the entire interpretation space of a keyword query.

As the expressiveness of IQ^P is bounded by the precomputed query templates, it is essential to generate query templates that cover as many informational needs of the users as possible. IQ^P generates query templates using three approaches:

- First, query templates can be automatically generated by exploring possible join paths of the database schema within a predefined length. The methods for automatic template generation can be found in some existing work, e.g., [7], [14], [15]. Using this method, IQ^P can achieve similar expressiveness as the ranking approaches that generate interpretations automatically.
- Second, if a database is used frequently, its query log should contain a large number of structured queries. The common patterns in the query log can be used as query templates.
- Lastly, query templates can also be manually defined or adjusted by a database administrator, based on the intended application of the database.

When creating structured queries, sometimes we may not find an appropriate complete interpretation for a keyword query. If one of the keywords is misspelled or does not exist in the target database, it is excluded from the query construction process. The user can either construct a partial interpretation without this keyword or reconsider the keywords. If the query desired by the user is beyond the expressiveness of IQ^P, the user has to resort to other means, e.g., creating a SQL query manually.

3.3 Subquery Relationship

During query construction, IQ^P utilizes the subquery relationships between the partial and complete query interpretations to quickly reduce the interpretation space of a keyword query.

Definition 7 (Subquery). *Given two structured queries Q and Q', we say that Q' is a subquery of Q (or Q' subsumes Q), iff Q' is a substructure of Q.*

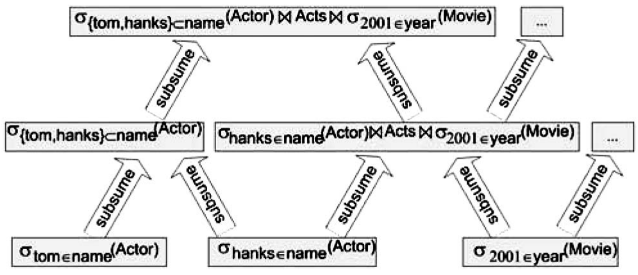


Fig. 2. A Query hierarchy for query "Tom Hanks 2001."

For instance, $Q = \sigma_{\{\text{tom,hanks}\} \subset \text{name}}(\text{Actor}) \bowtie \text{Acts} \bowtie \sigma_{2001 \in \text{year}}(\text{Movie})$ is an interpretation of $K =$ "movie tom hanks 2001." $Q' = \sigma_{\text{hanks} \in \text{name}}(\text{Actor})$ is a subquery of Q, because Q' is a substructure of Q. The subquery relationship is transitive. Namely, if Q' is a subquery of Q and Q'' is a subquery of Q', then Q'' is also a subquery of Q.

In a query construction process, the user is presented with a number of query construction options, i.e., partial interpretations of the keyword query (Fig. 1 (2)). When the user accepts an option, she actually confirms that the option is a subquery of the intended structured query. When the user rejects an option, she indicates that the option is not a subquery of the intended structured query. In either case, she reduces the interpretation space of her keyword query.

Using subquery relationships, we can connect all the complete and partial interpretations of a keyword query to form a **Query Hierarchy**. Fig. 2 shows a part of the query hierarchy for the keyword query $K =$ "Tom Hanks 2001."

The shape of a query hierarchy is like an upside-down triangle. The bottom of a query hierarchy is usually very small, as it contains only the smallest partial query interpretations. The top of a query hierarchy is much larger, as it contains all complete interpretations. IQ^P utilizes the query hierarchy for an incremental generation of complete and partial query interpretations. It first uses smaller query templates to generate partial interpretations at the bottom of the hierarchy, and then gradually expand them to generate more complete interpretations. With a complex database schema or a long keyword query, IQ^P may have to deal with an interpretation space that is prohibitively large. As shown in Section 5, a query hierarchy enables IQ^P to generate appropriate query construction options without materializing the entire interpretation space, so as to ensure the scalability of the system.

3.4 Query Construction Plan

As shown in Fig. 1, in each query construction step of IQ^P, a user is presented with a list of query construction options, i.e., partial interpretations. She is supposed to select the option that correctly interprets her keywords.

To simplify our presentation, we assume that a user decides on only one option at a time. If the option is a subquery of the intended query interpretation, the user accepts it. If the option is not a proper partial interpretation, the user rejects it. After the user accepts or rejects an option, the interpretation space of the keyword query can be reduced accordingly. The user keeps evaluating the options one after another, until only one possible query interpretation is left. Such a query construction process can be

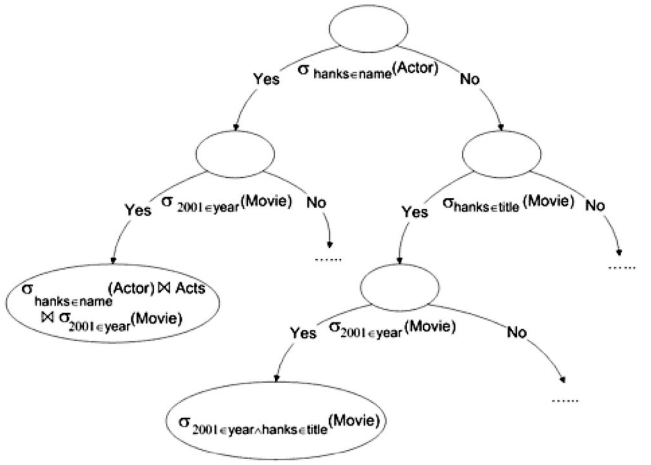


Fig. 3. Query construction plan as a binary tree.

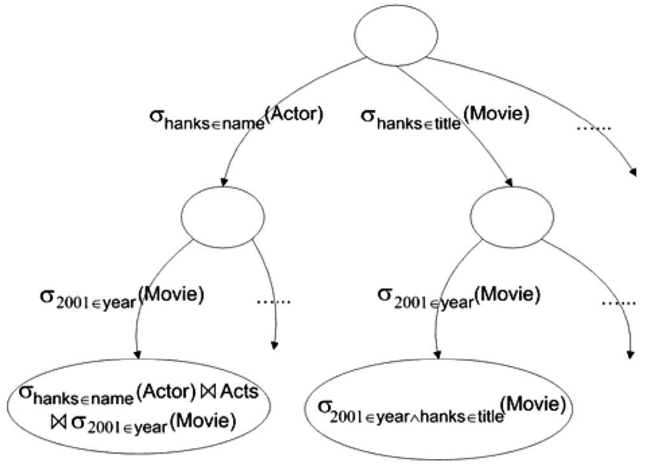

σ hanks∈name(Actor)
σ hanks∈title(Movie)
σ 2001∈year(Movie)
σ 2001∈year(Movie)
σ hanks∈name(Actor) ⋈ Acts ⋈ σ 2001∈year(Movie)
σ 2001∈year∧hanks∈title(Movie)

Fig. 4. Query construction plan as an N-ary tree.

modeled as a binary decision tree, which is called *query construction plan*.

Definition 8 (Query construction plan). *A Query Construction Plan (QCP) is a binary tree. Each node of the tree represents a set of structured queries, i.e., complete query interpretations. The left and right edges of a node represent the acceptance and the rejection of a query construction option, i.e., a partial interpretation, respectively. The tree satisfies: 1) the root represents the entire interpretation space of a keyword query; 2) each leaf node represents a single complete query interpretation; 3) given an edge (Y, X) (Y is X's parent), a) if (Y, X) is an acceptance of a query construction option O, then X is the subset of Y that subsumes O; b) if (Y, X) is a rejection of query construction option O, then X is the subset of Y that does not subsume O.*

A fragment of a query construction plan for the keyword query $K =$ "hanks 2001" is shown in Fig. 3. A query construction process is a traversal of the query construction plan. The user starts from the root of the plan. At each node, the user decides on the query construction option represented by the outgoing edges of that node. After accepting or rejecting this option, the user moves to the node pointed by the corresponding out-edge. The process continues until the user reaches a leaf of the tree, which is the structured query desired by the user.

In the user interface shown in Fig. 1, the user needs to decide on multiple query construction options in each round. That interface is based on an N-ary tree, which is illustrated in Fig. 4. Each edge in the N-ary tree represents a query construction option. In each step, the user is supposed to select the first option that satisfies her intent. It can be proven that the N-ary tree in Fig. 4 can be uniquely transformed from the binary tree in Fig. 3, and vice versa. To make the transformation, we traverse the binary tree in postorder. For each node *NB* in the binary tree, we remove the right edge (*RE*) and the right child (*RC*) of *NB*, and add *RC*'s edges and children to *NB*. To facilitate our presentation, we always refer a query construction plan as a binary tree in the rest of this paper.

For a single keyword query, there typically exist an arbitrary number of *QCP*s, whose efficiency differs significantly. Therefore, the key issue of query construction in IQ^P is to find an optimal *QCP* that enables the user to obtain the intended query interpretation as fast as possible. Each interaction between a user and a *QCP* always follows the same pattern, that is, the user receives a query construction option from *QCP*, evaluates it and chooses to accept or reject it. (In most of the cases, a user does not need to explicitly reject an option. Instead, the user proceeds with evaluating the next one). Therefore, we can measure the efficiency of a *QCP* by an average number of interactions between a user and the *QCP* until the user reaches a complete structured query at a leaf node. We call this measure interaction cost.

Definition 9 (Interaction cost of query construction plan). *We measure the interaction cost of a query construction plan by the* expected *number of query construction options (partial interpretations) a user has to evaluate to reach a structured query (complete query interpretation), i.e., a leaf of the binary tree. Given a structured query Q, the number of subqueries a user needs to evaluate to construct Q is the depth of Q in the query construction plan. Therefore, the interaction cost of the plan can be calculated by*

$$Cost(QCP) = \sum_{leaf \in QCP} depth(leaf) \times P(leaf), \qquad (1)$$

where depth (leaf) is the depth of the leaf in QCP, and P (leaf) is the probability that the leaf is the user intended query interpretation.

Then, the problem of incremental query construction is reduced to the problem of finding the query construction plan with the minimum interaction cost.

Definition 10 (Minimum query construction plan). *A query construction plan QCP is a minimum query construction plan for the keyword query K, iff there is no query construction plan of K whose interaction cost is less than that of QCP.*

3.5 Query Construction versus Ranking

In the ranking centric approaches [18], [32], [33], [37], the structural interpretations of a keyword query are ranked based on their probability of matching the user's intent. The user can then navigate through the ranked list to identify the desired interpretations. Such a ranked list of query interpretations is actually a special case of a *QCP* defined in Definition 8. In this special *QCP*, a user is always presented with options corresponding to complete query interpretations. If the user accepts the option, she gets the desired complete query interpretation directly. If the user rejects the option, she traverses to the next node representing the next complete query interpretation. However, such a *QCP* is not necessarily the optimal one. Using this QCP, a user can reach the top-ranked interpretations quickly, but has to undertake a lot of interactions to find the less probable interpretations.

In fact, a *QCP* for ranking is an unbalanced tree. It works for the cases where the semantics of the keyword query is obvious, such that the majority of the probability is assigned to a small number of query interpretations. It performs poorly in case a keyword query is ambiguous, such that the probability is more evenly distributed over a larger number of interpretations. In contrast, IQ^P always aims to create the optimal *QCP*. When the semantics of a keyword query is obvious, it generates a *QCP* similar to that of ranking. When the keyword query is ambiguous, it generates a more balanced *QCP*.

In Sections 4 and 5, we show how IQ^P generates query construction plans. We address two issues: 1) how to estimate the probability of a query interpretation, and 2) how to generate minimum query construction plans based on the probability estimation.

4 Estimating Query Probability

To support efficient query construction, it is important to have an accurate assessment of the probability of whether a structured query (or a query construction option) interprets a user's keyword query correctly. In this section, we introduce a probabilistic model, which enables IQ^P to compute these probabilities. We also discuss possible statistics for supporting probability assessment.

4.1 A Probabilistic Query Interpretation Model

Given a keyword query, IQ^P is uncertain about the exact informational need represented by this query. We quantify this uncertainty using probability.

Definition 11 (Probability of query interpretation). *Given a keyword query K, let the structured query Q be a complete interpretation of K, i.e., $Q : K$. Then, $\mathrm{P}(Q|K)$ represents the conditional probability that, given K, Q is the user intended complete interpretation of K. Analogously, given a query construction option (a partial interpretation) O of K, $\mathrm{P}(Q|K)$ represents the conditional probability that, given K, O subsumes the user intended complete interpretation of K.*

$\mathrm{P}(Q|K)$ corresponds to P (*leaf*) in (1). It is a crucial parameter used by IQ^P in creating query construction plans. If a keyword query K has been used repeatedly in a database, we can directly estimate $\mathrm{P}(Q|K)$ using the previous interpretations of K in a database's query log. However, in a large database, it is unlikely to find sufficient number of records for a particular keyword query. To compute $\mathrm{P}(Q|K)$, we need to resort to other statistics. In the following, we decompose $\mathrm{P}(Q|K)$ to a set of atomic probabilities which are much easier to obtain.

A query interpretation Q is composed of a set of keyword interpretations $\{A_i : k_i\}$ and the query template T of Q. Thus, the probability $P(Q|K)$ can be transformed to

$$P(Q|K) = P(\{A_i : k_i\}, T|K). \tag{2}$$

To decompose this probability, we make a number of assumptions in our probabilistic model.

Assumption 1 (Keyword independence). *First, we assume that the interpretation of each keyword in a keyword query is independent from the other keywords.*

The assumption of keyword independence is used to simplify the probability calculation, similarly to many other models in the literature (e.g., Vector Space Model in Information Retrieval and Naïve Bayes in Machine Learning [25]). Although the resulted probability estimation may not be very precise, our experiments in Section 6 show that this model provides an adequate prediction of the query relevance and significantly reduces the interaction cost. Based on Assumption 1 as well as Bayes' rule, we can transform (2) to

$$\begin{aligned} P(Q|K) &= \frac{P(\{A_i : k_i\}|T) \times P(T) \times P(K|\{A_i : k_i\}, T)}{P(K)} \\ &= \frac{\left(\prod_{k_i \in K} P(A_i : k_i|T)\right) \times P(T) \times P(K|\{A_i : k_i\}, T)}{P(K)}. \end{aligned} \tag{3}$$

As the set of keywords K can be known from the set of keyword interpretations $\{A_i : k_i\}$, we have $\mathrm{P}(\mathrm{K}|\{A_i : k_i\}, \mathrm{T}) = 1$. Therefore, we can transform (3) to

$$P(Q|K) = \frac{\left(\prod_{k_i \in K} P(A_i : k_i|T)\right) \times P(T)}{P(K)}. \tag{4}$$

In this formula, $\mathrm{P}(T)$ is the prior probability that the template T is used to form a query interpretation. $\mathrm{P}(A_i : k_i|T)$ represents the probability that, given that T is used to form a query interpretation, A_i is used to form the query interpretation too. *P(K)* is the prior probability that a user issues the keyword query K.

Assumption 2 (Keyword interpretation independence). *Second, we assume that the probability of a keyword interpretation is independent from the part of the query interpretation the keyword is not interpreted to. In other words, $\mathrm{P}(A_i : k_i|T) = \mathrm{P}(A_i : k_i|T \cap A_i)$, where $\mathrm{P}(A_i : k_i|T \cap A_i)$ represents the probability that, given that $T \cap A_i$ is a part of a query interpretation, A_i is also a part of the query interpretation.*

For instance, let $\mathrm{T} = \boldsymbol{\sigma}_{? \in \mathrm{name}}(\mathrm{Actor}) \bowtie \mathrm{Acts} \bowtie \boldsymbol{\sigma}_{? \in \mathrm{year}}(\mathrm{Movie})$.

Let $A_1 : k_1 = \boldsymbol{\sigma}_{\mathrm{hanks} \in \mathrm{name}}(\mathrm{Actor})$:*hanks*. Thus,

$$\begin{aligned} \mathrm{P}(A_1 : k_1|\mathrm{T}) &= \mathrm{P}(A_1 : k_1|T \cap A_1) \\ &= \mathrm{P}(\boldsymbol{\sigma}_{\mathrm{hanks} \in \mathrm{name}}(\mathrm{Actor}) : hanks|\boldsymbol{\sigma}_{? \in \mathrm{name}}(\mathrm{Actor})), \end{aligned}$$

which represents the probability that, given that the attribute *Actor.name* is a part of the query interpretation, $\sigma_{\text{hanks}\in\text{name}}(\text{Actor})$ is also a part of the query interpretation. We assume that the probability of $\sigma_{\text{hanks}\in\text{name}}(\text{Actor})$:*hanks* only depends on the fragment of $\sigma_{?\in\text{name}}(\text{Actor})$ of the template.

Furthermore, as we consider different interpretations of the same keyword query K, we can skip computing P(K), which is a constant for all query interpretations. Finally, we have

$$P(Q|K) \propto \left(\prod_{k_i\in K} P(A_i : k_i|T\cap A_i)\right) \times P(T). \qquad (5)$$

The probability of a partial interpretation O, i.e., $P(O|K)$, can be computed similarly as (5). Suppose that O only contains the keyword interpretations of $K' \subset$ K. Then,

$$P(O|K) \propto \left(\prod_{k_i\in K'} P(A_i : k_i|T\cap A_i)\right) \times P(T). \qquad (6)$$

4.2 Probability Estimation

According to (5) and (6), the calculation of the probability of a query interpretation requires the estimation of P(T), the prior probability of the query template T, as well as P($A_i : k_i|T\cap A_i$), the probability that, given that $T\cap A_i$ is a part of a query interpretation, A_i is also a part of the query interpretation.

Estimating probability of a template. If the database possesses a query log that is statistically representative, P(T) can be estimated directly using the log. Based on the maximum likelihood model, P(T) can be calculated as the frequency of T in the query log. Namely,

$$P(T) = \frac{\#occurrences(T) + \alpha}{N}, \qquad (7)$$

where *#occurences(T)* is the number of queries using T as a template, N is the total number of queries, and α is a smoothing parameter, which is typically set to 1. When the query log is absent or is not sufficient, we assume that all query templates are equally probable.

Estimating probability of a keyword interpretation. In this paper, we focus on two types of keyword interpretations defined in Definition 3. The first type interprets a keyword as part of a query template, such as a table name, an attribute name, or an operator name. The second type maps a keyword to a "contains" predicate, interpreting the keyword as a value of an attribute. This interpretation has the form $\sigma_{ki\in\text{ATi}}$(Table):k_i.

For the first type of interpretation, we can estimate the probability P($A_i : k_i|T\cap A_i$) using the query log too. If A_i is a table name, P($A_i : k_i|T\cap A_i$) is the frequency of using k_i to represent table A_i among the existing query interpretations containing table A_i. If A_i is an operator name, P($A_i : k_i|T\cap A_i$) is the frequency of using k_i to represent operator A_i among the existing query interpretations containing operator A_i. Without a query log, our system can use some empirical values set by domain experts.

In case, a keyword is interpreted as an attribute value, we can hardly estimate the probability of the keyword interpretation using a query log. This is because the number of occurrences of each particular attribute value in a query log is usually insignificant. Therefore, we estimate probability of this interpretation using statistics obtained from the database instances. We model the formation of a query interpretation as a random process. For an attribute AT_i, this process randomly picks one of its instances a_j and randomly picks a keyword k_i from that instance to form the expression $\sigma_{ki\in\text{ATi}}$(Table). Then, the probability of P($\sigma_{ki\in\text{ATi}}$(Table) : $k_i|\sigma_{?\in\text{ATi}}$(Table)) is the probability that $\sigma_{ki\in\text{ATi}}$(Table) is formed through this random process. Based on maximum likelihood mode, this probability can be estimated using Attribute Term Frequency (ATF), which is defined as

$$\begin{aligned} ATF(k_i, AT_i) &= P(\sigma_{k_i\in AT_i}(Table) : k_i|\sigma_{?\in AT_i}(Table)) \\ &= TF(k_i, AT_i) + \alpha. \end{aligned} \qquad (8)$$

In (8), TF(k_i, AT_j) is the normalized frequency of keyword k_i in the attribute AT_i and α is a smoothing parameter, which is typically set to 1. The concept of *TF* closely corresponds to the Term Frequency used in Information Retrieval if we treat each attribute instance as a document [25]. ATF is similar to the AF factor introduced in [27], which describes how typical the term is in the values of the respective attribute.

While some other probabilistic models and statistics can also be used to estimate P(Q|K), we show through experiments that our approach is highly effective.

5 Query Construction Algorithm

As mentioned in Section 3, given a keyword query, there typically exist multiple possible query construction plans. While every plan can allow a user to obtain the intended structured query, these plans can differ in efficiency significantly. A less efficient plan also means reduced usability of IQ^P. To find an intended interpretation, a user needs to traverse a branch of the *QCP* (as a binary tree). This requires the *QCP* to be balanced. As mentioned earlier, a special case of an unbalanced *QCP* is a simple ranked list of all possible query interpretations [18], [32], [33], [37]. In case, the intended interpretation does not appear within the top-ranked items, the user has to examine all queries prior to the intended one, which is tedious and error prone. Instead, a more balanced *QCP* tree can efficiently prune the search space and enable the user to obtain a desired query in fewer steps. In this section, we propose an algorithm to create a plan that imposes as little effort on the user as possible, i.e., a minimum query construction plan.

As a brute-force algorithm to generate the exact minimum *QCP* is costly, IQ^P uses a greedy algorithm to construct a near-optimal *QCP*. Instead of generating the entire plan, the algorithm generates query construction options one by one. Whenever an option is generated, it is presented to the user. After the user evaluates the option, (she either accepts it or rejects it,) it proceeds to generate the next option. The process is similar to that of the ID3 algorithm [30] in creating decision trees.

We present the pseudocode of the greedy algorithm in Algorithm 1. As mentioned in Section 3.2, IQ^P generates query interpretations by expanding the query hierarchy in a bottom-up fashion. Instead of fully expanding the query hierarchy, the greedy algorithm stops when the size of the top level of the query hierarchy reaches a certain threshold (denoted by T). Then, it searches for the best query construction option (denoted by *best_r*) within the current query hierarchy and presents the option to the user. If the user accepts the option, the algorithm keeps the part of the top level subsumed by this option and discards the rest. If the user rejects an option, the algorithm discards the part of the top level subsumed by this option. In either case, the algorithm would reduce the size of the top level of the current query hierarchy. The algorithm continues presenting query construction options to the user, until the size of the top level falls below the threshold T. When the threshold is reached, the algorithm expands the top level of the query hierarchy again to get a new level of query interpretations. This process continues until the algorithm reaches the level of complete query interpretations and the user identifies the final intended structured query.

Algorithm 1. A Greedy Algorithm for MQCP

```
Proc greedy_tree(HQ, TQ, T)
Input:
  HQ := Initial Query Hierarchy;
  TQ := Top Level of HQ;
  T := Threshold;
Output:
  Query C := Final Structured Query;
Program:
  while (true)
    if |TQ| < T, then
      if HQ can be expanded, then
        expand HQ;
      else if |TQ| = 1, then //let TQ = {c}
        return c;
      end if;
    end if;
    partial_query best_r := null;
    float best_gain := +∞;
    for each R in HQ, do
      if IG(TQ|R) < best_gain, then
        best_gain := IG(TQ|R);
        best_r := R;
      end if;
    end for;
    present best_r to user;
    if best_r is accepted, then
      Sub(best_r) := all queries subsumed by
      best_r;
      TQ := TQ ∧ Sub(best_r);
    else if best_r is rejected, then
      Sub(best_r) := all queries subsumed by
      best_r;
      TQ := TQ − Sub(best_r);
    end if;
  end while;
End Proc;
```

As the query hierarchy is selectively expanded, we avoid generation of all possible query interpretations. In fact, the number of query interpretations generated by the algorithm is only proportional to the number of options the user needs to evaluate, i.e., the interaction cost of the query construction plan.

Without fully expanding the query hierarchy, it is not possible to find the theoretically optimal query construction option. However, the partially expanded hierarchy is good enough to provide some near-optimal options. Our greedy algorithm tries to find a query construction option that can reveal as much information as possible about the intended structured query. We use *Information Gain* (denoted by $IG(TQ|R)$) to measure the amount of information that can be revealed by a query construction option. In our greedy algorithm, the query construction option that maximizes this information gain is selected and presented to the user. The information gain can be readily calculated using the probabilities of the structured queries in the query hierarchy TQ, by following the conventional information gain definition [30]. The probability estimations of the structured queries and the query construction options are obtained using the methods introduced in Section 4.

6 Evaluation

To evaluate our query construction approach, we performed extensive experiments. First, we assessed the efficiency of IQ^P in helping users to construct structured queries on two real-world databases. Then, we compared the usability of incremental query construction with that of the ranking approach. Finally, we carried out simulations to study the scalability of IQ^P with respect to the size of the database and the length of keyword queries.

6.1 Data Sets and Keyword Queries

In our experiments, we used two real-world data sets: a crawl of the Internet Movie Database (IMDB) and a crawl of a lyrics database from the web. The IMDB data set contains seven tables, such as movies, actors, and directors, with more than 10,000,000 records. The Lyrics data set [22] contains five tables, such as artists, albums, and songs, with around 400,000 records. We deployed IQ^P on both data sets. To generate templates, we used the automatic approach presented in Section 3.2. We set the maximal length of the join path to four, and obtained 74 templates for IMDB and 16 templates for Lyrics.

As these data sets do not have any associated query logs, we extracted the keyword queries from the query log of a major web search engine [29]. We pruned the queries based on their target URLs (www.imdb.com or any domain containing keyword "lyrics" in the URL), and obtained a few thousands of keyword queries targeted on the IMDB and lyrics domains. To assess the capability of IQ^P in creating complex structured queries, we further restricted the query set to the queries containing at least two attributes, such as movie-actor and artist-lyrics. This finally gave us 108 queries for IMDB and 76 queries for Lyrics. Each of these queries contains two-six terms, with an average length of four terms. For each of these queries, we manually assessed its meaning and constructed the corresponding structured queries, which we used as ground truth for our evaluation.

For our user study, we manually identified a set of keyword queries targeted at the IMDB data set. Some of the queries are ambiguous, such that none of the query ranking algorithms we tested produced acceptable results. We present these queries in Section 6.4.

We installed the data sets on a MySQL 5.0.22 database server, running on dual Xeon server with 4 GB RAM. The inverted index was constructed using Lucene [26]. Our IQ^P system was implemented using JDK 1.5 and installed on a laptop with a 2.0 GHz C2D and 2 GB RAM.

6.2 Effectiveness of the Probability Estimates

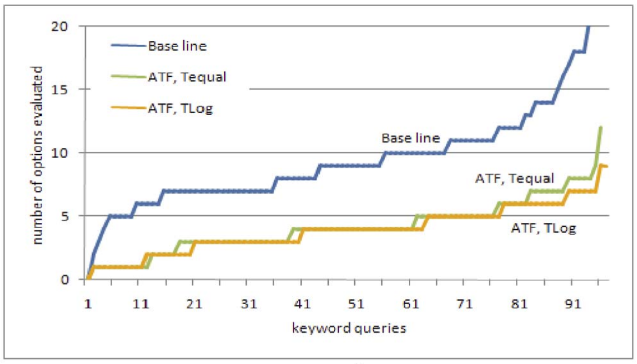
To assess how fast IQ^P can enable a user to construct a structured query, we measured the interaction cost of query construction, that is, the number of query construction options a user needs to evaluate to obtain the intended structured query. Our experiments were performed in an automatic way. We applied the greedy algorithm introduced in Section 5 to each of the keyword queries. The algorithm generates query construction options one by one. Based on the ground truth interpretations established a-priori, we let our system automatically accept the correct options and reject the incorrect options. The process of query construction stops when less than five complete query interpretations are left in the query window, in which the user is able to quickly identify the intended query. At the end of each construction process, we record the number of query construction options that have been evaluated.

To estimate the effectiveness of the proposed query probabilistic model, we used three variations of probability estimates. The first variation, also called base line, assumes that all structured queries and query construction options are equally likely. The second variation, referred as (ATF, Tequal), applied the probabilistic model introduced in Section 4.1 (represented by (5) and (6)). It uses the Attribute Term Frequency to estimate $\mathrm{P}(A_i : k_i | T \cap A_i)$, but assumes equal probabilities of query templates. The third version, represented by (ATF, TLog), not only used ATF to estimate $\mathrm{P}(A_i : k_i | T \cap A_i)$, but also used the query log to estimate the probabilities of the query templates.

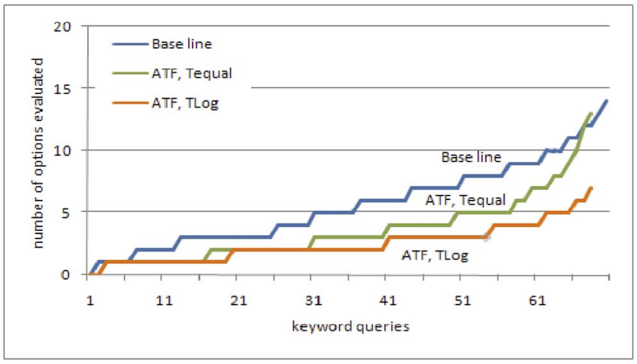
The experiment results for IMDB and Lyrics are shown in Figs. 5a and 5b, respectively. In both figures, each data point on the X-axis represents a keyword query. Each Y-value presents an interaction cost, which is the number of query construction options a user needs to evaluate until she identifies the intended structured query. In this context, "evaluate" means to decide whether an option correctly interprets the user's intent.

As shown in Fig. 5a, for the IMDB data set, using the base line probability estimate, a user needs to evaluate 1 to 20 query construction options to construct a structured query. In more than 50 percent of the cases, the interaction cost is below 10. In more than 80 percent of the cases, the interaction cost is below 15. Occasionally, the interaction cost can reach 20. By estimating the probabilities of structured queries, the interaction cost can be significantly reduced. As shown by the lines of (ATF, Tequal) and (ATF, TLog), in more than 70 percent of the cases, a user needs to evaluate at most five options to create a structured query. In most of the cases, the interaction cost falls below 10.

A similar trend can be seen in the results of the Lyrics data (Fig. 5b). Using the baseline probability estimation, the interaction cost ranges between 0 and 15. By applying our probabilistic model, especially by using the probability estimation of (ATF, TLog), the interaction cost is reduced by around 50 percent.



a



b

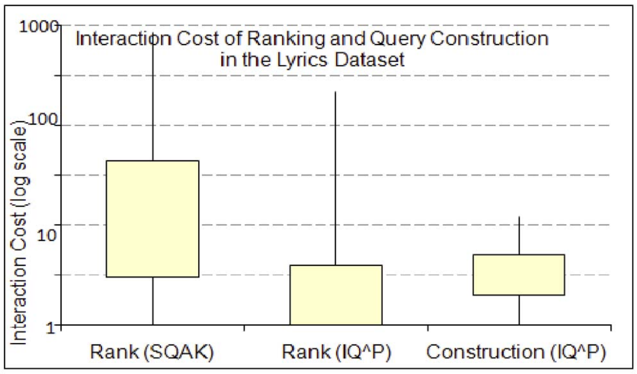
Fig. 5. (a) Interaction cost in number of options, IMDB. (b) Interaction cost in number of options, Lyrics.

Both Figs. 5a and 5b show that Attribute Term Frequency is a highly effective statistics for estimating probabilities of query interpretations. It helped IQ^P reduce users' interaction costs significantly. The probability estimation of query templates based on usage is also useful. However, its effectiveness differed in the two data sets. In the Lyrics data set, we observed a more significant improvement when considering the usage statistics of a template. This is because some query templates of Lyrics are used much more often than the others. For example, the most commonly used query template in Lyrics, which has the frequency of 0.85, is composed of five tables: Song->AlbumSong->Album->ArtistAlbum->Artist. In contrast, the usage of the IMDB query templates is more uniformly distributed. As a result, probabilities of query templates contribute less to the optimization of query construction process on this data set.

6.3 Query Construction versus Query Ranking

Our second set of experiments aimed to compare the interaction cost of query construction against that of query ranking. We used two query ranking functions: the ranking function of IQ^P and SQAK [32], one of the most recent query ranking functions in the related work. Note that query ranking approaches use different statistics from that of result ranking approaches like [14], [22], as the ranking is conducted without materializing query results.

We measure the interaction cost of query ranking as the rank of the intended query interpretation in the ordered



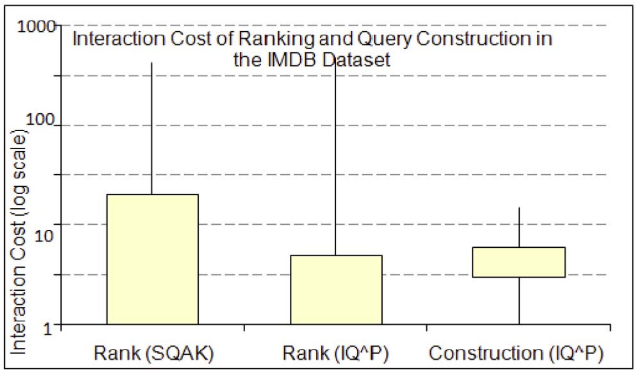


Fig. 6. Interaction cost of query ranking and query construction using SQAK and IQ^P on IMDB and Lyrics data sets.

query list. This reflects the fact that the user has to evaluate each interpretation in the ranked list until she finds the intended interpretation. In case of IQ^P query construction, we measure interaction cost as the number of options the user has to evaluate until she arrives at the intended query. In this context, "evaluate" means to decide whether an option suggested by the system correctly interprets the user's intent. For the IQ^P ranking and incremental query construction, we considered all templates to be equally probable (ATF, Tequal), to reflect the situation in which a query log is not available.

In SQAK, a query interpretation is regarded as a graph, whose score is the sum of the scores of its nodes and edges. Edges and nodes which do not contain keyword are assigned with unit scores. The score of a node containing a keyword is computed as TF-IDF of the keyword, which is then normalized using Lucene scoring function [26]. The original SQAK function does not consider the case where multiple keywords are contained in a single node. We used the score of the Boolean AND query of Lucene to assess the score of the nodes containing more than one keyword.

Fig. 6 shows the boxplots [8] of the interaction costs of ranking by SQAK and IQ^P as well as the interaction cost of IQ^P query construction. The Y-Axis of Fig. 6 (log scale) represents interaction cost, that is, the number of structured queries or query construction options to be evaluated by the user before the desired query can be identified. The box boundaries correspond to the upper and lower quartile of the data points, such that 50 percent of the data points lay inside of the boxes. "Rank (SQAK)" and "Rank (IQ^P)" represent the rank of the intended query using the corresponding ranking function. "Construction (IQ^P)" represents the number of options which needs to be evaluated to obtain the intended query using incremental query construction of IQ^P.

As Fig. 6 shows, our ranking function performs very well for the majority of the queries in the both data sets. The median value of Rank (IQ^P) is two for both data sets. In these cases, construction is not necessary as the user can find the desired interpretation immediately within the top results. However, ranking function has a high variance, such that interpretations of ambiguous queries can receive ranks above 400. If the intended query interpretation does not receive a good rank, the user has to scan through the entire query list, which can contain 3,500 queries for our test set, until she identifies the intended structured query. The process is tedious and error prone, as the user does not even know whether the intended query interpretation exists for the target database or if she occasionally missed the interpretation already.

In contrast, the interaction cost of incremental query construction has a much lower variance than the cost of ranking. As Fig. 6 shows, the cost of construction is around 3-4 options on average and 15 options in the worst case, which makes it highly helpful when user intended structured queries cannot be found within the top-ranked results. Using the construction interface, a user can reach any structured query in the interpretation space in a reasonable number of steps. In addition, when using IQ^P, the user does not have to rely on the incremental query construction only. As IQ^P simultaneously presents a refined ranked list of queries in the query window, a user can shortcut the construction plan if the correct query is already presented in the top items (Fig 1. (3)). To estimate the scope in which construction outperforms ranking in real life, we performed a user study described in Section 6.4.

The experiments have shown that incremental construction is especially helpful when users' keyword queries are ambiguous. For example, in the IMDB query set, the intended interpretations of keyword queries containing more than one person name (e.g., an actor, a director, or a character in a movie like "Melissa Gilbert Bruce Boxleitner") usually do not receive good ranks.

Fig. 6 shows that IQ^P's query ranking outperforms SQAK's query ranking on our test set. The median interaction cost of IQ^P is two on both data sets, whereas the median cost of SQAK is six on IMDB and 13.5 on Lyrics.

It appears that the attribute term frequency used by IQ^P is more effective for query ranking than the TF-IDF score used by SQAK. Intuitively, ATF prefers more typical interpretations, and TF-IDF prefers more distinctive interpretations. For our query set, typical interpretations are used more often. For example, "Garcia" is usually interpreted as an actor name, e.g., Andy Garcia. By using TF-IDF, it will be interpreted as movie title, as "Garcia" occurs less frequently in the movie title than in the actor name. Furthermore, for the Lyrics data set, Steiner tree minimization used by SQAK does not provide good results for many queries. This is because the majority of Lyrics queries, such as "mariah carey emotions," requires a relatively long join Artist-ArtistAlbum-Album-AlbumSong-Song, whereas Steiner tree minimization prefers shorter joins such as Artist-ArtistAlbum-Album. Therefore, we used IQ^P ranking for our user study. In our future work, we plan to perform a more detailed comparison between different ranking approaches.

TABLE 1
Example Tasks for the User Study

Task	C_1*	C_2*	|I|
Find a role of Brad Pitt in the movie directed by Steven Soderbergh in 2004.	18	6	3365
Find an actor who played in both movies: Frida and Three Sisters.	48	6	268
Find a movie in which both Tom Hanks and Diego Luna acted.	73	7	1550
Find the role of an actor in the movie Be Cool. The same actor played a character called Sam Baily in another movie.	104	10	1470
Find a movie where Blake Blue is a director and Conners Chad is an actor.	213	7	1648

*C_1: rank of the intended query interpretation in the ordered list of possible query interpretations while using IQ^P ranking; C_2: an approximate number of options to be evaluated by the user in the query construction process; |I|: size of the interpretation space.

6.4 Usability of Query Construction

To assess usability of IQ^P in real life, we conducted a user study. The user study aimed to compare usability of two user interfaces and to assess in which cases the IQ^P interface can enable more efficient data access than the query ranking interface. One interface is the IQ^P interface shown in Fig. 1. The other is a query ranking interface without using the query construction panel. For the query ranking, we used the IQ^P ranking function with the probability estimate (ATF, TEqual). For user convenience, the query ranking interface presented structured queries in several pages, each containing 20 queries.

Our users were 15 graduate students from the Computer Sciences Department. We selected 14 tasks for the users to perform. Each task required the user to retrieve certain information from the IMDB data set. For each of the tasks, we proposed a keyword query. Based on the rank of the correct query interpretation, which ranged from 0 to 220, we grouped the tasks into seven complexity categories: 0, 1, 2, 3, 4, 6, and 11. Each category contained two tasks. Category k means that the correct query interpretation appears in the kth page of the ranking interface. (0 means the correct interpretation is within the top-10.) Table 1 provides an overview over several example tasks.

We announced the user study as a time-based competition, such that our participants would solve the tasks as quickly as possible. The tasks were given to the participants in a random order. For the two tasks in each complexity category, each participant had to solve one using the IQ^P query construction interface and the other using the query ranking interface. Before the study, we provided the participants with a tutorial and some example tasks to practice, such that they could become familiar with the interfaces. To measure the time spent on each task, we recorded the interval between the time when a user clicked on the Search button and the time when the user executed (double clicked) the correct query interpretation. In case, the participant did not manage to complete the task we set the time for this task to 10 minutes. Fig. 7 presents the median time used for different categories of tasks with both interfaces.

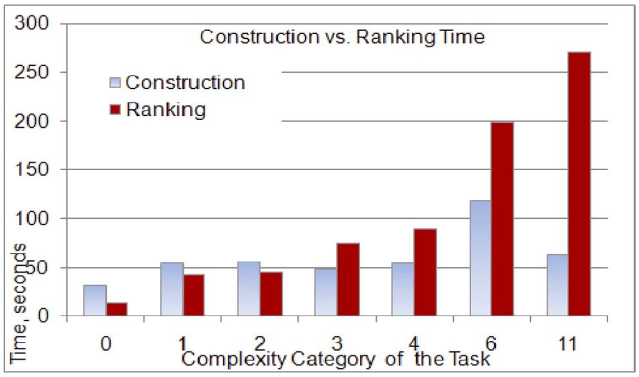


Fig. 7. Median interaction time for construction versus ranking.

Each data point on the X-axis of Fig. 7 represents complexity of the task. The Y-Axis of Fig. 7 represents the median time in seconds required for the user to complete a task. "Ranking" represents the results of the query ranking interface; "Construction" represents the results of the IQ^P construction interface. For example, for a task with complexity 11, the median time spent by a participant using construction was 63 seconds. In contrast, ranking interface in the same complexity category required 270 seconds, which is 4.3 times more.

As we can see from Fig. 7, the ranking interface outperforms the construction interface in the first three categories, where the ranks of the intended query interpretations are below 40. For the queries in Categories 3 and 4, where the ranks of the intended query interpretations range between 40 to 80, the IQ^P interface started to outperform the query ranking interface. For the queries in Category 6 and above, where the ranks of the intended query interpretations are above 120, the advantage of the IQ^P interface becomes very obvious.

We also asked the users to rate the percentage of queries for which they found the IQ^P interface to be more useful than the simple ranking interface on a 5-point Likert scale. Their answers were 70 percent of queries on average. Several participants pointed out, that they perceived time savings while using IQ^P as they proceed with tasks. We attribute this to increased familiarity with the interface.

6.5 Scalability

Given a keyword query, a keyword can occur in any textual attribute of a database, such that the number of possible query interpretations is polynomial in the size of the database schema and exponential in the number of keywords. To test the scalability of the query construction plan generation algorithm, we conducted a set of simulations.

In our simulations, we generated a database schema as a completely connected graph of a given size, where each node represents a relational table. Based on the schema graph, we generated a set of query templates, where each template is a randomly picked connected subgraph of the schema. Given a randomly generated keyword query, we assumed that each keyword occurs in a table with a certain probability (60 percent in our experiments). By combining the occurrences of the generated keywords and query templates, we obtained a number of possible structured queries. We assigned random probabilities to each table and

TABLE 2
Greedy Algorithm versus DB Size

# of tables	# of queries	Thresholds					
		10		20		30	
		#steps	time/step	#steps	time/step	#steps	time/step
5	28	4	1ms	3	3 ms	3	3 ms
10	328	10	1 ms	8	20 ms	7	32 ms
20	1953	13	2 ms	16	8 ms	13	30 ms
40	16895	33	11 ms	26	12 ms	36	19 ms
80	104962	65	43 ms	74	40 ms	70	43 ms

TABLE 3
Greedy Algorithm versus the Number of Keywords

# of keywords	# of structured queries	Threshold					
		10		20		30	
		#steps	time/step	#steps	time/step	#steps	time/step
2	48	5	1 ms	3	7 ms	4	6 ms
4	1468	14	1 ms	12	19 ms	11	114 ms
6	30463	19	3 ms	15	31 ms	14	252 ms
8	787777	29	10 ms	25	33 ms	21	220 ms
10	47859840	40	25 ms	34	65 ms	36	239 ms

keyword occurrence, and used the probabilistic model in Section 4 to estimate probabilities of structured queries and query construction options.

We believe that this simple simulation is representative, because 1) it simulates the basic structure of a general database schema; 2) it simulates trend that the number of structural interpretations of a keyword query grows polynomially with the size of the database schema and exponentially with the number of keywords.

Size of the database schema. To study the efficiency and scalability of the proposed greedy algorithm in generating query construction plans, we conducted two sets of experiments. In the first set of experiments, we fixed the number of terms in a keyword query to three, and varied the number of tables in the database from 5 to 80. Note that varying the number of columns gives a similar effect as varying the number of tables; therefore, we report only one set of results in this paper. In each experiment, we tuned the threshold of the greedy algorithm from 10 to 30, and simulated the construction of a randomly picked structured query. For each database size, we repeated the experiment 20 times, and recorded 1) the average number of possible structured queries that can be used to interpret a keyword query; 2) the average time for generation of a query construction option and; 3) the average number of options a user needs to evaluate to obtain the intended query. The results of these experiments are shown in Table 2. It can be seen that for a constant number of keywords, the number of possible query interpretations grows very sharply (even polynomially) with the size of database. However, the number of options a user needed to evaluate to construct a query grows only in a similar scale as the database size.

The computation time for generating each query construction option increases with the size of the database as well, but at a low speed. This conforms to the complexity analysis performed in Section 5. We also observed that, with a higher threshold, the greedy algorithm can produce more efficient query construction plans. This is because a higher threshold allows the greedy algorithm to use a larger fraction of the query hierarchy to estimate the information gain of query construction options. This improvement becomes less visible when the threshold increases to a certain value, such as 20 in our experiment. This indicates that the greedy algorithm only needs to evaluate a small fraction of the query hierarchy to achieve its optimal performance.

Size of the keyword query. In the second set of experiments, we fixed the database size to 10 tables and varied the size of a keyword query from 2 to 10 keywords. We repeated the experiments described above. The results are shown in Table 3. As expected, the number of query interpretations grows exponentially with the number of keywords. In contrast, the average number of query construction options a user needs to evaluate grows only linearly with the number of keywords. The computation time for generating a single query construction option also increases slowly. Similar to the results of the previous experiments, higher thresholds of the greedy algorithm can result in better query construction plans, and the improvement becomes insignificant when the threshold increases to 20.

7 Related Work

IQ^P supports efficient incremental construction of structured queries. In this section, we discuss some related work in the database and Information Retrieval areas.

Keyword search in databases. In recent years, keyword search over structured and semistructured data has been extensively investigated. The technology evolved from simple retrieval and ranking of joined tuples [2], [12], [14], [15], [22], [24], to ranking and selection of structured queries [18], [32], [33], [37]. The methods for interpreting keywords evolved from considering attribute values only, to include schema terms (e.g., [22]) and aggregation operators [32]. However, most of these approaches aimed at ranking structured queries and/or search results, and do not offer users opportunities to clarify the semantics of their queries. In contrast, IQ^P enables users to incrementally refine keywords into the desired structured queries.

Faceted search. User-driven query disambiguation has been successfully applied in Information Retrieval in the context of faceted search [25]. Faceted search engines, such as the product search engine of Google and the Clusty search engine [9], 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 [13], [20], [31], [34] were proposed to support users in finding information in a hierarchy of faceted categories. The interface of IQ^P is similar to a faceted interface, whereas each facet corresponds to a query construction option.

In [11] and [38], we presented SUITS, a faceted interface that enables users to interactively disambiguate keyword queries. However, SUITS lacks a theoretical foundation for verifying its effectiveness. In [35], we extended SUITS with a formal framework for incremental query construction and applied this framework to Semantic Web data. However, as the framework of [35] does not take the probability of query interpretations into account, the resulting query construction process may be not optimal. Compared with [11], [35], [38], the contributions of IQ^P include: 1) a probabilistic framework to define the process of incremental query construction; 2) a probabilistic model to estimate the probabilities of structural query interpretations; and 3) an algorithm for generating an optimal QCP based on Information Gain.

Database usability. Database usability is a long-term research issue [16]. One of the early approaches to address this problem was the Query by Example interfaces [5], [36]. Recent approaches include NL query interfaces [1], [3], [6], [21], query autocompletion [4], [28], and adaptive forms [17]. Some commercial DB products, e.g., Microsoft Access, offer visual query builder interfaces. Query graphs in a typical visual query builder interface have to be created manually from scratch. The user has to study the DB schema and manually put together pieces of the query graph. In contrast, IQ^P enables user to focus on interpretations of keywords, and automatically suggests structured queries, without requiring any a-priori schema knowledge.

Natural Language Query Interfaces [1], [3], [6], [21] are intended to enable users to specify structured queries in a human language. Although this provides certain flexibility for database access, state-of-the-art natural language interfaces still require users to use terminology compatible with the database schema and to form grammatically well-formed sentences. IQ^P offers similar expressivity, but with much more flexibility. It accepts arbitrary keyword queries and requires no a-priori schema knowledge.

Query autocompletion [4], [28] assists users to form structured queries, by suggesting possible structures or terms based on the already entered subquery. These suggestions enable users to create database queries without complete schema knowledge. However, this technique still requires users to use a dedicated query language to form structured queries. IQ^P allows users to start with arbitrary keywords and identifies the intended structured query by using meaningful interaction items efficiently.

Forms are typically used to query databases through a predefined query template. Adaptive forms [17] are new techniques to enable easy and flexible access of databases. In contrast to form-based interfaces, IQ^P enables end users to construct queries more flexibly from keywords.

8 Conclusion

In this paper, we presented IQ^P—a novel system, which enables construction of structured queries from keywords. We presented a conceptual framework for the incremental query construction as well as a probabilistic model, which enables consistent assessment of the probability of a query interpretation. We presented an algorithm for generating optimal query construction plans, which enables the user to obtain the intended structured query with a minimal number of interactions. Our experimental results and user study show that IQ^P is highly helpful when user intended structured queries cannot be found within the top-ranked results. Moreover, our simulations confirm the scalability of the proposed algorithms for the data sets with up to 100 tables.

Acknowledgments

The authors would like to thank Irina Oelze for supporting the implementation of the user study. This work has been partially supported by the FP7 EU Projects OKKAM (contract no. 215032), and ARCOMEM (contract no. 270239), and the "HGJ" Projects of China (grant no.2010ZX01042-001-002).

References

[1] I. Androutsopoulos, G.D. Ritchie, and P. Thanisch, "Natural Language Interfaces to Databases—An Introduction," *J. Language Eng.*, vol. 1, no. 1, pp. 29-81, 1995.

[2] S. Agrawal, S. Chaudhuri, and G. Das, "DBXplorer: A System for Keyword-Based Search over Relational Databases," *Proc. Int'l Conf. Data Eng. (ICDE)*, 2002.

[3] M. Al-Muhammed and D.W. Embley, "Ontology-Based Constraint Recognition for Free-Form Service Requests," *Proc. Int'l Conf. Data Eng. (ICDE '07)*, 2007.

[4] H. Bast, A. Chitea, F. Suchanek, and I. Weber, "ESTER: Efficient Search on Text, Entities, and Relations," *Proc. 30th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR)*, 2007.

[5] D. Braga, A. Campi, and S. Ceri, "XQBE (XQuery By Example): A Visual Interface to the Standard XML Query Language," *ACM Trans. Database Systems*, vol. 30, no. 2, pp. 398-443, 2005.

[6] A. Blum, "Microsoft English Query 7.5: Automatic Extraction of Semantics from Relational Databases and OLAP Cubes," *Proc. 25th Int'l Conf. Very Large Data Bases (VLDB)*, 1999.

[7] V.T. Chakaravarthy, H. Gupta, P. Roy, and M. Mohania, "Efficiently Linking Text Documents with Relevant Structured Information," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, 2006.

[8] J. Chambers, W. Cleveland, B. Kleiner, and P. Tukey, *Graphical Methods for Data Analysis.* Wadsworth, 1983.

[9] Clusty, http://clusty.com/, 2011.

[10] E. Demidova, X. Zhou, and W. Nejdl, "IQP: Incremental Query Construction, a Probabilistic Approach," *Proc. 26th IEEE Int'l Conf. Data Eng. (ICDE)*, 2010.

[11] E. Demidova, X. Zhou, G. Zenz, and W. Nejdl, "SUITS: Faceted User Interface for Constructing Structured Queries from Keywords," *Proc. 14th Int'l Conf. Database Systems for Advanced Applications (DASFAA)*, 2009.

[12] H. He, H. Wang, J. Yang, and P.S. Yu, "BLINKS: Ranked Keyword Searches on Graphs," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, 2007.

[13] M.A. Hearst, "Clustering versus Faceted Categories for Information Exploration," *Comm. ACM*, vol. 49, no. 4, pp. 59-61, Apr. 2006.

[14] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient IR-Style Keyword Search over Relational Databases," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, 2003.

[15] V. Hristidis and Y. Papakonstantinou, "DISCOVER: Keyword Search in Relational Databases," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, 2002.

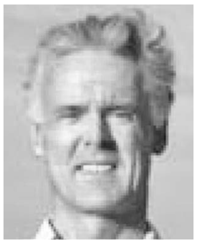
[16] H.V. Jagadish, "Making Database Systems Usable," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, 2007.

[17] M. Jayapandian and H.V. Jagadish, "Expressive Query Specification through Form Customization," *Proc. 11th Int'l Conf. Extending Database Technology: Advances in Database Technology (EDBT)*, 2008.

[18] E. Kandogan, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu, "Avatar Semantic Search: A Database Approach to Information Retrieval," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD),* 2006.
[19] G. Koutrika, A. Simitsis, and Y.E. Ioannidis, "Explaining Structured Queries in Natural Language," *Proc. Int'l Conf. Data Eng. (ICDE),* 2010.
[20] M. Käki, "Findex: Search Result Categories Help Users when Document Ranking Fails," *Proc. SIGCHI Conf. Human Factors in Computing Systems,* 2005.
[21] Y. Li, H. Yang, and H.V. Jagadish, "Constructing a Generic Natural Language Interface for an XML Database," *Proc. Int'l Conf. Extending Database Technology (EDBT),* 2006.
[22] F. Liu, C. Yu, W. Meng, and A. Chowdhury, "Effective Keyword Search in Relational Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD),* 2006.
[23] Z. Liu and Y. Chen, "Identifying Meaningful Return Information for XML Keyword Search," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD),* 2007.
[24] Y. Luo, X. Lin, W. Wang, and X. Zhou, "SPARK: Top-k Keyword Query in Relational Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD),* 2007.
[25] C.D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval.* Cambridge Univ. Press, 2008.
[26] M. McCandless, E. Hatcher, and O. Gospodnetic, *Lucene in Action,* second ed. Manning, 2008.
[27] F. Mesquita et al., "LABRADOR: Efficiently Publishing Relational Databases on the Web by Using Keyword-Based Query Interfaces," *Information Processing and Management: An Int'l J.,* vol. 43, no. 4, pp. 983-1004, 2007.
[28] A. Nandi and H.V. Jagadish, "Assisted Querying Using Instant-Response Interfaces," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD),* 2007.
[29] G. Pass, A. Chowdhury, and C. Torgeson, "A Picture of Search," *Proc. First Int'l Conf. Scalable Information Systems (InfoScale '06),* 2006.
[30] J.R. Quinlan, "Induction of Decision Trees," *Machine Learning,* vol. 1, no. 1, pp. 81-106, Mar. 1986.
[31] S. Roy, H. Wang, G. Das, U. Nambiar, and M.K. Mohania, "Minimum Effort Driven Dynamic Faceted Search in Structured Databases," *Proc. ACM Conf. Information and Knowledge Management (CIKM),* 2008.
[32] S. Tata and G.M. Lohman, "SQAK: Doing More with Keywords," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD),* 2008.
[33] T. Tran, P. Cimiano, S. Rudolph, and R. Studer, "Ontology-Based Interpretation of Keywords for Semantic Search," *Proc. Sixth Int'l the Semantic Web and Second Asian Conf. Asian Semantic Web Conf. (ISWC),* 2007.
[34] P. Wu, Y. Sismanis, and B. Reinwald, "Towards Keyword-Driven Analytical Processing," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD),* 2007.
[35] G. Zenz, X. Zhou, E. Minack, W. Siberski, and W. Nejdl, "From Keywords to Semantic Queries—Incremental Query Construction on the Semantic Web," *J. Web Semantics,* vol. 7, no. 3, pp. 166-176, 2009.
[36] M.M. Zloof, "Query-by-Example: A Data Base Language," *IBM Systems J.,* vol. 16, no. 4, pp. 324-343, 1977.
[37] Q. Zhou, C. Wang, M. Xiong, H. Wang, and Y. Yu, "SPARK: Adapting Keyword Query to Semantic Search," *Proc. Sixth Int'l the Semantic Web and Second Asian Conf. Asian Semantic Web Conf. (ISWC),* 2007.
[38] X. Zhou, G. Zenz, E. Demidova, and W. Nejdl, "SUITS: Constructing Structured Queries from Keywords," technical report, L3S Research Center, Hannover, Germany, http://www.l3s.de/~demidova/suits-TR.pdf, 2008.

Elena Demidova received the MSc degree in 2006 from the University of Osnabrück, Germany. She is currently working toward the PhD degree at the Leibniz Universität Hannover, Germany. Her research interests include databases and information retrieval.

Xuan Zhou received the PhD degree from the National University of Singapore in 2005. He was a researcher at the L3S Research Center in Hannover, Germany, from 2005 to 2008, and a researcher at CSIRO Research Centre, Australia, from 2008 to 2010. Since 2010, he has been an associate professor at the Renmin University of China. His search interests include database system and information management.



Wolfgang Nejdl received the MSc (1984) and PhD degrees (1988) from the Technical University of Vienna, and was an associate professor at the RWTH Aachen from 1992 to 1995. He has been a full professor of computer science at the University of Hannover since 1995. He worked as a visiting researcher/professor at Xerox PARC, Stanford University, UIUC, EPFL Lausanne, and at PUC Rio. He heads the L3S Research Center (http://www.L3S.de/) as well as the Distributed Systems Institute/Knowledge-Based Systems, and does research in the areas of search and information retrieval, information systems, Semantic Web technologies, peer-to-peer infrastructures, databases, technology-enhanced learning, and artificial intelligence. He published more than 230 scientific articles, as listed at DBLP, and has been a program chair, program committee, and editorial board member of numerous international conferences and journals, see also http://www.kbs.uni-hannover.de/~nejdl/. He is a member of the IEEE Computer Society.