Automatic URL Completion and Prediction Using Fuzzy Type-Ahead Search

Jiannan Wang[†] Guoliang Li[†] Jianhua Feng[†] Chen Li[‡]
[†]Department of Computer Science and Technology, Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 10084, China
[‡]Department of Computer Science, University of California, Irvine, USA
wjn08@mails.thu.edu.cn; {liguoliang,fengjh}@tsinghua.edu.cn; chenli@ics.uci.edu

ABSTRACT

Type-ahead search is a new information-access paradigm, in which systems can find answers to keyword queries "on-the-fly" as a user types in a query. It improves traditional autocomplete search by allowing query keywords to appear at different places in an answer. In this paper we study the problem of automatic URL completion and prediction using *fuzzy type-ahead search*. That is, we interactively find relevant URLs that contain words matching query keywords, even approximately, as the user types in a query. Supporting fuzzy search is very important when the user has limited knowledge about URLs. We describe the design and implementation of our method, and report the experimental results on Firefox.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*query formulation, search process*

General Terms

Algorithms, Experimentation, Performance

Keywords

Type-Ahead Search, Fuzzy Search, URL Completion

1. INTRODUCTION

Search engines changed the way people surf the Internet. They help users easily find relevant Web pages after users type in keywords. In the case where a user has limited knowledge about the underlying data, often the user feels "left in the dark" when issuing queries, and has to use a try-and-see approach for finding information. Recently type-ahead search has been proposed to address this problem [1, 4]. Type-ahead search helps users interactively find answers as they type in query keywords. Modern browsers, such as Firefox 3.0[1] and Sogou[2], support type-ahead search on URLs. They use the location bar as a search box, and suggest URLs in a drop-down menu as users type in query keywords.

In this paper we study how to improve these browsers based on the following observations. Firefox only supports client-based type-ahead search. That is, it only uses local (personal) information for type-ahead search. Firefox cannot predict URLs that have not been visited by the user. Moreover, a user cannot use his/her URL history from another computer. Sogou solves the problem by supporting server-based type-ahead search. In some cases if a user does not know the exact spelling of a URL due to limited knowledge, Sogou cannot predict such URL based on mistyped keywords. For instance, if a user mistypes a keyword "sougou" for "`www.sogou.com`", Sogou cannot find the correct Web site.

To address these problems, we propose a fuzzy type-ahead technique to support automatic URL completion and prediction. As a user types in keywords, our method predicts relevant URLs that contain words *similar* to the query keywords. We devise novel techniques to support this feature, implemented the method on a large number of URLs to demonstrate its efficiency and practicality.

[1] http://www.mozilla.com/en-US/firefox/
[2] http://ie.sogou.com/

SIGIR'09, July 19–23, 2009, Boston, Massachusetts, USA.
ACM 978-1-60558-483-6/09/07.

2. FUZZY TYPE-AHEAD SEARCH ON URLS

Server Design: We use a client-server architecture. The server has several components. A component called "Indexer" indexes the URL dataset as a trie structure. A FastCgi module on the server stores the data and indices. The module waits for queries from the client. For each query keyword, a module called "Prefix Finder" computes the prefixes similar to the query keywords. A module called "top-k Answer Finder" computes the best URLs that contain keywords *similar* to the query keywords.

Indexer: It reads a data set with URLs and their titles, tokenizes the URLs, and creates a trie structure with inverted lists on the leaf nodes. The URLs on each inverted list are sorted according to their weights (e.g., PageRank) in a descending order. Figure 1 shows a partial index structure for a URL dataset.

Incremental Prefix Finder [3]: It incrementally identifies prefixes in the dataset that are similar to the query keywords, measured by their edit distance. The output of this module is a set of prefixes with their edit distances to the query keyword. Suppose we are given an edit-distance threshold $\delta = 1$ for the keyword "`sun`". This module will find the exactly matched prefix "`sun`" (trie node 3), and the similar prefixes "`su`", "`son`", and "`sin`" (trie nodes 2, 5, and 8).

Top-k Answer Finder: It finds the best URLs for the partial keywords. The URLs are ranked by their inherent importance and the relevance to the query. Suppose the query is $Q = \{p_1, p_2, \ldots, p_n\}$, and p'_i is the best matched prefix for p_i. Let $sim(p_i, p'_i)$ be an edit distance between p'_i and p_i.

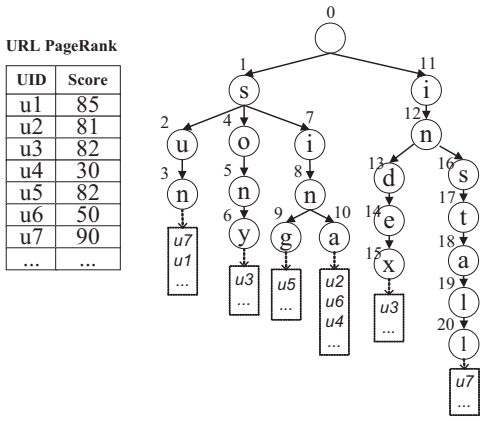


Figure 1: Trie with inverted lists at leaf nodes

The score of a URL u for p_i can be computed as:

$$Score(u, p_i) = \frac{pr(u)}{\alpha \cdot sim(p_i, p_i')^2 + \beta}, \quad (1)$$

where $pr(u)$ is the PageRank of URL u, and α and β are smoothing parameters to adjust the relative importance between $sim(p_i, p_i')$ and $pr(u)$ ($\alpha, \beta > 0$). The score of a URL u for Q is computed as:

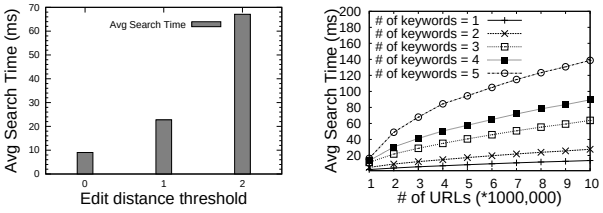
$$Score(u, Q) = \sum_{i=1}^{n} Score(u, p_i). \quad (2)$$

Consider a query with n keywords. Traditional threshold-based algorithms [2] compute the top-k results on n materialized sorted lists. For our fuzzy search, such lists must be computed on-the-fly as there are multiple predicted words and corresponding inverted lists for each input keyword. For instance, given a query "`sun ins`", there are two sorted lists: one for "`sun`" and one for "`ins`". Each list consists of the URLs sorted by the score computed by Equation (1), including multiple similar prefixes. For example, the URLs in the sorted list for "`sun`" are the merged results of the inverted lists of the trie nodes 3, 6, 9, and 10. To compute such sorted lists, we build a max-heap on top of the inverted lists of its predicted words. After popping the top element from the heap, we adjust the heap and get the next element with the maximal score. For each popped URL, we compute its score based on Equation (2). Thus, we can use the threshold-based algorithm to find the top-k answers [2].

Client Design: We implemented the feature as an add-on to Firefox. A user can install the add-on to enjoy the feature of fuzzy type-ahead search on URLs. When a user types in a query on the location bar using our implemented add-on, Firefox issues an AJAX query to the server, and waits until the request has been answered. The matched URLs are returned with predicted words highlighted. Firefox 3.0 uses an XPCOM component called "history" to support its URL suggestion. Firefox implements a client-based function. To enable the server-based URL suggestion, we develop an autocomplete search component by implementing the required interface. Firefox highlights search results in the drop-down menu based on query keywords. Any parts of the URL that exactly match the keyword will be highlighted. In our method, we need to highlight the similar prefixes of every input keyword. We override the Firefox highlighting algorithm, in case the user wants to use both the client-based URL suggestion and the server-based URL suggestion. We merge existing Firefox search results with ours. When the user query returns enough results from the client, we will not send the query to the server. If the user inputs a mistyped query or searches a URL that has never been visited before, the results from the server are appended to the drop-down menu.

3. EXPERIMENTS

We evaluated our methods on Firefox using the SogouRank dataset[3], which includes 130 million URLs with PageRank values. We selected 10 million distinct URLs with the highest PageRank values. We generated five sets of queries, and each set contained 1,000 queries with the same number of keywords. For each keyword in a query, we applied a random number of edit operations (0 to 2) on the keyword. The backend server was implemented in C++. The experiments were run on a computer with an Intel 3GHz CPU and 8 GB RAM, running Ubuntu.



(a) Top-10 search efficiency (b) Scalability

Figure 2: Experimental results

We evaluated the average search time of a query for finding top-10 answers. Figure 2(a) shows the experimental results. Each keystroke for most queries can be processed within 70ms. We then evaluated the scalability of our techniques, and Figure 2(b) shows the results. We observe that the algorithm scaled linearly as the number of URLs increased. For 1 million URLs, each keystroke was answered within 15ms. For 10 million URLs, each keystroke was answered within 140ms. These numbers show the efficiency and practicality of our method on large URL data sets.

Acknowledgements

The work was 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. 2007AA01Z152, the National Grand Fundamental Research 973 Program of China under Grant No. 2006CB303103, 2008 HP Labs Innovation Research Program, and the US NSF award No. IIS-0742960.

4. REFERENCES

[1] H. Bast and I. Weber. Type less, find more: fast autocompletion search with a succinct index. In *SIGIR*, pages 364–371, 2006.
[2] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*. ACM, 2001.
[3] S. Ji, G. Li, C. Li, and J. Feng. Efficient interactive fuzzy keyword search. In *WWW*, pages 371–380, 2009.
[4] G. Li, S. Ji, C. Li, and J. Feng. Efficient type-ahead search on relational data: a tastier approach. In *SIGMOD*, 2009.

[3] http://www.sogou.com/labs/dl/t-rank.html