ASCII based Technique of Obtaining Data Set to Perform Fuzzy Keyword Searching

Pooja Prashar
Amity School of Engineering & Technology
Sector 125, Noida
Uttar Pradesh, India

Shivanku Mahna
San Jose State University
1 Washington Sq
San Jose, CA, 95192

ABSTRACT
The world we live in is full of amazing art of science, where the role of technology is highly important. Hence, in such a conducive environment, major break-throughs are bound to happen. One such major development which has brought lot of positive results in the current scenario is how we save our information. Earlier, we used to save our data in tape based hard-disks and floppy drives, whereas now we are saving our data in some remotely located server or cloud based servers. However, it is quite evident that, with a positive aspect, there is always a drawback associated with it. One major drawback associated with the current practice of saving our data which cannot be ignored is situating the confidential files and folders at the risk of being exposed to unwanted people, so much so that the precious data loses its confidentiality and integrity. Another huge problem that arises with the usage of cloud is that, even in this day and age of artificial intelligence, where we are trying to predict and understand well in advance what other person is trying to say; searching on cloud is still not typo friendly. In this paper, we have tried to analyze the already existing fuzzy type searching algorithms and provide a completely innovative and more effectual method that aims at making searching typo friendly and efficient in time & space on any platform , be it windows, local hosting servers, cloud or backend servers. The ASCII based algorithm that we are going to suggest, will have its focus on finding a method that provides results in effective & efficient manner and yet does not compromises on the data integrity and confidentiality of the data being searched or being stored on the cloud/server. In order to do so, we have also used the encryption and decryption techniques like AES algorithm which is very safe and efficient. Making a system friendly to fuzzy type keyword searching will enhance its usability and make it typo friendly. We shall be using ASCII coding in order to quantify keywords similarity and for constructing fuzzy keyword sets.

General Terms
Cloud, Encryption, Decryption, Gram Based, Wild Card.

Keywords
AES, Fuzzy type keyword, ASCII

1. INTRODUCTION
With so much of advancement happening in every sector, it is extremely hard for an individual to keep themselves abreast of the latest developments in their surroundings. There is one sector which has had a lot of development happening and has brought a lot of ease in our lives, and that is technology. With the technology bloom happening, therein comes a new range of more efficient options for us to choose from. If we consider this in the context of the way we are saving our ever increasing data, then we surely have come a long way since the beginning of computing era. In the older times, it was extremely hard to manage our records. The task of maintaining them and keeping them up to date was extremely burdensome. Since the records were maintained physically, it became really difficult for the changes being made to certain data be known to rest of the other people associated with it. Then, the technology came to rescue and data was being stored in tape based hard disks and floppy drives, but they also were either too bulky or too short on space in most cases. Hence a better solution was required. The technology of modern days allows us to maintain all our files and information on cloud. Now we can store all our information as if it is being stored on a data server which is located in a different country. It can also play the role of an achieved copy and replica, which can be retrieved if the center is damaged due to any reason. But one thing that is yet to be a little more developed or enhanced is the searching techniques which are being used today. Today, if we want to search a file on a computer or a server, then we need to enter the exact same name of the file in order to get the file retrieved. If, by chance, we have a typographical error while typing in the name of the file (which is a highly probable case in today's time of fast typing), then no result is retrieved. It is common for people to have typos while typing and thus the method currently in place in cloud or computer systems is inefficient in processing our language and realizing what we wanted to find while making that typo and as a result, no file is retrieved. As an example, we can consider the case taken for the screenshots attached below. If we try and search for a file named ‘Pooja’, then the system returns a lot of files but when we try and search for a file named ‘Pooja’ but, by mistake, make a typographical error and type in Pooja as ‘oooja’, then nothing is returned. It is hard to believe that in today's day and age of artificial intelligence, where we are trying to understand all the possible cases and cover the maximum possible outcomes, even before they have occurred, the searching techniques on windows, cloud or local host servers is still not able to understand what we are trying to search for and returns back unfavorable or 0 results. We, as a result, decided to dig deep into this problem to obtain a solution and finally, after a lot of research work and calculations, were able to obtain a method using which we can make searching more efficient and more friendly to typographical errors. We have also analyzed all the different possible fuzzy searching algorithms which are currently available for use and then, developed one of the most efficient algorithm in today’s day and age, which makes the cloud typo friendly, enhanced the results computing power by drastically reducing the time being used to calculate and search for the desired results and last but not the least, do all this, with minimum memory and space consumption. The section following this has listed a few of the earlier methods, which are being used till date at some of the places and we, while explaining that, have tried to explain why each of them is more inefficient than the one we are suggesting in the upcoming sections post section 2.0 [1].

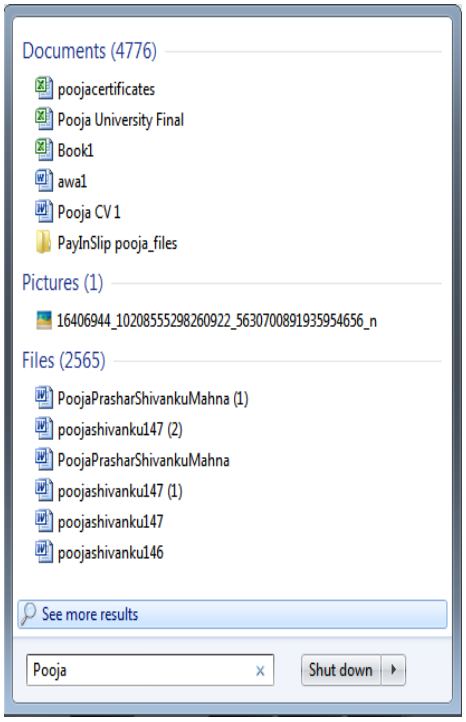


Fig: 1.1 Search Results that we get on searching for Pooja.

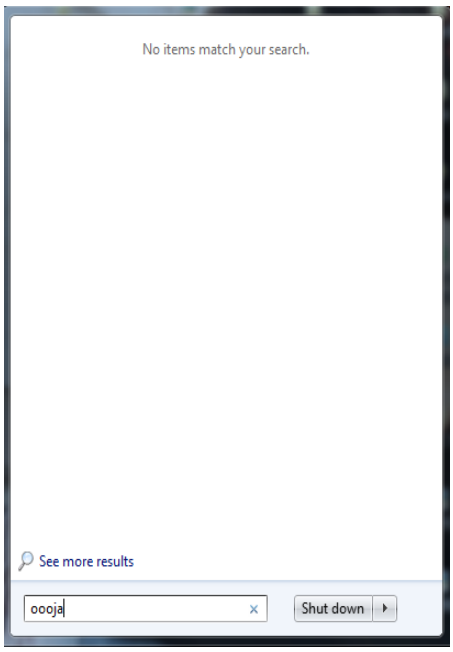


Fig: 1.2 Search Results that we get on searching for oooja (Pooja with typo error).

2. CONCEPT OF KEYWORD NAMING

A cloud is not only used by people for personal use but also used by companies or associations to save their highly classified and secret data, losing which can cause a loss of millions. Hence we realized this fact and came to the conclusion that directly applying a searching algorithm on the file name can cause a lot of trouble and can lead to leaking of sensitive information if two important file names are quite similar. Hence we suggest the use of keywords to save the file. When a file has to be saved, the user will firstly have to assign it a keyword and then the file will be saved on the cloud using that keyword. There will never ever be a mention of the name of the file and hence this problem of data integrity will be solved. Also the fact that once a keyword is entered, the system will calculate the possible typo keywords that can be entered by the user and will save those in the memory, the user cannot enter the same keyword again and hence no ambiguity will be there in terms of naming of the keywords as well. And hence even after doing a typo, only that file will be shown that is associated with the actual keyword and not any random file. Also file names can sometimes give away a little information about what might possibly be there inside the file. But saving the file with a keyword solves that problem as well [2].

3. ANALYSING EARLIER METHODS OF KEYWORD SEARCHING

Before showing how our suggested method works, we shall first showcase the drawbacks that exist within the earlier methods that make them both inefficient and inappropriate to use [8].

So, now that we have saved our file with a keyword that will exclusively be used for that file and will represent that file in the database, there are 3 methods, explained below in detail, of breaking keywords into individual letters and making its fuzzy sets, which are then saved into the computer memory.

3.1 Straightforward Method of Set Construction

In the first method, we take a keyword, break it into individual letters and replace every letter with the letter from A to Z and save the words so formed into the computer memory so as to cover all the probable cases of a typo error that can occur while typing in the keyword. But as cumbersome as it sounds, it will take a lot of time and memory which is unnecessary and will lead to very inefficient way of formation of a fuzzy set and saving of the fuzzy set [3].

For example if we take “KEY” as the keyword, then the algorithm will make the computer save the following values into its database as the probable cases of typo associated with this keyword:

KEY: - {AEY, BEY, CEY, ZEY, KAY, KBY, KCY, KZY, KEA, KEB .. KEZ}

Even though this approach might lead us to file saved with the keyword entered, but even for such a small keyword, there are as many as 3x26 cases being saved in the computer memory, which is sheer wastage and underutilization of resources at hand. This act might make the system inefficient and incompetent. Hence, there was a need for a more efficient method of searching and in came the concept of ‘Gram Based Error Assumption’, explained in the following section [7].

3.2 Gram Based Error Assumption

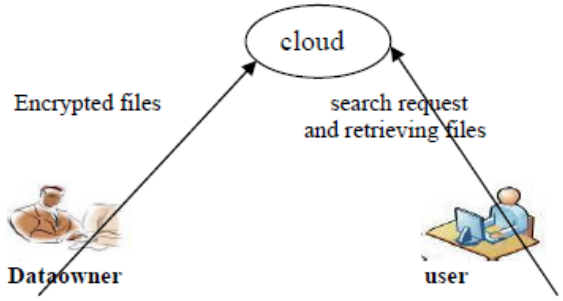
In this method, an assumption was made that the error can be of as many letter as possible and it was assumed that while typing the keyword, the typo error, can be as small as of a single letter or as big as 26 letters, and while doing the same, the person either replaced the letter being typed or added extra letters that were not required in the keyword [5]. An example of the same can be seen below:

Ex: For the keyword Magic, the following words will be saved in the fuzzy set so formed:

{*MAGIC, *AGIC, M*AGIC, MA*GIC, MAG*IC, MAGIC*C, MAGIC*, M*GIC, MA*IC, MAG*C, MAGI*}

Where * can be a typo error of as many letters as possible.

But the fault in this method lies in the fact that the assumption says that error can be of as many letters as possible whereas usually the error is of just one letter [5]. Hence, while covering all the unnecessary cases, whose probability of occurring is very less, we are wasting a lot of our resources in terms of time and memory. Hence another more efficient method was required and in came another revision of this method explained below [5].



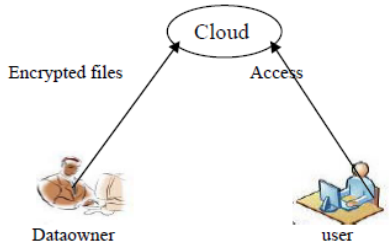
3.3 Unit Distance Error Assumption

In this method, the assumption made is that the error is at unit distance and there is at most a single letter error or a unit error and that can be identified and using that, file can be retrieved even after committing a typo error [4].

For example, considering the keyword "MAGIC", and if we type MNAGIC, the file will be retrieved as error is at unit distance.

Hence by doing this, the size of the fuzzy set thus being formed will be reduced significantly.

But the very fact that none of the algorithms till now are thinking about the fact that a typographical error occurs only around the key that one was about to type is something we realized and made use of in the algorithm suggested by us in the next section [6]. As a result of our efforts, we have made the fuzzy searching process even more efficient by using the method suggested by us in the following section.



4. ASCII BASED ALGORITHM

Whenever a typographical error occurs, it occurs out of the letters that surround the letter that we were about to type. So for example, if we were about to type in the letter 'k', the only probable typos that can occur while typing k are:

K – L M J I O

We realized this fact and made use of this logic in order to develop our algorithm [9].

Hence, if we think this way, then the size of the set that we were creating gets reduced significantly as only 5 probable cases are actually going to occur out of the otherwise 26 cases that were being stored in the earlier methods. And this reduction is just with one letter's case. As the size of the keyword will go on increasing, this efficiency will go on further increasing.

So, if we take the same example of using the keyword "KEY", the set so obtained and saved in the computer memory using ASCII based algorithm will be as follows:

KEY:
{KEY,LEY,MEY,JEY,IEY,OEY,KRY,KWY,KDY,KSY,KFY,KEU,KET,KEG,KEH,KEJ }

So, the number of cases so obtained are = 16 as compared to the number of cases obtained in normal approach which were = 78.

The lower the number of cases, the lower amount of memory used, and hence, the searching will be more efficient and faster both in terms of speed as well as in terms of memory consumption.

The way in which this algorithm will be used is the fact that every letter of the keyboard has an ASCII value associated with it and we have tried to exploit this fact to obtain the letters surrounding the letter entered in the keyword using its ASCII value. The algorithm is given in the following section below [8].

5. WORKING OF ALGORITHM

```
private void calculateASCIIDifference(String str1, String str2)
  {
   try{
     n = str1.length();
     m = str2.length();
     init = new double[n][m];
     for (int i = n - 1,i2 = 0; i >= 0; i--, i2++) {
        for (int j = 0; j < m; j++) {
           init[i2][j] = Math.abs((int) str1.charAt(i) - (int) str2.charAt(j));
        }
     }
   } catch(Exception e){
      System.out.println("calculateASCIIDifference exception: "+e.toString());
     }
  }
   private double[][] getASCIIDifference()
  {
     return init;
  }
private void setASCIIDifference(double[][] a,int n, int m) {
     this.n = n;
     this.m = m;
```

```
    this.init = a;
  }
  public double calculateSimilarity(String str1, String str2)
  {
    calculateASCIIDifference(str1,str2);
  try{
     result = new double[n][m];
     double res, val1, val2, val3;

     //initialize first element
     result[n - 1][0] = init[n - 1][0];

     //initialize first column
     for (int i = n - 2; i >= 0; i--) {
        result[i][0] = result[i + 1][0] + init[i][0];
     }
     //initialize first row(from down)
     for (int j = 1; j < m; j++) {
        result[n - 1][j] = result[n - 1][j - 1] + init[n - 1][j];
     }
     //initialize others
     for (int i = n - 2; i >= 0; i--) {
        for (int j = 1; j < m; j++) {
           val1 = result[i][j - 1] + init[i][j];
           val2 = result[i + 1][j] + init[i][j];
           val3 = result[i + 1][j - 1] + (init[i][j] * 2);

           //minimum of the 3 val's
           res = val1 < val2 ? val1 : val2;
           res = res < val3 ? res : val3;

           result[i][j] = res;
        }
     }
     return getSimilarityValue();
  }
  catch(Exception e){
   System.out.println("calculateSimilarity exception:
"+e.toString());
  }
  return -1;
  }
  //getting result of Similarity
  private double getSimilarityValue() {
   //need to devide if m != n
   if(m != n)
     return result[0][m - 1] / (m + n);
   else
   return result[0][m - 1];
  }
  //getting result array
  private double[][] getSimilarityArray()
  {
     return result;
  }
     FuzzySearch(String sval)
  {
          double result1;
           int i=0;
try   {
               connn obj= new connn();
               obj.connn1();
               decription dd= new decription();
                obj.st.executeUpdate("delete from  similardt");
               ResultSet    res=obj.st.executeQuery("select    *
from database");
             Statement    st1=obj.con.createStatement();
             while(res.next())
          {
                    String a=res.getString(2);
                    String use=res.getString(4);


System.out.print(a);
System.out.print("\t\t");
 result1 = calculateSimilarity(a,sval);
 System.out.println(result1);
    String dt= dd.decription1(a);
st1.executeUpdate("insert          into          similarity
value('"+result1+"','"+dt+"','"+a+"','"+use+"')");
}
}
 catch(Exception ee)
  {
          ee.printStackTrace();
  }
```

6. RESULTS

The ASCII based algorithm suggested by us increases the efficiency of the searching the database and result retrieval, both in terms of speed and time. As shown in the sections above, at one place, where we were saving 78 possible cases of error for a three letter keyword, the method suggested by us made it to come down to 16, which is a drastic decrement. And the very fact that this level of efficiency will only keep on increasing with every increase in the size of the keyword makes in the best possible solution for fuzzy searching. Hence, the larger the keyword, even more efficient will be the results of the ASCII based algorithm suggested by us.

7. CONCLUSION

As a result of our efforts and research on this topic, we were able to design an algorithm which will help in performing fuzzy keyword searching in one of the most efficient ways possible till date and hence, while doing so, will make our searching typo friendly. As a result, we were able to make our cloud database typo friendly, and were able to make it retrieve the results in an extremely efficient and fast pace, making the algorithm a success.

8. REFERENCES

[1] Daniyal M. Alghazzawi, Syed Hamid Hasan and Mohamed Salim Trigui, "Advanced Encryption Standard - Cryptanalysis research", 2014 International Conference on Computing for Sustainable Global Development.

[2] S. Chandragandhi and L. M. Nithya, "Optimizing fuzzy search in XML using efficient trie indexing structure", 2013 International Conference on Recent Trends in Information Technology (ICRTIT).

[3] Simran Bijral and Debajyoti Mukhopadhyay, "Efficient Fuzzy Search Engine with B -Tree Search Mechanism", 2014 International Conference on Information Technology.

[4] Seba Susan, Abhishek Jain and Aakash Sharma, "Fuzzy match index for scale-invariant feature transform (SIFT) features with application to face recognition with weak supervision" IET Image Processing, Volume: 9, Issue: 11, 11 2015.

[5] A. Behm, S. Ji, C. Li, and J. Lu, "Space-constrained gram-based assortment for economical approximate string search," in Proc. of ICDE'09.

[6] D. Song, D. Wagner, and A. Perig, "Practical techniques for searches on encrypted information," in Proc. of IEEE conference on Security and Privacy'00, 2000.

[7] He Tuo and Ma Wenping, "An Effective Fuzzy Keyword Search Scheme in Cloud Computing", 2013 5th International Conference on Intelligent Networking and Collaborative Systems.

[8] M. Armbrust and et.al, "Above the clouds: A berkeley view of cloud computing," Tech. Rep., Feb 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html.

[9] Li Xue, Ren Wuling and Jiang Guoxin,"A solution which can support privacy protection and fuzzy search quickly under cloud computing environment", Proceedings of 2nd International Conference on Information Technology and Electronic Commerce, 20-21 Dec. 2014.