Instant Fuzzy Search Engine with Phrase Based Ranking

Kate Sangita Rajendra, Gore Sneha Subhash, Sayyad Laila Mahamud, Solankar Punam Vitthal.
Department of Computer S.V.P.M.C.O.E. Malegaon(Bk), Baramati Pune, India.
Mrs. Dabhade A.S.(Assi. Prof.) Mrs. Sawant V.V.(Assi. Prof.)

ABSTRACT—Instant search retrieves results as a user types 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. Also to make instant search result computation is done incrementally. Fuzzy search allows user to type query in the fly even he don't have more information about search. Autocompletion provides suggestions while user types query, it provides way for what to type next. Main problem of retrieving quick result is solved in this paper by using efficient trie data structure with efficient trie search technique. More relevant answers generation is based on phrases, so ranking of result becomes efficient. The records with exact phrases in the query are ranked higher to give more efficient result to user.

Keywords: Proximity Ranking, Autocompletion, Phrases, Fuzzy search, Incremental Computation, Trie, Active Nodes.

1. INTRODUCTION

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 “computer” even though user mistyped a query as “conpute” then system retrieves result[3].

```
computer
```
Result : computer

Fig 1.1. Instant Fuzzy Search.

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.

2. LITERATURE SURVAY

1. **Instant Search**: Instant search is also called type-ahead search. Recent studies contains techniques and algorithms to support instant search in [4]. This was query techniques. To overcome the problems related to efficiency new system is presented in [5]. The previous systems work on the relational databases which requires more time even name is instant search. This is avoided by using trie data structure in our system.

2. **Auto-Completion**: Previous studies provides auto-completion results i.e. these provides suggestions while typing query in case ‘what to type next’. There are many studies on auto-completion which includes paper [6]. This suggestions include queries that starts with the partial query that user typing as prefix. This feature is extended here to make system user friendly.

3. **Fuzzy Search**: Fuzzy search allows user to don’t worry about mistakes while typing query. The studies related to fuzzy search classified into two types gram-based technique and trie based technique. In gram-based approach sub-strings of query are used for fuzzy search to match with documents in database [7],[8]. Our system is phrase based so only the continuous sub-strings are used as phrase.

4. **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].

3. PROPOSED SYSTEM

3.1 Terms:

**Data**: Let R= <r1,r2,…rn> be a set of records which contains tuples in a relational table. D be the dictionary which contain all the different words in R. Table I shows example of Book records. Each records has text attributes such as title of the book.

**Query**: A query Q is a string which contains a no of keywords W= <w1,w2,w3…..wl>. In instant search request is send to the server for each keypress. When user types a query character by character, each query is constructed by appending one character at the end of earlier query. For example when user types “language” keyword character by character the server receives the following queries one by one q1=<l>, q2=<la>,q3=<lan>,q4=<lang>……… q8=<language>.

**Answers**: A result set is a records r from the data set R. For example: for the query q=< java, security > the answers of this query is records r1, r2 ,r3, r4, r6 because all these record contain the keyword “java”. The similarity between two keywords can be checked by using various methods such as edit distance, cosine similarity. For example, the edit distance between the keywords “clear” and “clean” is 1, because the former can be transformed to the latter by substituting the character “r” with “n”. 

**ISRJournals and Publications**
**Ranking:** Ranking the answer of the query is based on the relatedness of the query with the keywords in the dictionary \( D \). For example, for the query \( q = \langle \text{java,security} \rangle \) record \( r_1 \) in Table 1 containing a phrase “java security” is more related than the record \( r_1 \) containing the keyword “java” and “security” separately.

### 3.2 System:

To overcome the restriction of previous systems, the new technique that is based on phrase based indexing has been developed. A phrase is a sequence of keywords that has high probability to come in the records and queries. The answer of the query having a matching phrase in the query that has high score than the query without matching phrase. We want to access the records which includes phrases first.

![Trie with inverted list at leaf nodes](image)

**Fig 2.1:** Trie with inverted list at leaf nodes

<table>
<thead>
<tr>
<th>Book name</th>
<th>Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>( r_1 )</td>
</tr>
<tr>
<td>Java languages</td>
<td>( r_2 )</td>
</tr>
<tr>
<td>Java security</td>
<td>( r_3 )</td>
</tr>
<tr>
<td>Java service</td>
<td>( r_4 )</td>
</tr>
<tr>
<td>GUI programming</td>
<td>( r_5 )</td>
</tr>
<tr>
<td>Computer security</td>
<td>( r_6 )</td>
</tr>
</tbody>
</table>

For example, for the query \( q = \langle \text{java,security} \rangle \), we need to access records which includes the phrase “java security” before the records containing “java” and “security” separately. The inverted list attached at the end of the leaf node which is sorted list based on relevancy of the keywords. Order of inverted list for keyword “security” based on the relevancy to the phrase “java security”, the best processing order for a other phrase, say, “computer security”, “java servlet” may be different. Fig 2.1 shows trie with inverted list attached at the end of leaf for data in Table 1. For example, the phrase “java security” is indexed in the fig 2.1 the leaf node for the keyword “java” points to the inverted list \( r_1, r_2, r_3, r_4, r_6 \).
4, SYSTEM ARCHITECTURE

Fig 3.1. Server Architecture of Instant Fuzzy Search.

Above figure shows the server architecture of instant fuzzy search for one database server. In this architecture book database is used as a database server. Admin is responsible for uploading books on the server, when book uploaded on server trie has been built and inverted list is prepared. The input to the system is a query i.e typed characters and output of the system is a records.

When server receives request from client then server identifies all phrases in the query that are in the database. For identifying valid phrases in the query we have a module called as phrase validator. For example for the query q= "java, security > “java” is a valid phrase for the data set in Table 1, in addition “security” and “java security” is also valid phrase. Phrase validator computes all valid phrases and returns active nodes for all these phrases i.e. If query keyword come in to multiple valid phrases then query can be segmented into phrases in distinct ways. For example, “java | security ” and “java security” are two distinct segmentation for query q. After identifying valid phrases Query Plan Builder generates a Query Plan Q, which includes all the possible valid segmentations in a definite order. After query plan is generated, the segmentations are passed to the Index Searcher one by one until the top-k answers are computed. Index searcher computes top-k results by using segmentation. Combining the result set of query plan and segmentation the final. For storing computed result of the previous queries that can be use for expedite the computation of later queries for that reason cache module is used. The phrase validator uses cache module for validate phrases without traversing whole trie, while Index Searcher uses cache for retrieving result of the earlier queries to reduce computation cost.

5, ALGORITHM

1. Incremental Computation Of Valid Phrases:

The subsequent queries of the user differ from each other by one character and their computation of valid phrase have lot of overlap. For incrementally computing valid phrases of a query qj cached vector is used which stores the valid phrases of a previous query qi.

Fig 4.1 shows the active nodes of the valid phrases queries q1 = <compute > q2 = < computer > and q3 = < computer , security >. For the phrase “computer” q1 and q2 have the active nodes n1 and n2. For q2 has an active
node \( n_2 \) for the phrase “computer”, which is closed to active node \( n_1 \) of the phrase “compute” in \( q_1 \). Hence to compute the \( n_2 \) efficiently we can use the active node \( n_1 \).

All the active nodes of \( q_2 \), i.e. \( n_1, n_2 \) are also active node for \( q_3 \). \( q_3 \) has active node \( s, n_3 \) and \( n_4 \) for the keyword “security”, i.e. “security” and “computer security”. “computer security” has a phrase from \( q_2 \) as a prefix and from \( n_2 \) its active nodes can be computed incrementally.

Algorithm 1 is for computing valid phrases of a query using previously valid phrase vector.

Fig 2 shows valid phrase vector which is used for incrementally computation of valid phrases. \( v_1 \) is stored in the cache then \( v_2 \) can be computed incrementally from \( v_1 \) as follows:

If the first keyword is same then as it is copied into (lines of 6. in the algorithm for copying the vector). The first element of \( v_2 \) is computed incrementally starting from active node set \( S_{1,1} \) in the first element of \( v_1 \) (lines of 8-10 in the algorithm). The incremental computation of \( v_3 \) from \( v_2 \) in this case where there are additional keywords in the new query. In this case the first keyword copied as it is from \( v_2 \) to \( v_3 \) . We compute the second element of \( v_3 \) from active node set of \( v_2 \) (lines of 11. in the algorithm used for third case).

Algorithm for Incremental Computation of Valid Phrase Vector:

Input: \( new\_q=\{w_1, w_2, \ldots, w_N\} \)
\( N \): no. of keywords. ANS: Active Node Set
Output: \( new\_V \) – Valid Phrase Vector
1. \( Q\_Log \) = Read Query Log.

Above figure is for the queries \( q_1, q_2 \) and \( q_3 \) with valid phrase vector \( v_1, v_2 \) and \( v_3 \) respectively. For example the third element of \( v_3 \) shows the starting points of all the valid phrases ending with a prefix similar to “computer”.

Fig 4.1 Active nodes for valid phrases

Fig 5.1 computing valid phrases incrementally using cache
2. \((old_q, old_V) = \text{getLongestPrefix}(Q_{Log}, new_q)\)
3. total = no. of words in old_q
4. All = no. of words in new_q
5. If (total > 0)
6. For i=0 to total
   New_V[i] = old_V[i];  //Copy ANS for same phrases
7. If(old_q==new_q)
   Return old_V;
8. else  //compute ANS for cached phrases incrementally
   new_V[total] = \text{Find_ANS}(old_V[total], new_q[0] to new_q[total]);
9. new_ANS = new_V[total]
10. if(All>total)
    for i=total+1 to All
        new_ANS = \text{Find_ANS}(new_ANS, new_q[0] to new_q[i]);  //by appending new_q[i] every time
        Union(new_V, new_ANS);
11. //compute ANS for non-cached phrases
    Initial_ANS = \text{compute_initial}(root, threshold); for i=total+1 to All 
        new_V[i] = \text{Find_ANS}(initial_ANS, new_q[i]);
        new_ANS = new_V[i]; 
        for j=i+1 to All
            new_ANS = \text{Find_ANS}(new_ANS, new_q[i] to new_q[j]);  //by appending new_q[j] every time
            Union(new_V[j], new_ANS); }
12. Return new_V.

6. EXPERIMENTAL

RESULT

1) Phrase Computation From Book Title:

<table>
<thead>
<tr>
<th>Book Title</th>
<th>Phrases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Servlet</td>
<td>java, servlet, java servlet</td>
</tr>
<tr>
<td>Java Security</td>
<td>java, security, java security</td>
</tr>
<tr>
<td>Java Program</td>
<td>java, program, java program</td>
</tr>
</tbody>
</table>
2) Trie Build from Computed Phrases:

<table>
<thead>
<tr>
<th>Book Title</th>
<th>Builted Trie</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Java Servlet</td>
<td>![Trie Diagram for Java Servlet]</td>
</tr>
<tr>
<td>2. Java Security</td>
<td>![Trie Diagram for Java Security]</td>
</tr>
<tr>
<td>3. JavaProgram</td>
<td>![Trie Diagram for JavaProgram]</td>
</tr>
</tbody>
</table>
VII. CONCLUSION

In this paper we studied how to improve performance in case of time and space requirement in instant-fuzzy search. We proposed a system which cover the space & time efficiency problems in previous gram based system by introducing new phrase based ranking algorithm. We studied how to use and extend previous techniques to overcome the problem including early termination, using inverted list of document, ranking top-k results.

REFERENCES


277.


