

if a related suggestion has the query keywords but not consecutively, then this suggestion cannot be found

H. Bast and I. Weber proposed many indexing and query techniques to support instant search, Imagine a user of a search engine typing a query. Then with every letter being typed, we would like an instant display of completions of the last query word which would lead to good hits. At the same time, the best hits for any of these completions should be displayed. Known indexing data structures that apply to this problem either incur large processing times for a substantial class of queries, or they use a lot of space. We present a new indexing data structure that uses no more space than a state-of-the-art compressed inverted index, but that yields an order of magnitude faster query processing times. Even on the large TREC Terabyte collection, which comprises over 25 million documents, we achieve, on a single machine and with the index on disk, average response times of one tenth of a second. We have built a full-fledged, interactive search engine that realizes the proposed autocompletion feature combined with support for proximity search, semi-structured (XML) text, subword and phrase completion, and semantic tags.

S. Chaudhuri, V. Ganti, and R. Motwani, Proposed Detecting and eliminating fuzzy duplicates is a critical data cleaning task that is required by many applications. Fuzzy duplicates are multiple seemingly distinct tuples, which represent the same real-world entity. We propose two novel criteria that enable characterization of fuzzy duplicates more accurately than is possible with existing techniques. Using these criteria, we propose a novel framework for the fuzzy duplicate elimination problem. We show that solutions within the new framework result in better accuracy than earlier approaches. We present an efficient algorithm for solving instantiations within the framework. We evaluate it on real datasets to demonstrate the accuracy and scalability of our algorithm.

G. Li, J. Wang, C. Li, and J. Feng Proposed the approach is especially suitable for instant and fuzzy search since each query is a prefix and trie can support incremental computation efficiently.

Type-ahead search can on-the-fly find answers as a user types in a keyword query. A main challenge in this search paradigm is the high-efficiency requirement that queries must be answered within milliseconds. In this paper we study how to answer top-k queries in this paradigm, i.e., as a user types in a query letter by letter, we want to efficiently find the k best answers. Instead of inventing completely new algorithms from scratch, we study challenges when adopting existing top-k algorithms in the literature that heavily rely on two basic list-access methods: random access and sorted access. We present two algorithms to support random access efficiently. We develop novel techniques to support efficient sorted access using list pruning and materialization. We extend our techniques to support fuzzy type-ahead search which allows minor errors between query keywords and answers. We report our experimental results on several real large data sets to show that the proposed techniques can answer top-k queries efficiently in type-ahead search.

3. Proposed System

We are maintaining log for user searchlog and part of relevant retrieved contents in TLB like mechanism. When user enters query to search engine, we will extract keywords from the query, this will need following algorithms .

3.1 Porters Stemming Algorithm

Porters stemming algorithm used to remove suffixes from the keyword . For example, we need to find occurrences of keyword play in a result. We will get different words like **plays, played, playing etc.** We cant find exact occurrence of **play** without removing suffixes like ed, ing, s, sses, ation, ational etc. There are some exceptional cases like **Red** this keyword is ended with **ed** but we dont want to remove **ed** from the **Red**. For stemming with such kind of examples porter have defined algorithm which has 6 steps implementation. Porter have defined some rules to remove suffixes for example (V-Vowel , C-Consonent) , PLAYED RED
CCVCVC CVC

Count m= |VC pairs| ie. for played m=2 and for red m=1. Rule defined by porter is if (m>1 && str.endsWith("ed")) remove "ed" from str else dont remove "ed".

3.2 Stopword Removal Algorithm

We need to download stopwords dataset from internet which is easlily available on many websites. Compare the words with stopwords database and remove stopwords occurrences for further computations.

3.3 K-Means clustering algorithm with consideration of hop count

K-means clustering algorithm clusters the user keyword log based on its number of occurrences. High frequency cluster is maintained at TLB side. When user enters query it will firstly look in TLB, if he dont gets relevant results then it calls for the pages/ results from fuzzy search mechanism .

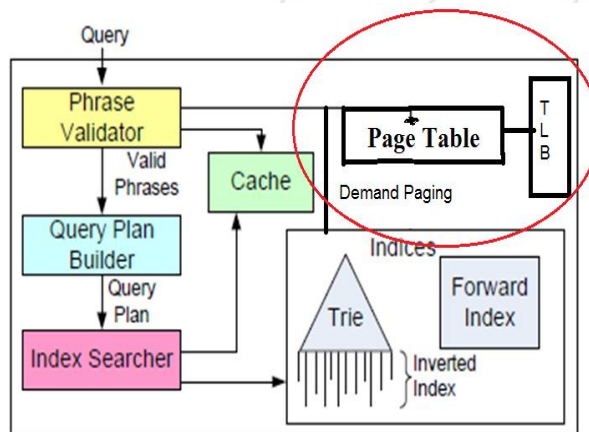
3.4 Fuzzy Search Mechanism

We formalize the problem of interactive, fuzzy search on a relational table, and our method can be adapted to textual documents, XML documents, and relational databases. Consider a relational table T with m attributes and n records. Let $A = \{a_1, a_2, \dots, a_m\}$ denote the attribute set, $R = \{r_1, r_2, \dots, r_n\}$ denote the record set, and $W = \{w_1, w_2, \dots, w_p\}$ denote the distinct-word set in T. Given two words w_i and w_j , " $w_i w_j$ " denotes that w_i is a prefix string of w_j . A query consists of a set of prefixes $Q = \{p_1, p_2, \dots, p_l\}$. For each prefix p_i , we want to find the set of prefixes from the data set that are similar to p_i . In this work we use edit distance to measure the similarity between two strings. The edit distance between two strings s_1 and s_2 , denoted by $ed(s_1, s_2)$, is the minimum number of edit operations (i.e., insertion, deletion, and substitution) of single characters needed to transform the first one to the second. For example, $ed(\text{smith}, \text{smyth}) = 1$. Definition 1 (Interactive Fuzzy Search). Given a set of records R, let W be the set of words

in R . Consider a query $Q = \{p_1, p_2, \dots, p_k\}$ and an edit-distance threshold δ . For each p_i , let $P_i = \{w \in W, p_i \text{ is a substring of } w \text{ and } \text{ed}(p_i, w) \leq \delta\}$. Let the set of candidate records $RQ = \{r \in R, \forall 1 \leq i \leq k, \exists p_i \in P_i \text{ and } p_i \text{ appears in } r\}$. The problem is to compute the best records in RQ ranked by their relevancy to Q . These records are computed incrementally as the user modifies the query, e.g., by typing in more letters.

3.4 Indexing

We use a trie to index the words in the relational table. Each word w in the table corresponds to a unique path from the root of the trie to a leaf node. Each node on the path has a label of a character in w . For simplicity, a node is mentioned interchangeably with its corresponding string in the remainder of the paper. Each leaf node has an inverted list of IDs of records that contain the corresponding word, with additional information such as the attribute in which the keyword appears and its position. For instance, Figure 3 shows a partial index structure for publication records. The word “vldb” has a trie node ID of 15, and its inverted list includes record IDs 6, 7, and 8. For simplicity, the figure only shows the record ids, without showing the additional information about attributes and positions.



Server architecture of instant-fuzzy search.

Figure 1: Proposed System

4. Conclusion

We studied how to improve ranking of an instant-fuzzy search system by considering proximity information when we need to compute top-k answers. We studied how to adapt existing solutions to solve this problem, including computing all answers, doing early termination, and indexing term pairs. We proposed a technique to index important phrases to avoid the large space overhead of indexing all word grams. We presented an incremental-computation algorithm for finding the indexed phrases in a query efficiently, and studied how to compute and rank the segmentations consisting of the indexed phrases. We compared our techniques to the instantfuzzy adaptations of basic approaches. We conducted a very thorough analysis by considering space, time, and relevancy tradeoffs of these approaches. In particular, our experiments on real data showed the efficiency of the proposed technique for 2-

keyword and 3-keyword queries that are common in search applications. We concluded that computing all the answers for the other queries would give the best performance and satisfy the high-efficiency requirement of instant search.

References

- [1] Cetindil*, J. Esmaelnezhad, Taewoo Kim and Chen Li and Irvine "Efficient Instant-Fuzzy Search with Proximity Ranking", Available: <http://www.imdb.com>, as of July 2013.
- [2] Cetindil, J. Esmaelnezhad, C. Li, and D. Newman, "Analysis of instant search query logs," in *WebDB*, 2012, pp. 7–12.
- [3] R. B. Miller, "Response time in man-computer conversational transactions," in *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, ser. AFIPS '68 (Fall, part I). New York, NY, USA: ACM, 1968, pp. 267–277. [Online]. Available: <http://doi.acm.org/10.1145/1476589.1476628>.
- [4] C. Silverstein, M. R. Henzinger, H. Marais, and M. Moricz, "Analysis of a very large web search engine query log," *SIGIR Forum*, vol. 33, no. 1, pp. 6–12, 1999.
- [5] G. Li, J. Wang, C. Li, and J. Feng, "Supporting efficient top-k queries in type-ahead search," in *SIGIR*, 2012, pp. 355–364.
- [6] R. Schenkel, A. Broschart, S. won Hwang, M. Theobald, and G. Weikum, "Efficient text proximity search," in *SPIRE*, 2007, pp. 287–299.
- [7] H. Yan, S. Shi, F. Zhang, T. Suel, and J.-R. Wen, "Efficient term proximity search with term-pair indexes," in *CIKM*, 2010, pp. 1229–1238.
- [8] M. Zhu, S. Shi, N. Yu, and J.-R. Wen, "Can phrase indexing help to process non-phrase queries?" in *CIKM*, 2008, pp. 679–688.
- [9] A. Jain and M. Pennacchiotti, "Open entity extraction from web search query logs," in *COLING*, 2010, pp. 510–518.