International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 02 Issue 11
November 2015

Instant Fuzzy Search Proximity Information by Novel Indexing Technique

Srivaishnav Gandhe[1], Snigdha Cheekoty[2]

[1]B-Tech Dept. of ECE Sreenidhi institute of science & technology, Hyderabad, Telangana
Mail Id: - srivaishnavg@gmail.com

[2]B-Tech Dept. of ECE Sreenidhi institute of science & technology, Hyderabad, Telangana,
Mail Id: - madcheekoty@gmail.com

Abstract— Instant search retrieves results as a user type's keyword character by character. On every keystroke result of previously typed prefixed query is used to generate result of newly typed query with one new character. We are using phrase threshold value which is used to limit the answer set generated by instant fuzzy search. For that main challenge is that to improve the performance as well as get answer set to retrieval of desired documents for the user query. At the same time, we also need better searching operates that look at the proximity of keywords to calculate relevance scores. In this paper, we study how to calculate proximity information with help of instant-fuzzy search while reaching efficient time and space complexities. A novel indexing technique is used to overcome the space and time restrictions of these solutions, we propose an approach that concentrates on common phrases in the database. The string generation algorithm based on pruning is assured to give the optimal top *k* candidates. The proposed method is utilized to reformulation of queries in web search and develops a computational algorithm for efficiently segmenting a query into phrases and computing these phrases using algorithm to find relevant answers to the user query.

Keywords: Instant Fuzzy Search, proximity information, a novel indexing technique.

1. INTRODUCTION

Information retrieval (IR) is the method of obtaining necessary information from a collection of large data set. User usually searches by providing the keyword or query. Queries can be a single or multiple keywords. In information retrieval, search for a query will not show single result instead many results which match the query will be shown. In Information Retrieval ranking the result set is very much important as the user will be interested in getting required information from first few documents of result set.

While entering characters there may be some typographical errors (typos), fuzzy search finds similar keywords and displays results for the predicted keywords. If the user wants to search for a documents containing keyword Sachin Tendlukar but by mistake he or she types Tendlukar then because of fuzzy search [1] the system will be able to retrieve the document containing Sachin Tendlukar. In case a query contains more than one term, then considering proximity [1] (the distance between keywords) is very important. To illustrate the importance of proximity let us consider the query "knowledge management". Systems that do not take proximity into account return general documents in which all the two terms knowledge and management are individually important, but the document does not necessarily contain information about knowledge management. On the other extreme an exact phrase match would make sure that the document retrieved matches the query but implementing such phrase matching search would result in not displaying many relevant results. A proximity ranking, ranks query results based on distance between query keywords.

Pre-processing is an important step for fast retrieval of search results. Pre-processing involves text extraction from data set files, stop word removal, stemming, unique words extraction and Index creation. Indexes are very important in any search engine. Combination of indexes like Tree Index[2] and Inverted List index[3] is used in current research. Tree Index is a special kind of trie; unique words are inserted into Tree Index which helps in fuzzy search. Inverted Index contains a word ID and list of Document ID i.e. all those documents which contains the word. Along with document ID list of position ID will be stored, i.e. where and all the word is present in the document. Storing position ID is necessary for ranking search results based on proximity ranking.

Fuzzy search is based on finding similar words from the dictionary. Levenshtein distance or edit distance in combination with Trie index finds similar words faster. Edit distance here refers to number of single character operations such as insertion, replacement or deletion need to be done in order to transform one word to another word. For example edit distance between "bin" and "pin" is one, since replacing character 'b' by 'p' word "bin" can be converted to "pin". Based on the length of the word a threshold for edit distance is deter-mined all similar words

within the threshold distance will be considered for fuzzy search.

Proximity ranking is implemented based on binning concept. Inverted index contains document ID which is associated with list of bin ID. The document is divided into bins, the number of bins varies from document to document but each bin contains equal number of words. For proximity ranking Inverted index is searched to find if two or more query words are within the same bin or in adjacent bin. If the query words are in same bin or adjacent bin the document is ranked higher otherwise the document is ranked lower.

2 PROBLEM STATEMENTS

Implementation of search system with advanced search features such as fuzzy search with proximity ranking. Existing search systems provide different kinds of ranking such as page ranking, ranking based on number of citations of the documents and ranking based on term frequency and inverse document frequency (tf-idf). Proximity ranking is very important since it determine the relevance of the answers as search queries usually contain related keywords and user is mostly looking for documents which have query keywords together.

3 RELATED WORK

Instant Query Search: Instant search retrieves results to the user as they types query character by character. For example, one database has a search interface that returns results to user while user typing a query character by character. When user types in “Laptop Customer Service”, then the system returns “Laptop Customer Service reviews ” , ” Laptop Customer Service comparison”, “Laptop Customer Services ranking”. This instant search technique provides user quick access of answers while typing instead of left in the blank until user types whole query and enter on search[1].

Fuzzy Keyword Search: When user makes typing mistakes in the query, then in this type the system can’t find the related answers by finding keywords in the database similar to query keyword. But by using fuzzy search this problem can be solved as the system finds answers to the query that are similar to the database keywords and not exactly same. Figure 1. shows an instant fuzzy search interface. The system finds answers to query “cloud” even though user mistyped a query as “cloud computing” then system retrieves result[3].

International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 02 Issue 11
November 2015

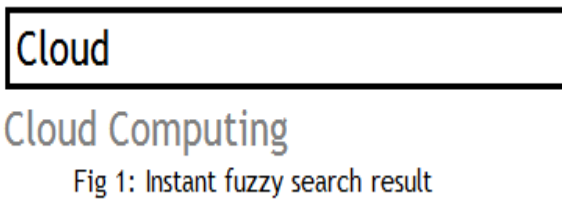

Fig 1: Instant fuzzy search result

Time Limit on Retrieving Answers: In current era the challenging part in searching is required very high speed. Results needs to retrieve within some milliseconds only. This time again includes the time required over the network, time taken at server and time taken by browser to execute & show results. So, our main aim to provide high speed requirement by reducing the time required at server side for finding results in database [2]. More challenging here to use perfect database and algorithms which will reduces time required to search in database as server have lots of data and searching sequentially or by old methods of search it's difficult to achieve high speed requirement.

Proximity Ranking: Proximity ranking means the document that are more correlated with query words are provided at higher in result. So it provides more efficient top results [9]. Studies related to this improves query efficiency by using early-termination techniques [10],[11].

4 PROPOSED METHOD

4.1 Proposed system Architecture

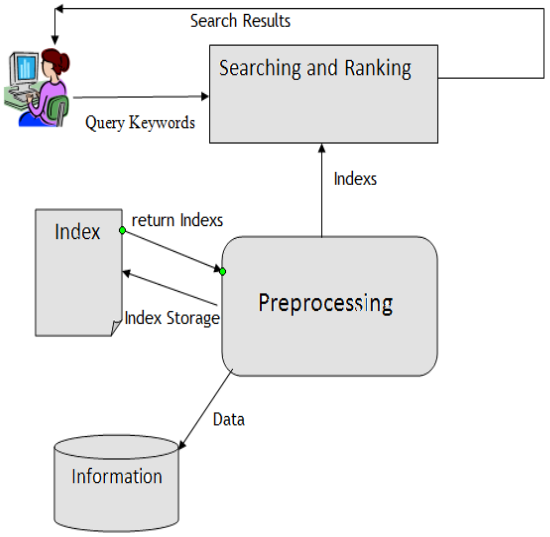

Fig. 2. System Architecture

Fig. 2. Shows the proposed system architecture, pre-processing module is meant for creating and retrieval of indexes. Searching and ranking module is responsible for searching suitable documents using query keywords and indexes, this module ranks documents based on proximity distance between query keywords. User uses the system to search relevant documents. File system stores data set as well as index files. Data-base holds abstract information of each file.

Preprocessing

Text data from each of data set file is read special characters are ignored, stop words [14] are ignored, duplicate words are

ignored and stemming [15] is performed. Words are arranged alphabetically. Each unique word is given a word Id. The collection of word Id and word name corresponds to Dictionary. Dictionary will be stored in file system during this phase. It will be retrieved to main memory during the start of search server. Dictionary will also be represented as Trie data structure since using Trie data structure helps in faster retrieval of similar words for fuzzy search. Trie data structure will be stored in file system during this phase. It will be retrieved to main memory during the start of search server. During this phase an inverted list is created by reading in each file of data set word by word if the word in the dictionary exists then in which bin the word falls will be noted. The bin position helps in proximity ranking. The inverted index is created as a map-ping between word Id and list of document Id's i.e. the documents in which the word exists and list of bin Id's i.e. the bin position at which the word is present in the document

4.2 Rule Index

The rule index storages all the rules plus their weights using an Aho-Corasick tree (AC tree) [13], which can make the references of rules very efficient. The AC tree is a trie with "failure links", on which the Aho-Corasick string corresponding algorithm can be performed. The Aho-Corasick algorithm is a well-known dictionary-matching algorithm which can rapidly locate the elements of a finite set of strings (here strings are these in the rules $\alpha \rightarrow \beta$) within an input string. The time complexity of the algorithm is of linear order in the length of input string plus the number of matched entries. We construct an AC tree using all these within rules. Each leaf node corresponds to an α, and the matching βs are stored in an associated list in decreasing order of rule weights, as illustrated in Fig. 3. One may want to further improve the efficiency by using a trie rather than a ranking list to store the βs associated with the same α. However the improvement would not be significant because the number of βs associated with each α is usually small. In string generation, given an input string, we first retrieve all the applicable rules and their weights from the AC tree in time complexity of input string length plus number of matched entries.

International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 02 Issue 11
November 2015

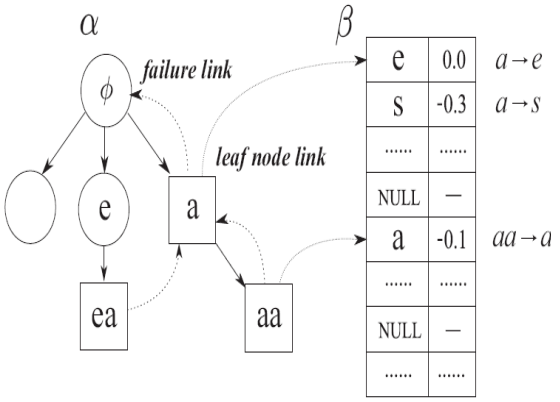

Fig 3: Rule Based Index AC Tree

The entire set of documents which have all query keywords or words similar to query keywords are considered for ranking. The model consists of rules and weights. A rule is formally represented as $\alpha \rightarrow \beta$ which denotes an operation of replacing substring α in the input string with substring β, where $\alpha, \beta \in 2\{s|s = t ; s = ^t; s = t\$$, or $s = ^t\$\}$ of possible strings overthe alphabet, and ^ and $ are the start and end symbols respectively.

Algorithm Name: search

Input:

Query Wordlist, // List of query keywords

Output:

Search Result

Steps:

1. Remove stop words from query word list
2. Apply stemming to each query keyword
3. For each keyword in query Word List
4. Find threshold edit distance
5. Similar Word List = find Similar Words(keyword, node, threshold)
6. Documents with phrases = proximity Ranking (similar-Word List, word Id List) .
7. For each keyword
8. Find documents without phrases
9. Result = (documents with phrases) union (documents without phrases)

Above algorithm search, is used to search relevant documents. Initially preprocessing of query keywords is done, this involves removal of stop words from the keyword list. For each query keyword stemming is performed. To find list of similar words to each query keyword threshold distance is calculated based on length of query keyword. Similar words are found using AC Tree.

5 EXPERIMENTAL RESULTS

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.

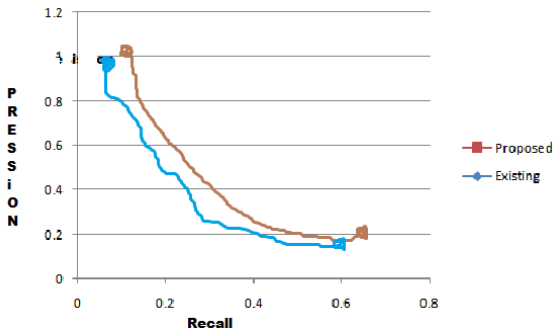

Fig 4: Graph representing Average interpolated precision versus recall of Existing & Proposed Sysetm..

6 CONCLUSIONS

In this paper 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 index store all the rules and their weights using an Aho-Corasick tree. 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.

7 REFERENCES

[1]I. Cetindil, J. Esmaelenzhad, C. Li, and D. Newman, " Analysis of instant search query logs,"in WebDB,2012,pp.7-12.

[2]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). NewYork, NY, USA: ACM, 1968, pp. 267– 277.

[3] G. Li, J. Wang, C. Li, and J. Feng, "Supporting efficient top-k queries in type-ahead search," in SIGIR, 2012, pp. 355–364

[4] M. Persin, J. Zobel, and R. Sacks-Davis, "Filtered document retrieval with frequency-sorted indexes," JASIS, vol. 47, no. 10, pp. 749–764, 1996

[5] A. Singhal. "Modern information retrieval: A brief overview." Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[6] C. Silverstein, m. R. Henzinger, H. Marais, and M. Moricz, "Analysis of a very large web search engine quearylog," SIGIR Forum. Vol. 33, no. 1, pp. 6-12, 1999.

[7] A. Nandi and H. V. Jagadish, "Effective phrase prediction," in VLDB, 2007, pp. 219–230.

[8] H. Bast and I. Weber, "Type less, find more: fast autocompletion search with a succinct index," in SIGIR, 2006, pp. 364–371.

[9] R. Song, M. J. Taylor, J.-R. Wen, H.-W. Hon, and Y. Yu, "Viewing term proximity from a different perspective," in ECIR, 2008, pp. 346–357.

[10] R. Schenkel, A. Broschart, S. won Hwang, M. Theobald, and G. Weikum, "Efficient text proximity search," in SPIRE, 2007, pp. 287– 299.

[11] M. Zhu, S. Shi, M. Li, and J.-R. Wen, "Effective top-k computation in retrieving structured documents with term-proximity support," in CIKM,2007, pp. 771–780.

[12] S. Ji, G. Li, C. Li, and J. Feng, "Efficient interactive fuzzy keywordsearch," in *WWW*, 2009, pp. 371–380.

[13] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," Commun. ACM, vol. 18, pp. 333–340, June 1975

[14] Stopword Lists "http://www.ranks.nl/stopwords"

[15] The Porter Stemming Algorithm "http://tartarus.org/martin/PorterStemmer/ "