ISSN:2320-0790

An International Journal of Advanced Computer Technology

Improved Fuzzy Technology for Efficient Searching

Mr. Rajendra P. Sabale[1], Prof. Amruta Amune[2]
Computer Enggineering Department
G. H. Raisoni College of Engg. And Management,
Chas, Ahmednagar.

Abstract: Instant searching technique finds answers to a query instantly when user types in keywords character-by-character. Fuzzy searching is advancement in instant searching which finds perfect matching keywords to query keywords. User expects fast results within few milliseconds and perfect match. The main challenge in this is the high-speed requirement, i.e., each query needs to be answered within milliseconds to achieve an instant response and a high query throughput. Number of fuzzy techniques such as compute all, early termination technique and segmentation technique are being used. Each of them as various difficulties that they requires more time to search in case of large data and space required is also more. The accuracy is also less as they are not providing relevant answers. In this paper I will be studying available techniques and new proposed algorithm. It will be compared with existing techniques with regard to time, space and relevancy.

Keywords: instant searching, fuzzy searching, computing all, segmentation.

I. Introduction

Instant Searching: It is a way of searching in which the answers are returned when the user types the partial query character by character. Queries can be formed accordingly by seeing the results, which will be helpful to users in forming queries.

Fuzzy Searching: Flat fingers problem can be solved
by using fuzzy search, where Users number of times makes mistakes while typing queries due to flat fingers and small keyboards like smart mobiles, lack of attention. In this case we cannot find perfect answers.

Finding Perfect Answers in less Time:
A main requirement in this search is its high speed .Users expects fast result, from the time a user types in to the time the results are shown on the device, the total time should be within few milliseconds.

II. Problem Statement

In this paper, we study to find the perfect answers efficiently by using instant-fuzzy searching. The probability of matching keywords in answers is very important to determine the accuracy of the answers. Search queries normally contain correlated keywords as well as answers that have these keywords together are mostly what the user desires.

We will study number of ways to this problem and show the merits and demerits with respect to space, time, and answer quality. One of the techniques is to compute all i.e. First of all find all the answers, find the score of each answer based on a ranking function, sort them using the score, and return the top results. However, enumerating all these answers can be computationally difficult and costly when these answers are more in number.

III. Previous Work

Auto-Complete: It provides number of queries the user may type in next. Lot of studies which predicts it [9] [10] is done already. Many of the system do prediction by treating a query with more keywords as a single prefix string. Therefore, if a suggestion has the query keywords but not consecutively, then this suggestion cannot be known.

Instant Searching:

It is also called as type-ahead search. The studies in [11] [12] [13] proposed indexing and query techniques to support instant search and [14] [15] provides trie based techniques to solve this problem.

Fuzzy Searching:

It can be classified into gram-based approaches and trie-based approaches. In this approach, sub-strings of the data are used for fuzzy string matching [17][18] [19] [20] and the other way indexes the keywords as a trie, and depends on a traversal on the trie to find similar keywords [14] [15]. This approach is useful for instant and fuzzy searching

[14] since each query is a prefix and trie can support incremental computation efficiently.

My work differs from the earlier studies since we focus on how fast to compute perfect answers based on new proposed algorithm.

IV. PRELIMANARIES

Data: Record p = {p1, p2,...,pn} be a set of records with text attributes, such as the tuples in a relational table or a collection of documents.

D is the list containing all the different words of p .Each record has text attributes.

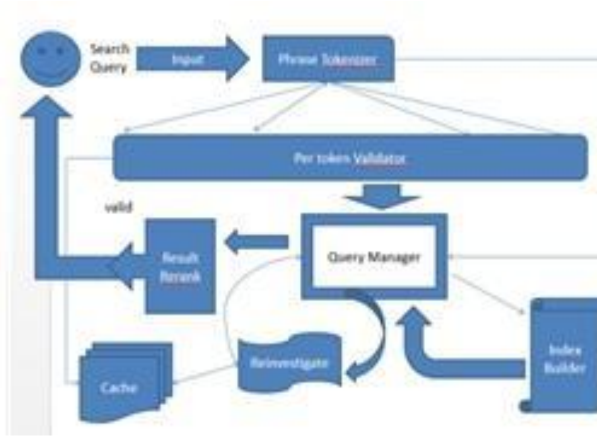
Query: **que** contains a list of keywords. que= (q1, q2,...,qn) separated by space.

In an instant-searching system, a query is submitted for each keystroke of a user. When a user types in a string character by character, each query is constructed by appending one character at the end of the previous query. The last keyword in the query represents the word currently being typed, and is treated as prefix.

Ans: A record from the data set R is an answer to the query q.

V. PROPOSED SYSTEM ARCHITECTURE

The proposed system consist of various modules such as search query, Phrase Tokenizer, Per Token Validator, Index builder, Query Manager, Reinvestigator, Resultrerank.



Modules Description

- **Search Query:**

This module is meant for accepting input in the form of text. The user will enter data which is related to the results he wants from the system.

- **The Phrase Tokenizer:**

The query entered by the user will be input to this module. The query will be divided. Decomposition will help in optimization by generating tokens of search query.

- **Per Token Validator:**

As phrase tokenizer generates tokens, these tokens will be forwarded as input to the Validator. The validator will check whether to store this data into cache or not and forwards the query to query manager.

- **The Index builder:**

It will take training data to be searched as input. It will build an index on this data and these indices will help in fast searching and getting search results.

- **The Query Manager:**

It comes across the center of the system. It is responsible for building query plan. It manages input query, output results and caches.

The Reinvestigator:
It does the main optimization task. It will use our proposed algorithm to search fastly from cached data and send the results to query manager.

- **Resultrerank:**

Input to this module is output of query manager. Initial results will be input to this module. This module will do rearranging to the results depending on our proposed algorithm and ultimately user will get efficient results at the top.

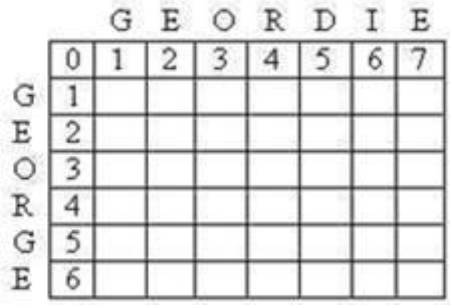
VI. PROPOSED ALGORITHM

This algorithm counts the number of changes that must occur in one string to transform it into another string. Consider two strings, "George" and "Geordie" . Find the number of characters required to change to transform “George” into "Geordie"? It will be two characters.

1.”George “? “Georde “ (replace 'g' with 'd').

2.”Georde” ? “Geordie “ (add 'i' between the 'd' and 'e').

The actual algorithm itself is very simple, requiring a matrix to represent the values as the calculation progresses through both strings. A simple version of the loops results in an O (n2) Implementation, but since only very small strings (under 10 characters typically) existed in the problem space. Before getting into the loop to calculate the Algo, check to see if either string is empty. If not single string is empty, then construct the matrix.. The matrix should be the length of the first input string +1 into the length of the second input string +1.



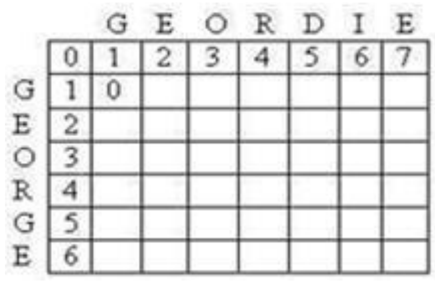
		G	E	O	R	D	I	E
	0	1	2	3	4	5	6	7
G	1							
E	2							
O	3							
R	4							
G	5							
E	6							

Figure 1: Initial Matrix

While executing the strings, the indexes of both strings are the coordinates of the matrix. The actual calculation is interesting as it observes at three cells, the cell to the left, the cell above and the cell to the upper left. The point of the three comparisons is to take the small value from these cells; this can be represented simply in Java with a single line of code.

Once the matrix is formed, the remaining of the values may be filled. After the matrix is filled , the answer will be in the lower right hand cell.

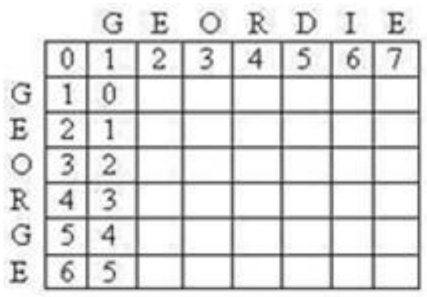
For each time either 0 or 1 is the result. If characters under consideration are same, then the result is 0; if they are different, the result is 1.



		G	E	O	R	D	I	E
	0	1	2	3	4	5	6	7
G	1	0						
E	2							
O	3							
R	4							
G	5							
E	6							

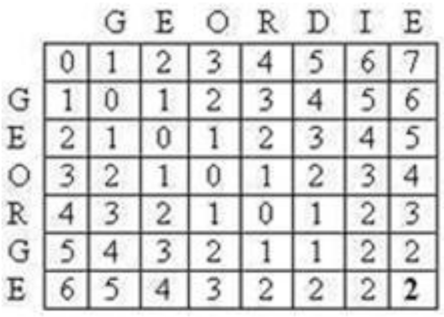
Matrix after the first iterative execution will as above .

The result of the first pass would then be as follows:



		G	E	O	R	D	I	E
	0	1	2	3	4	5	6	7
G	1	0						
E	2	1						
O	3	2						
R	4	3						
G	5	4						
E	6	5						

After complete execution the will be as follows:



		G	E	O	R	D	I	E
	0	1	2	3	4	5	6	7
G	1	0	1	2	3	4	5	6
E	2	1	0	1	2	3	4	5
O	3	2	1	0	1	2	3	4
R	4	3	2	1	0	1	2	3
G	5	4	3	2	1	1	2	2
E	6	5	4	3	2	2	2	**2**

Algorithm:

```
q =get query
D = initialize dictionary
for each(word in q )
{
If(word ∩ D)
{
Word = d1
}
Else
{
Continue
}
for each(word in synonyms
s )
{
If(word ∩ s)
{
Word = s1
}
Else
{
Continue
}
for each(word in stop st)
{
If(word ∩ st)
{
Word = st1
}
Else
{
Continue
}
```

```
// Toknize string
q[] = tokenize(q,' ');
for each qtemp € q[i]
forward for preference search
End of for
```

VII. MATHEMATICAL MODEL

S will describe total system

S= {r,u,f}

Here

Details of each element given below:

Input Set:

Input={q}

q:query

Output Set:

Output={r1,r2,r3}

r={ac,is,fs} where

r :result generated by system.

ac:generated by system

is:generated by system

fs:generated by system

u={pr,v,pb,c} where

u :uses of system

pr:proposed system(forword for consideration).

v:validation

pb:plan builder
c:cache

f={i1,s2,r1} where

f:function of system.

i1: indexer functionality

s1: search functionality

r1: reranking

The elements in set r where

ac=autocomplete

is=instant search

fs=fuzzy search

VIII. RESULTS

In this section, I will evaluate the performance of the system by using IMDB data set. IMDB data set can be obtained from their website. The data set contains movies, characters tables and constructed the table. I will evaluate the performance by using: (1) auto complete (2) instant searching**.** **(**3) Fuzzy searching. The performance will be compared with respect to time, space and accuracy of result.

1. Performance finding with respect to time.

Table 1: Performance finding w.r.t time.

Method	Autocomplete	Instant	Fuzzy
No. of Keywords	22	34	56

2. Performance finding with respect to memory size.

Table 2: Performance finding w.r.t memory size.

Method	Autocomplete	Instant	Fuzzy
Size of memory (MB)	2	1.5	1

3. Performance finding with respect to accuracy.

Table 3: Performance finding w.r.t accuracy.

Method	Autocomplete	Instant	Fuzzy
Accuracy in percentage	57	69	88

CONCLUSION

In this paper we studied how to find results by using auto completion, instant search and fuzzy search. Compared the performance with respect to time, space and accuracy.

REFERENCES

[1] Cetindil, J. Esmaelnezhad, C. Li, and Chen Li. “Efficent instant fuzzy search with proximaty ranking” in IEEE, July 2014.

[2] I. Cetindil, J. Esmaelnezhad, C. Li, and D. Newman, “Analysis of instant search query logs,” in WebDB, 2012, pp. 7-12. Adding Persuasive features in Graphical Password to increase the capacity of KBAM, Uma D. Yadav, Prakash S. Mohod Computer Science & Engineering G. H. R. I. E. T. W. Nagpur, India

[3] 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.

[4] G. Li, J. Wang, C. Li, and J. Feng, “Supporting efcient top-k queries in type-ahead search,” in SIGIR, 2012, pp. 355-364.

[5] R. Schenkel, A. Broschart, S. won Hwang, M. Theobald, and G. Weikum,“Efcient text proximity search,” in SPIRE, 2007, pp. 287- 299.

[6] H. Yan, S. Shi, F. Zhang, T. Suel, and J.-R. Wen, “Efcient term
proximity search with term-pair indexes,” in CIKM, 2010, pp.
1229- 1238.

[7] H. Bast and I. Weber, “Type less, nd more: fast autocompletion search with a succinct index,” in SIGIR, 2006, pp. 364-371.

[8] H. Bast and I. Weber, “The completesearch engine: Interactive, efcient, and towardsirdb integration,” in CIDR, 2007, pp. 88-95.

[9] S. Ji, G. Li, C. Li, and J. Feng, “Efcient interactive fuzzy keyword search,” in WWW, 2009, pp. 371-380.

[10] S. Chaudhuri and R. Kaushik, “Extending autocompletion to tolerate errors,” in SIGMOD Conference, 2009, pp. 707-718.

[11] G. Li, S. Ji, C. Li, and J. Feng, “Efcient type-ahead search on relational data: a tastier approach,” in SIGMOD Conference, 2009, pp. 695-706.39

[12] M. Hadjieleftheriou and C. Li, “Efcient approximate search on
string collec- tions,” PVLDB, vol. 2, no. 2, pp. 1660-1661, 2009.

[13] K. Chakrabarti, S. Chaudhuri, V. Ganti, and D. Xin, “An efcientlter for approximate membership checking,” in SIGMOD Conference, 2008, pp. 805-818.

[14] S. Chaudhuri, V. Ganti, and R. Motwani, “Robust identication of fuzzy du- plicates,” in ICDE, 2005, pp. 865-876.

[15] Behm, S. Ji, C. Li, and J. Lu, “Space-constrained gram-based indexing forefcient approximate string search,” in ICDE,
2009, pp. 604-615.

[16] Fagin, A. Lotem, and M. Naor, “Optimal aggregation algorithms for mid- dleware,” in PODS, 2001.

[17] F. Zhang, S. Shi, H. Yan, and J.-R. Wen, “Revisiting globally sorted indexes forefcient document retrieval,” in WSDM, 2010, pp. 371-380. [23] M. Persin, J. Zobel, and R. Sacks-Davis, “Filtered document retrieval with frequency- sorted indexes,” JASIS, vol. 47, no. 10, pp. 749-764, 1996

[18] R. Song, M. J. Taylor, J.-R. Wen, H.-W. Hon, and Y. Yu, “Viewing term proximity from a diff erent perspective,” in ECIR, 2008, pp. 346-357.