Search-As-You-Type in Forms: Leveraging the Usability and the Functionality of Search Paradigm in Relational Databases

Hao Wu
Supervised by Prof. Lizhu Zhou

Department of Computer Science and Technology,
Tsinghua National Laboratory for Information Science and Technology,
Tsinghua University, Beijing 100084, China

haowu06@mails.tsinghua.edu.cn

ABSTRACT

Querying, or searching, is one of the most important issues in relational databases. There are many search paradigms, such as Structured Query Language (SQL), keyword search, and form search, a.k.a. Query-By-Example (QBE). Among them, QBE is a good trade-off between usability and functionality. However, existing QBE systems are often inconvenient for users to compose high-quality queries quickly.

In this PhD workshop paper we investigate the problem of improving the usability of form-based interfaces by enabling them to (1) response a query in real time and (2) tolerate the misplacing of keywords among input boxes. We give the research challenges for achieving high performance and scalability, and introduce two of our prototype systems.

1. INTRODUCTION

Nowadays, relational databases are widely adopted by applications from various domains, and different search paradigms are needed by different users. Experienced users, such as database administrators, need a search paradigm that can provide them accurate and fully functional accessing abilities. In contrast, most of unexperienced users, such as casual Internet users, hope to search databases as easily as possible. Besides, some users, such as system analysts, call for new search paradigms that leverage the usability (ease of use) and functionality (expressive power of queries).

In early days, people could only use *Structured Query Language* (SQL) [5], which is a declarative programming language designed for managing relational data, to access databases. Although SQL is a powerful tool to precisely express users' query intents and control the manner of result display, it is not suitable for casual users because of its complexity and the requirement of programming skill. In other words, it has a low usability for casual users.

VLDB 2010 PhD Workshop, September 13, 2010, Singapore

In recent years, keyword search has become a popular tool to access structured and semi-structured data, such as relational databases and XML documents [6]. Compared with SQL, this search paradigm is extremely easy to use: a user can find her answers only by typing in several keywords in a single input box. However, its functionality is limited by the semantic ambiguity of keywords: a single keyword may refer to different entities, and multiple keywords may refer to a single entity as well. The system could hardly guess a user's accurate intent without any prior knowledge. For example, if we want to find all papers whose titles contain the word "`database`" with CompleteSearch[1] [2], which is a search engine on the Computer Science Bibliography (DBLP)[2] dataset, and pose the keyword "`database`" as the query to the system, then some of the top-ranked results are irrelevant because in each of them the word is contained in conference name instead of paper's title. Another example is that, if we want to find all of Wei Wang's publications, and simply input "`wei wang`" as the query to CompelteSearch, none of the top 20 results are relevant because the system interprets "`wei`" and "`wang`" as two names.

Among various search paradigms, a good trade-off between usability and functionality is the so-called *Query-By-Example* (QBE) [33], which uses previously defined forms as query interfaces. A form usually has multiple input boxes in which users can fill with keywords or drop-down lists for users to select from. Using separated input fields in a form, a user can composite her query more precisely than using a single input box, in which users could only put all their keywords in one place. In a regular QBE-based application, the system first translates the filled form into a structured query, e.g. an SQL statement, and then retrieves the results from the underlying database. This paradigm is more easy-to-use than SQL for causal users, and it is also more expressive and controllable than keyword search with a single input box for experienced users. As a result, QBE has been widely adopted for querying underlying data in real applications, e.g. eBay Advanced Search[3], PubMed Advanced Search[4], and IMDB Power Search[5], etc.

[1] `http://dblp.mpi-inf.mpg.de/dblp`
[2] `http://www.informatik.uni-trier.de/~ley/db`
[3] `http://shop.ebay.com/ebayadvsearch`
[4] `http://www.ncbi.nlm.nih.gov/pubmed/advanced`
[5] `http://www.imdb.com/list`

However, traditional QBE-based systems have one inherent limitation as well: the usability of a form is not satisfactory. Firstly, since the user should first compose a complete query and then submit it to the system, it is often a boring and time-consuming job to find expected results: the user have to repeatedly refine the query, submit the new query, and check the returned results, for many times. Secondly, since there are separated input fields on the form, the errors of misplacing keywords can hardly be avoided. We cannot assume that all users would always input their query conditions into correct input fields.

In this paper, we investigate the problems of improving the usability for form-style interfaces. The new features require more computational resource and storage, so scalability is also a critical issue. Our basic ideas are as follows.

Search-as-you-type. Search-as-you-type is a user-friendly feature which can reduce the efforts of users to refine their queries by returning the results instantly as users type in letters. Existing works [21, 24] on featuring keyword-search systems with search-as-you-type focus on single-input-box interfaces. The advantages of enabling this feature are more obvious in form-style interfaces. Firstly, since a form often consists of multiple input boxes, a user may have to take more effort to refine her query. With the search-as-you-type feature, a user could see the results at once when she gives her query a modification, thus the inconvenience of refining a query in the form can be greatly reduced. Secondly, this feature could also enable the *faceted search* ability of forms. We will discuss these in more detail in Section 2.1.

Tolerate the misplacing of keywords. One of the biggest problems of existing form-based search systems is that they cannot tolerate the misplacing of keywords. What should the system do if a user inputs a person's name into the input box for the movie title? In addition, consider an extreme condition: when a user types all her keywords into one input box, the form-style interface would reduce to the single-input-box interface. That is to say, if we solve the problem of tolerating the misplacing of keywords in forms, we could take the advantages of both of the form and the single-input-box interface. We investigate two possible solutions to address this problem in Section 2.2.

Improve the scalability. Both of the above two features rely on efficient algorithms and effective index structures. If a dataset gets too large, both of the algorithm efficiency and the index size may become unsatisfactory (see the efficiency and scalability evaluations of our initial work in Section 3.2). As a result, we must consider novel algorithms that could handle large datasets. The basic idea is to make use of the DBMS itself or to derive top-k algorithms. We will discuss these in more detail in Section 2.3.

To address the main problems and validate some of our plans, we proposed *Seaform* (stands for Search-As-You-Type in Forms), which is a new search paradigm that supports search-as-you-type feature in form-style interfaces. We also developed two prototype systems in this paradigm and deployed them as web applications[6]: (1) Seaform-DBLP, which searches 1.3 million computer science publications on the DBLP dataset by Title, Authors, Conference/Journal Name, and Year, and (2) Seaform-IMDB, which searches over 500 thousand movies on the Internet Movie Database (IMDB)[7] dataset by Title, Actors/Actresses, Directors, and Year. The screenshot of Seaform-DBLP is shown in Figure 1.

The remainder of the paper is organized as follows. In Section 2 we provide an overview of the problems and give the research challenges and our plans. In Section 3, we describe Seaform, which is our initial work on featuring the search-as-you-type to form-style interfaces over single table datasets. We review related works in Section 4. Finally, we conclude the paper in Section 5.

Figure 1: Screenshot of Seaform-DBLP.

2. PROBLEM STATEMENT

We first consider the case that the database has only one relational table (the case of multiple tables will be discussed in Section 2.4). The table has several *searchable attributes*, such as Title, Name, etc., by which users can search for desired tuples. To enable the faceted search ability (see Section 2.1), we split the original table into several *local tables*, each for a specified searchable attribute. A local table stores all the distinct values of the attribute. Each record in a local table is called a *local record*, and is assigned with a *local id*. Accordingly, the original table is called the *global table*, in which each record is called a *global record* and is assigned with a *global id*. We associate each local table with one or more input boxes in the form.

A query of a form-style interface is segmented into a set of *fields*, each of them contains the query string of the corresponding input box. For example, a query (Author:"`wei wang`",Title:"`data`") has two fields: one contains "`wei wang`" that belongs to the Author input box, the other contains "`data`" that belongs to the Title input box. Each keyword in the query denotes a *prefix* of a word in the dataset.

For each query triggered by a keystroke in an input box, the system returns not only the global ids, called the *global results*, but also the matched local ids in the corresponding local table, called the *local results*, to the user in real time. For example, as is shown in Figure 1, if we type "`wei wang`" in the Author input box, the system returns the names of matched authors (local results) below the form, such as Wei Wang, Weixing Wang, etc., and their publications (global results) on the right side.

Sometimes a user may put some of her keywords into incorrect input boxes. For example, in Seaform-DBLP, a user may input "`wei wang data`" in the Title input box, in which

[6] `http://tastier.cs.tsinghua.edu.cn/seaform`.
[7] `http://www.imdb.com`

"wei wang" is supposed to be an author's name. The system should tolerate the error of misplacing "wei wang" into the wrong input box by enumerating all the possible keyword-to-input-box mappings, each of which is called an *interpretation*, and then showing them to the user by a descending order of possibilities. In the above example, two of the possible interpretations are (Author:"wei wang",Title:"data") and (Author:"wei wang",Venue:"data"). The system shows these interpretations as a list to the user to choose from.

2.1 Search-As-You-Type

With the help of the search-as-you-type feature, a user can get the search result instantly when she modifies her query. Besides, enabling search-as-you-type could also let users benefit from another interesting feature: the *real-time faceted search*. Faceted search, also called faceted navigation or faceted browsing, is a technique which provides an on-the-fly categorization ability to the system to make the underlying data more comprehensible to users [12]. For example, CompleteSearch can group the result publications by Author, Venue, or Year according to the query keywords. A form with search-as-you-type feature inherently has this faceted search ability. For example in Figure 1, by displaying the matched authors according to "wei wang", the system groups the publications by authors. A user can navigate the results by clicking on one of those authors, and get a deeper understanding about the dataset.

Enabling the search-as-you-type feature to form-style interfaces is not trivial. Generally, there are two challenges to achieve this goal. Firstly, as we wish the system to return both of the local results and the global results simultaneously in real time, we will encounter the challenge of accelerating the speed of synchronization:

Challenge 1: Synchronization Speed

The local results of an input box cannot be retrieved only according to the keywords in the current input box, but should also be constrained the keywords in other input boxes. For example, as is shown in Figure 1, if we input "pattern" in Title and then type "wei wang" in Author, the author *Weizhao Wang* is not included in the local results, although it matches the keywords "wei wang", because this author does not have a paper whose title contains "pattern". We call the process of filtering out the false matched local records *synchronization*. A straightforward way is to retrieve the global results according to all the input boxes, and then check each local record that matches the keywords in the currently editing input box to see whether it appears in a result global record to filter out false positives. But this method cannot satisfy the requirement of real-time response of search-as-you-type feature. Our initial works proposed *on-demand synchronization*, and our experiments show that it can greatly accelerate the overall search speed (see Section 3.2). However, in some cases, e.g. the set of global results is very large, our on-demand synchronization paradigm would also be inefficient.

Secondly, an effective index and its corresponding efficient search algorithms for large datasets are necessary:

Challenge 2: Effective Index

Existing works on search-as-you-type in single-input-boxes use *trie* structures to index all the prefixes of keywords and use inverted lists to store corresponding record ids for each keyword [21, 24]. However, the situation is more complicated in form-style interfaces because (1) we need to return the local results and the global results simultaneously and (2) we also need to perform synchronization. In our initial work (see Section 3.2), we introduce *dual-list tries* to index the ids of both the local records and the global records in which a keyword appears. Our experimental results show that the search algorithm is 2 times faster than using the classical *single-list tries*. However, a dual-list trie often requires more storage than a single-list trie, so it is a trade-off between space and efficiency. In addition, the need for an effective index would get more urgent if we also take other challenges, e.g. the scalability issue, into consideration.

2.2 Misplacing of Keywords

As is explained before, tolerating misplacing of keywords is a necessary feature for a form-style interface. As a user inputs her query keywords into the input boxes, the system should enumerate all the possible interpretations and then sort them according to their possibilities. This enumeration process requires both of high accuracy and high performance if we retain the search-as-you-type feature. Thus the challenge is that how to find the interpretations as fast as possible, referred to *interpret-as-you-type*.

Challenge 3: Interpret-As-You-Type

One of the possible methods is based on statistical analysis, i.e. to estimate the probability of each interpretation. There are two issues for us to consider: (1) how to measure the probability, and (2) how to permutate all the possible interpretations quickly and sort them in descending order according to their probabilities. For the first issue, we could assume that each keyword are independent to others, thus the calculation would be simple. However, this independence assumption is often too strict: some keywords may belong to the same phrase. For the second issue, the independence assumption may lead to performance problems because the search space of interpretations may get very large. However, the above method has a limitation: the estimation could not be accurate enough. So we should consider other methods in a different way.

Another possible method is based on on-the-fly search. Given a query posed by the user, the system permutates all the possible interpretations, and then performs an on-the-fly search for each of the them. Obviously, this method leads to high computational overhead. So it is necessary to (1) reduce the search space, (2) design effective pruning techniques, and (3) devise an incremental algorithm. Note that the performance of this method also depends on the efficiency of the basic form search algorithm itself.

In addition, consider an extreme condition that the user types all her keywords in just one input box in the form. The situation is similar to traditional single-input-box keyword search. The difference is that the system should not only return the results, but also provide the interpretations to the user, and all these tasks require high response speed.

2.3 Scalability

Our initial work shows that both of the index size and the response time increase linearly as the dataset gets larger (see Section 3.2). When the size of the dataset is too large, we cannot store the index in the main memory. In this case, it is necessary to investigate other ways to build the index, as well as more scalable search algorithms. We investigate two

of the possible solutions to this issue: (1) use native DBMS support and (2) design a top-k algorithm.

Challenge 4: Native DBMS Support

When data is stored in a relational database, it is natural to use SQLs to perform the searching. In this way, the system would require less efforts to deploy and less main memory to run. In addition, we could also pay less attention on reducing the index size, since the storage and retrieval algorithms are already carefully designed and chosen in the system natively. That is to say, we only put them all in the DBMS, and then use DBMS capabilities to support the two new features. Nevertheless, how to maintain the data and the indexes, as well as how to design a high-performance search algorithm, are still need to be take into consideration. As described before, the whole search process can be divided to three sub-tasks: (1) search for local results, (2) search for global results, and (3) synchronization.

The first two sub-tasks are similar to the classical search-as-you-type in single-input-boxes. As existing works search-as-you-type with single-input-boxes use trie structures to index all the prefixes of keywords, we can 'serialize' each of the trie structures into a relational table: we store the `(prefix,record_id)` tuples for all of the keywords in all of the records. The real challenge is that how can we use SQLs and the indexed `(prefix,record_id)` tuples to retrieve the local/global results and perform the synchronization operation quickly, due to the fact that often the performance of executing a complex SQL query in the DBMS is not high enough for real time responses.

Challenge 5: Top-k Algorithms

As is mentioned before, our initial work shows that, the query time (running time of search algorithm) increases linearly with the growth of the dataset. In another word, the algorithm incorporated by our initial work can hardly be adapted to handle large datasets, since we need to achieve high interactive speed. Traditional keyword search algorithms incorporate top-k algorithms to guarantee that the query time increases sub-linearly as dataset grows. With a top-k search algorithm, given a user-defined parameter k, the system returns only k top-ranked results, instead of all of them to the user. Because many unnecessary computations to retrieve the results with low ranking scores can be avoided, the overall query time can be improved. However, specifically, for search-as-you-type in forms, the problem is a little bit more complex: since the system returns the local results and the global results at the same time, there should be 2 parameters, l and k, with which we could control the amount of returned local results and global results respectively. We call this kind of algorithm the top-(l, k) algorithm. Recall that in the non-top-k case, we perform the process of synchronization to filter out all false positive local results according to retrieved global results. The question in the top-(l, k) case is that, how can we perform the synchronization without knowing all of the global results? This is an open issue that needs to be addressed carefully.

2.4 Other Issues

Challenge 6: Informativeness of Local Results

Recall that search-as-you-type in forms inherently has the ability of faceted search. That is, when a user types in some keywords in an input box, besides the global results, the system would also return the matched attribute values according to the keywords in the current input box, which are called local results. For example, if we input "`wang`" in the Author input box in our Seaform-DBLP prototype system, a list of names, such as "`Wei Wang`", "`Jun Wang`", etc., will be also shown to us for faceted navigation. We call a local result *informative* if it maps to many global results, for example "`Wei Wang`" in the above example. In contrast, if we input keywords in the Title input box, often a local result can only maps to 1 or 2 global results, because it is rare that two publications have the same title, especially if the title is long. We say these local results are *uninformative*, which have a very low necessity to display for faceted search. It is possible to incorporate text mining techniques to group these uninformative local results by meaningful phrases, and then show these phrases as suggestions. However, how to define *meaningfulness* and how to obtain meaningful phrases on-the-fly are also open issues to tackle.

Challenge 7: Multiple Tables

Most of the existing search-as-you-type systems, including our Seaform-based prototype systems, use single relational tables as their underlying data, whereas in real applications a dataset usually has multiple tables. Traditional form-based systems use pre-defined SQL to join all the matched tuples together from different tables on the fly. Because of the performance issue of the on-the-fly join using SQL, we have to consider more efficient techniques that can handle multiple tables to achieve a high interactive speed. In fact, some of the techniques used in our prototype systems can also be adapted to support multiple tables. However, the scalability of our original algorithms are not good enough because all the computations, including the costly on-the-fly joins, should be performed in the main memory. If we want to improve the scalability using, say, native DBMS support or top-k algorithms, performing on-the-fly joins efficiently would be getting even harder.

3. INITIAL ACHIEVEMENTS

We have proposed a new search paradigm, called *Seaform* (stands for Search-As-You-Type in Forms), to address some of the problems summarized in Section 2. We hvae also developed two prototype systems, Seaform-DBLP and Seaform-IMDB. In this section, we briefly introduce (1) the main features, (2) the underlying techniques, and (3) the current status and future directions, of Seaform-based systems.

3.1 Main Features

A Seaform-based system takes a single relational table as its underlying data. For example, the DBLP dataset is a table with Title, Authors, Booktitle, Year, etc., as its columns, and each row in this table refers to a publication. We will illustrate 3 of the main features of a Seaform-based system by providing several examples.

Feature 1: Precise search paradigm. Suppose a user wants to find papers by Wei Wang whose titles contain the word "`pattern`". If she types in "`wei wang pattern`" in CompleteSearch, many returned results are not very relevant. In contrast, if she types in "`wei wang`" and "`pattern`" in different input boxes in Seaform-DBLP, she can find high-quality results.

Feature 2: Search-as-you-type. Suppose the user wants to find the movie titled *The Godfather* made in 1972 using

the IMDB Power Search interface. She is not sure if there is a space between the word "god" and the word "father", so she fills in the Title input box with "god father". Unfortunately, after waiting for several seconds, the user still does not get relevant result. So she has to try a new query. In contrast, in Seaform-IMDB, she can modify the query and see the new results instantaneously.

Feature 3: Faceted search. Suppose the user has limited prior knowledge about the KDD conference and wants to know more about it using Seaform-DBLP. At first, she wants to know how many papers were published in this conference each year. She types in "kdd" in the Venue input box and then changes the editing focus to the Year input box. The listed local results show the years sorted by the number of published papers. Next, she wants to know who published how many papers in KDD 2009. To do this, she chooses the year 2009 by clicking on the list, and changes the focus to the first Author input box. The list below the form shows the authors. She can see that the most *active* author. For instance, the author with the most publications is Christos Faloutsos. She then chooses "Christos Faloutsos" and changes the focus to the second Author input box. Then all the co-authors are listed. After several rounds of typing and clicking, the user can get a deeper understanding about the conference.

3.2 Under the Hood

A Seaform-based system uses the client-server architecture. On the client side, each keystroke event triggered by the user in any of the input boxes invokes the JavaScript code to issue an AJAX query to the server, and then the client displays the results returned from the server with query keywords highlighted. The server side has the following components. The Indexer indexes the underlying data. When a query is received, the Searcher searches the index for both the global results and the local results incrementally with the help of the Cache component, which caches the previous queries and their results. The Result Composer component ranks the results and sends them to the client. All components are in a FastCGI module.

We build three types of indexes for each of the searchable attributes: (1) a *trie structure*, which indexes all the prefixes of keywords over all the attribute value strings and the inverted lists of local ids; (2) a *local-global mapping table*, which maps each local id to a list of global ids; and (3) a *global-local mapping table*, which maps each global id to a list of local ids. The trie structure are used for supporting the search-as-you-type feature, and the last two tables are used for synchronization.

We have designed two incremental search algorithms: one is based on *single-list trie* (SL), which means that we only attach the inverted list of local ids to the leaf nodes of tries, and the other is based on *dual-list trie* (DL), which means that we attach both of the inverted lists of local ids and global ids to the leaf nodes of tries. Besides, we have also designed the *on-demand synchronization* (OD) mechanism, which means that we perform the synchronization operation only when the user's input focus is changed, as an alternative to the original *brute-force synchronization* (BF) mechanism, i.e. to synchronize on every keystroke. We investigate the performances of the four combinations, i.e. SL-BF, SL-OD, DL-BF, and DL-OD, by comparing their average query times of a workload of 45,276 real queries using the Seaform-DBLP prototype system on a Intel Core-2 2.4GHz PC with 2GB of RAM. All the code were implemented in C++ and compiled using Visual Studio 2008. The result shows that the DL-OD combination is the fastest, which can response a user's query within 50 ms. on average (see Figure 2).

Figure 3 shows the scalability of Seaform-DBLP. As we can see, both of the the index size and the average query time would increase linearly with the growth of the dataset. Although the index size will get slightly larger if we use DL compared with that of using SL (10% larger), it is worthy to choose the former because we can make the search 2 times faster according to Figure 2.

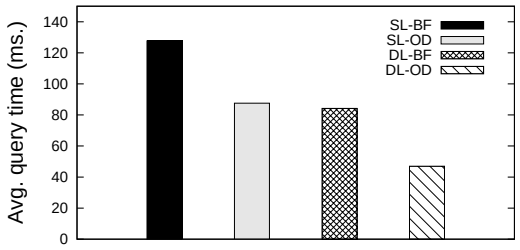
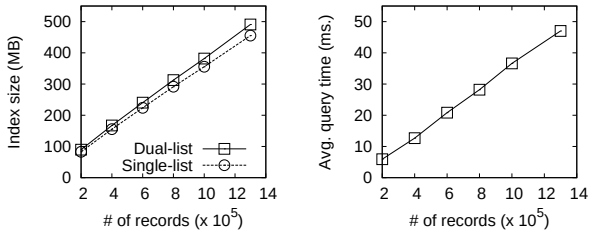


Figure 2: Performances of the four combinations.



(a) Index sizes using SL/DL. (b) Avg. query time of DL-OD.

Figure 3: The scalability of Seaform-DBLP.

3.3 Current Status and Future Works

Our two prototype systems have been launched since Nov., 2009. We have received over 40,000 queries from hundreds of distinct IP addresses. Users' feedbacks show that, when a user wants to search for something *accurately*, our systems can indeed make the task easier. Nevertheless, there are still many works to do in the future.

Firstly, users reported that the returned local results of Title input box cannot provide useful information for faceted search because there are little publications/movies that have the same title. We should summarize these local results into groups to improve the faceted search ability. Secondly, our systems support only AND-semantics for the queries. We will investigate the OR-semantics and top-k algorithms in the future to make the searching more flexible and efficient. Thirdly, we should also tolerate misplacing of keywords to leverage the simplicity and usability of forms.

4. RELATED WORKS

Query-By-Example [33] is the earliest paradigm that could enable a user query a database without using SQLs. The basic idea is to provide a form with input boxes and drop-down lists for the user to fill in with, and convert the filled form to SQL statements to retrieve the results.

In recent years, keyword search has been used as a novel search paradigm in structured and semi-structured databases [6]. There are two categories of techniques: one is based on *candidate networks* [1, 16, 15, 25, 26, 30], and the other is based on *data graphs* [4, 22, 10, 14]. Some of these works, such as [15, 26, 14], support OR-semantics and top-k queries. In the scenario of using form-style interfaces, to our best knowledge, there is no existing works on supporting these features. See [17] for a good survey of the algorithms that answer top-k queries.

Utilizing search-as-you-type feature in forms is inspired by the existing works on Autocomplete [18, 13, 28, 29], type-ahead keyword search [21, 24], and the CompleteSearch [2]. These works are all based on single-input-box interfaces.

Faceted search [12, 27, 8, 9, 31], originally proposed in the information retrieval community, provides users a convenient way to navigate the dataset by drilling down or up on a dynamically computed structure. In the database community, [3] enables a navigational technique for relational databases on faceted search. [2] and [11] also provide faceted search interfaces over the DBLP dataset. In addition, [23] and [32] enable users to navigate the underlying dataset by choosing one of the frequently occurred terms.

There are also recent works on keyword search in form-style interfaces, in which [19, 20] focused on form creation, and [7] focused on finding the most possible interfaces for keyword search. Compared with them, our works focus on enabling the search-as-you-type feature to the form.

5. CONCLUSIONS

In this paper, we analyze three paradigms to access relational databases: SQL, QBE, and keyword search, and investigate the problems and challenges of enabling the search-as-you-type feature to form-style interfaces. The challenges include: (1) increase the usability of forms, and (2) improve the scalability of the search-as-you-type algorithms of forms. We illustrate our recently developed prototype systems, and point out some possible future works. In a word, the form with the search-as-you-type feature is a nice interface for access relational databases, and there are also important problems that need to be addressed in the future.

6. ACKNOWLEDGEMENTS

This work is partially supported by the National Natural Science Foundation of China under Grant No. 60873065, the National High Technology Development 863 Program of China under Grant No. 2009AA011906, and the National Grand Fundamental Research 973 Program of China under Grant No. 2006CB303103.

7. REFERENCES

[1] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *ICDE*, pages 5–16, 2002.

[2] H. Bast and I. Weber. Type less, find more: fast autocompletion search with a succinct index. In *SIGIR*, pages 364–371, 2006.

[3] S. Basu Roy, H. Wang, G. Das, U. Nambiar, and M. Mohania. Minimum-effort driven dynamic faceted search in structured databases. In *CIKM*, pages 13–22, 2008.

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

[5] D. D. Chamberlin and R. F. Boyce. SEQUEL: A structured english query language. In *SIGMOD Workshop, Vol. 1*, pages 249–264, 1974.

[6] Y. Chen, W. Wang, Z. Liu, and X. Lin. Keyword search on structured and semi-structured data. In *SIGMOD Conference*, pages 1005–1010, 2009.

[7] E. Chu, A. Baid, X. Chai, A. Doan, and J. F. Naughton. Combining keyword search and forms for ad hoc querying of databases. In *SIGMOD Conference*, pages 349–360, 2009.

[8] W. Dakka, P. G. Ipeirotis, and K. R. Wood. Automatic construction of multifaceted browsing interfaces. In *CIKM*, pages 768–775, 2005.

[9] W. Dakka, P. G. Ipeirotis, and K. R. Wood. Faceted browsing over large databases of text-annotated objects. In *ICDE*, pages 1489–1490, 2007.

[10] B. B. Dalvi, M. Kshirsagar, and S. Sudarshan. Keyword search on external memory data graphs. *PVLDB*, 1(1):1189–1204, 2008.

[11] J. Diederich and W.-T. Balke. The semantic growbag algorithm: Automatically deriving categorization systems. In *ECDL*, pages 1–13, 2007.

[12] J. English, M. A. Hearst, R. R. Sinha, K. Swearingen, and K.-P. Yee. Hierarchical faceted metadata in site search interfaces. In *CHI Extended Abstracts*, pages 628–639, 2002.

[13] K. Grabski and T. Scheffer. Sentence completion. In *SIGIR*, pages 433–439, 2004.

[14] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: ranked keyword searches on graphs. In *SIGMOD Conference*, pages 305–316, 2007.

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

[16] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *VLDB*, pages 670–681, 2002.

[17] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):1–58, 2008.

[18] M. Jakobsson. Autocompletion in full text transaction entry: a method for humanized input. *SIGCHI Bull.*, 17(4):327–332, 1986.

[19] M. Jayapandian and H. V. Jagadish. Automated creation of a forms-based database query interface. *PVLDB*, 1(1):695–709, 2008.

[20] M. Jayapandian and H. V. Jagadish. Automating the design and construction of query forms. *IEEE Trans. Knowl. Data Eng.*, 21(10):1389–1402, 2009.

[21] S. Ji, G. Li, C. Li, and J. Feng. Efficient interactive fuzzy keyword search. In *WWW*, pages 371–380, 2009.

[22] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, pages 505–516, 2005.

[23] G. Koutrika, Z. M. Zadeh, and H. Garcia-Molina. Data clouds: summarizing keyword search results over structured data. In *EDBT*, pages 391–402, 2009.

[24] G. Li, S. Ji, C. Li, and J. Feng. Efficient type-ahead search on relational data: A TASTIER approach. In *SIGMOD Conference*, pages 695–706, 2009.

[25] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD Conference*, pages 563–574, 2006.

[26] Y. Luo, X. Lin, W. W. 0011, and X. Zhou. SPARK: top-k keyword query in relational databases. In *SIGMOD Conference*, pages 115–126, 2007.

[27] I. Martin and J. M. Jose. A personalised information retrieval tool. In *SIGIR*, pages 423–424, 2003.

[28] A. Nandi and H. V. Jagadish. Assisted querying using instant-response interfaces. In *SIGMOD Conference*, pages 1156–1158, 2007.

[29] A. Nandi and H. V. Jagadish. Effective phrase prediction. In *VLDB*, pages 219–230, 2007.

[30] L. Qin, J. X. Yu, and L. Chang. Keyword search in databases: the power of RDBMS. In *SIGMOD Conference*, pages 681–694, 2009.

[31] E. Stoica, M. A. Hearst, and M. Richardson. Automating creation of hierarchical faceted metadata structures. In *HLT-NAACL*, pages 244–251, 2007.

[32] Y. Tao and J. X. Yu. Finding frequent co-occurring terms in relational keyword search. In *EDBT*, pages 839–850, 2009.

[33] M. M. Zloof. Query-by-example: the invocation and definition of tables and forms. In *VLDB*, pages 1–24, 1975.