The International Journal of Engineering and Science (IJES)
|| Volume || 6 || Issue || 9 || Pages || PP 50-55 || 2017 ||
ISSN (e): 2319 – 1813 ISSN (p): 2319 – 1805

Advanced Indexing Based General Gram Filter for String Similarity Search

*Kranti Sawant, Priyambiga Rajamanickam
PG Student, Department of Computer science & Engg Dr. D.Y. Patil College of Engineering and Technology Kasaba Bawada,Kolhapur,India.
Assistant Professor,Department of Computer science & Engg SSDGCT's Sanjay Ghodawat Institutes
Corresponding Author: Kranti Sawant

ABSTRACT
Many applications such as data integration, protein detection, and article copy detection share a similar core problem that is if a string is given as a query then how to efficiently find all the similar answers from a large scale string collection. Along these lines, many existing techniques receive a prefix-channel based structure to take care of this issue, and various late works intend to utilize propelled channels to enhance the general pursuit execution. The main objective of research is to develop a gram based filter-and-verification framework to achieve the near maximum filter performance. The aim is extend to gauge the similarity between two sets of string documents using advanced N-Gram technique with respect to computational speed, accuracy and precision.We judiciously choose the high-Quality grams as the prefix of query according to their estimated ability to filter candidates. We also have used techniques and algorithms like terms frequency/inverse document frequency, generation of grams, levenshtein distance.

Keywords: *Data integration, Similarity search, gram-based framework.*

Date of Submission: 29 AUG 2017 Date of Accepted: 17 SEP 2017

I. INTRODUCTION

Text comparison now appears in all areas of the discipline, from compression and pattern matching to computational biology and web searching. The basic notion of string similarity used in such comparisons is that of edit distance between pairs of strings.
Similar string searching is an important problem because it involves many applications, such as query suggestion in search engines, spell checking, similar DNA searching in large databases, and so forth. From a given collection of strings, such queries ask for strings that are similar to a given string, or those from another collection of strings.
Similarity search has attracted considerable attention from database community recently, due to its broad range of applications in data cleaning, near-duplicate detection, natural language processing and so on. For example, data records that represent the identical real world entities may have minor differences in their representations when they are merged from different data sources. In this case the similar records need to be detected and correlated.
We have presented efficient solutions for multiple string matching using n-grams.Our algorithms work in three phases: preprocessing, filtering and verifying.

- Motivation

Nowadays, Google search engines are popular to find the respective answers. So, here in order to achieve the maximum filter performance (i.e. find all the similar answers from a large scale string collection) gram based filter-and-verification framework is used to achieve similarity search. The search engine used is G-Filter search engine.

- Problem Statement

String similarity search is a fundamental operation in many areas, such as data cleaning, information retrieval, and bio informatics. There are many system are available for string similarity search. We Underline some drawbacks in these systems like performance, accuracy and issues with the data length and proposed a gram based solution on these problems.
The aim of the project is extend to gauge the similarity between two sets of string documents using advanced N-Gram technique with respect to computational speed, accuracy and precision.

- Objectives

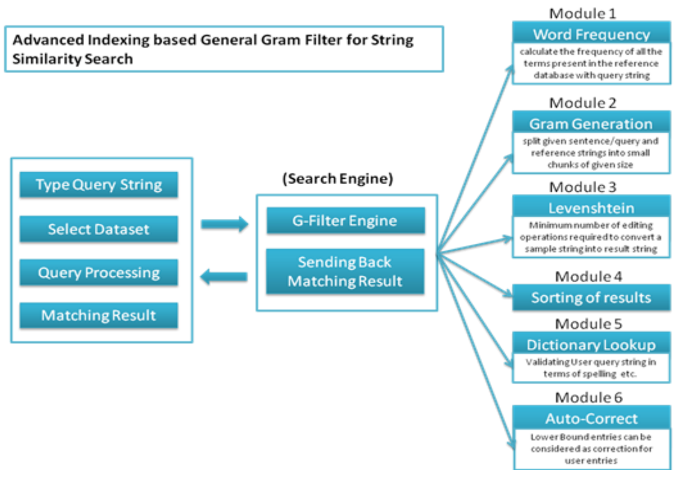
The objectives of the proposed work are as follows :

1) To calculate the frequency of all the terms present in the reference database and query string.
2) To generate the gram by splitting given sentence and reference strings into small chunks of given size.
3) To find the minimum number of editing operations required to convert a sample string into sorted result string.
4) Validating user query string in terms of spelling.

II. LITERATURE REVIEW

- In gram filter for string similarity search Haoji Hu, Kai Zheng, Xiaoling Wang, Aoying Zhou proposed a gram-based framework [1] to achieve near maximum filter performance. The main idea is to judiciously choose the high-Quality grams as the prefix of query according to their estimated ability to filter candidates. As this selection process is proved to be NP-hard problem, we give a cost model to measure the filter ability of grams and develop efficient heuristic algorithms to an high-quality grams.
- Pass-Join partition a string into a set of segments and creates inverted indices for the segments. Then for each string, Pass-Join [2] selects some of its substrings and uses the selected substrings to find candidate pairs using the inverted indices. J. Wang , G. Li and J. Feng proposed a technique which requires a heal lot of time for result retrieval and the main problem with this technique is size of the segments. The code complexity is very high with this approach.
- In Top-k string similarity search D. Deng, G. Li ,J. Feng and W. Li used Edit-Distance Constraints [3] which includes a given collection of strings and a query string, returns the top-k strings with the smallest edit distances to the query string. Existing methods usually try different edit-distance thresholds and select an appropriate threshold to find top-k answers. However it is rather expensive to select an appropriate threshold.
- In approximate string matching the problem is to find a text where a text given pattern occurs allowing a limited number of "errors" in the matches. G.Navarro proposed a possible operations of insertion,deletion,substitution.[4.]This was modeled as searching for given "patterns" in a "text".
- In SSJ [5] the main problem is a document simply a set of words in the document, in the VSJ problem a document can be modeled with a vector of words with TF-IDF weights. H. Lee, R.T. Ng, and K. Shim proposed that it can also deal with multiset semantics with occurrences. In fact, most of the studies on similarity joins formulate the problem with sets and then extend it with TF-IDF weights, which is indeed a vector similarity join.
- In Efficient exact set-similarity joins , SSJoin [6] identifies all pairs of sets, one from each collection, that have high similarity. SSJoin is as a powerful primitive for supporting (string)similarity joins. A variety of indexing and join techniques can help to speed up the generation of candidate pairs. Using SSJoins for string similarity joins is the observation that if two strings have small edit distance, then their n-gram sets are similar.
- In similarity estimation techniques from rounding algorithms it represent sets by their characteristic vectors and use this locality sensitive hashing scheme for measuring similarity between sets which is proposed by M.S. Charikar and explore constructions of locality sensitive hash functions[7] for various other interesting similarity functions.

III. PROPOSED WORK


Advanced Indexing based General Gram Filter for String Similarity Search
Type Query String
Select Dataset
Query Processing
Matching Result
(Search Engine)
G-Filter Engine
Sending Back Matching Result
Module 1
Word Frequency
calculate the frequency of all the terms present in the reference database with query string
Module 2
Gram Generation
split given sentence/query and reference strings into small chunks of given size
Module 3
Levenshtein
Minimum number of editing operations required to convert a sample string into result string
Module 4
Sorting of results
Module 5
Dictionary Lookup
Validating User query string in terms of spelling etc.
Module 6
Auto-Correct
Lower Bound entries can be considered as correction for user entries

The proposed system will be implemented in the following modules :-
1] Word Frequency/Inverse Document Frequency :
This module is used to calculate the frequency of all the terms present in the reference database with query string.
2] Gram Generation :
Gram generation module is used to split the given sentence/query and reference strings into small chunks of given size
3] Levenshtein Distance :
Levenshtein distance module is used to find the minimum number of editing operations required to convert a sample string into result string.The distance is the number of deletions,insertions, or substitutions required to transform the given string.
4] Sorting of results :
In this module, depending upon results of previous stages, final results are calculated by sorting the results in an ascending manner and presented.
5] Dictionary Lookup :
User entries must be validated for desired output or they must be suggested corrections in case of spelling mistakes/unmatched queries.
6] Auto-Correct Suggestions :
In this module the other index terms that have low Levenshtein distance which do not exactly match with query string can be proposed as possible corrections to user entries.

IV. SCOPE

We propose a gram-based system to accomplish close greatest channel execution. The principle thought is to wisely pick the amazing grams as the prefix of question as per their assessed capacity to channel competitors. As this choice procedure is turned out to be NP-difficult issue, we give a cost model to gauge the channel capacity of grams and create productive heuristic calculations to a great grams.

Methodology :
The accompanying are the strategies and calculations in proposed procedure :
1] Module 1 : Terms Frequency/Inverse Document Frequency :
This strategy would be utilized to ascertain the recurrence of the considerable number of terms exhibit in the reference database. It is helpful to discover relevant matches to the inquiry string and disdain more refined/exact outcomes. Commonly, the tf-idf weight is made by two terms: the primary figures the standardized Term Frequency (TF). The quantity of times a word shows up in a report, separated by the aggregate number of words in that record; the second term is the Inverse Document Frequency (IDF), registered as the logarithm of the quantity of the archives in the corpus partitioned by the quantity of archives where the particular term shows up.
Term Frequency : It gauges how much of the time a term happens in a record. Since each record is distinctive long, it is conceivable that a term would seem considerably more circumstances in long reports than shorter ones.
TF(t) = (Number of times term t shows up in a record)/(Total number of terms in the archive).
Opposite Document Frequency : It quantifies how vital a term is. While registering TF, all terms are considered similarly essential. In any case it is realized that specific terms, for example, "is", "of", and "that", may show up a ton of times yet have little significance.

IDF(t) = log_e(Total number of records/Number of archives with term t in it).

Case : Consider a record containing 100 words wherein the word feline shows up 3 times. The term recurrence (i.e., tf) for feline is at that point (3/100) = 0.03. Presently, accept we have 10 million records and the word feline shows up in one thousand of these. At that point, the reverse record recurrence (i.e., idf) is ascertained as log(10,000,000/1,000) = 4. Along these lines, the tf-idf weight is the result of these amounts: 0.03 * 4 = 0.12.

2] Module 2 : Generation of Grams :
This technique would be utilized to part given sentence/question and reference strings into little lumps of given size. These lumps are additionally used to ascertain Levenshtein remove between strings to discover similitude.
n-gram calculation :

A n-gram is a sub-arrangement of n things from a given succession. The n-grams are utilized as a part of different zones of factual normal dialect handling and hereditary succession examination. The things being referred to can be characters, words or base sets as per the application. For instance, the succession of characters "Hatem mostafa helmy" has a 3-gram of ("Hat", "ate", "tem", "em ", "m", ...), and has a 2-gram of ("Ha", "at", "te", "em", "m ", " m", ...). This n-gram yield can be utilized for an assortment of R&D subjects, for example, Statistical machine interpretation and Spell checking. A n-gram of size 1 is alluded to as a "unigram"; estimate 2 is a "bigram" (or, less usually, a "digram"); measure 3 is a "trigram". Bigger sizes are at times alluded to by the estimation of n, e.g., "four-gram", "five-gram", et cetera.

```
ArrayList<String> nGrams = new ArrayList<String>();

int counter = 0, i = 0,int chunkSize=3;

String str="This is my auto";

char[] gram = new char[chunkSize];

while (i < str.length() + chunkSize)

{

in the event that (counter == chunkSize)

{

counter = 0;

nGrams.add(new String(gram));

gram = new char[chunkSize];

i = i - (chunkSize - 1);

}

gram[counter] = i >= str.length() ? '' : str.charAt(i);

counter++;

i++;

}
```

Pseudo Code :

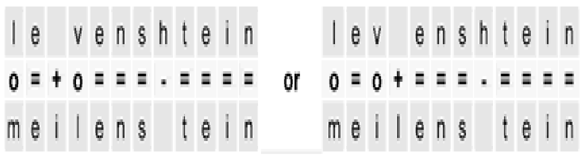
```
1. Allocate the buffer and assign it to des variable.
2. Copy the source information to the destination.
3. If any delimiters is in des discard it.
4. Initialize pattern dictionary and assign to the variable dic.
5. Initialize new pattern(des) and assign to the variable pattern.
6. Continue further steps until des pattern is found.
6.1. Insert the pattern into the dictionary and assign that to the node variable.
6.2. If it is not fixed. Add that word to the pattern else initialize new pattern des and assign it to the variable pattern.
6.3. Finally update the buffer.
```

In order to improve the performance of matching the query and the dataset
KShingLing algorithm is used.

KShingLing algorithm :
K-shingling is the operation of transforming a string (or text document) into a set of n-grams, which can be used to measure the similarity between two strings or documents. Multiple subsequent spaces are replaced by a single space.The default value of k = 9.

3] Module 3 : Calculation of Levenshtein Distance (LD):
Levenshtein remove is the base number of altering operations required to change over a specimen string into result string. Contrast in the lumps is figured utilizing this and included for conclusive outcomes.

The most well-known method for ascertaining this is by the dynamic programming approach. The lattice can be filled from the upper left to the lower right corner. Each bounce evenly or vertically compares to an embed or an erase, separately. The cost is typically set to 1 for each of the operations. The slanting hop can cost it is possible that one, if the two characters in the line and section don't match or 0, on the off chance that they do. Every phone dependably limits the cost locally. Along these lines the number in the lower right corner is the Levenshtein separate between the two words.



"=" Match; "o" Substitution; "+" Insertion; "-" Deletion
Levenshtein separate is a measure of the closeness between two strings, which we will allude to as the source string (s) and the objective string (t).The distance is the number of deletions, insertions, or substitutions required to transform s into t. For example,
1. If s is "test" and t is "test", then LD(s,t) = 0, because no transformations are needed. The strings are already identical.
2. If s is "test" and t is "tent", then LD(s,t) = 1, because one substitution (change "s" to "n") is sufficient to transform s into t.
Pseudo Code :

```
Compare n(First character of a string s2) with length of s2. If n=length of s2 then subtract length of string s1 with m(First character of string s1).
2. Compare m with length of string 1.If m=length of string 1 then subtract length of string s2 with n.
3. If string 1 and string 2 is equal then recursively call the method by passing the parameters of string s1 and s2, m+1, n+1.
4. If it is not equal to then
     1+min(min(count(s1,s2,m,n+1),count(s1,s2,m+1,n)),count(s1,s2,m+1,n+1)).
```

4] Module 4 : Analysis module:
a) Dataset table 1 :
1] Name : Uniprot dataset
Size : 878148
Number of records : It is based upon the average length of dataset.
Description : It is used for searching sequentially the records. It is related to protein Sequence data.
2] Name : TREC dataset
Size : 417698
Number of records : It is based upon the average length of dataset.
Description : It is used for Question/Answering purpose.
3] Name : DBLP dataset
Size : 650207
Number of records : It is based upon the average length of dataset.

DataSets	avg_len	max_len	min_len	sizes	Description
Uniprot	406	4017	120	878,148	Protein Sequence Data
TREC	240	779	100	417,698	question answering track
DBLP	105	1626	28	650,207	bibliography dataset of author names and article titles
DBLP-author	39	100	16	650,207	bibliography dataset of author names and article titles

Description : It is used to find the bibliography dataset of author names and article titles.
4] Name : DBLP-author dataset
Size : 650207
Number of records : It is based upon the average length of dataset.
Description : It is used for retrieval concept to find the bibliography dataset of author names and article titles. The comparison is based on R & D research. Basically dataset used is DBLP.

b) Dataset table 2 :

DataBase Name	Best Case Result %	Average Case %	Worst Case %
DBLP	97	91	86
Vocabulary	91	89	71
Uniprot	86	83	33
Trec	92	71	67

Finally, we will evaluate the performance of advance indexing with that of existing system using the following parameters :
1) Matching time and accuracy analysis (best case, average case, worst case) will be experimented based on two datasets and then performance is calculated on database.
2) Matching time will be calculated by executing the query in existing system as well as in the proposed system.

V.CONCLUSION

In existing approach prefix filter based framework is used.As prefix length have significant effect on performance and does not also achieve high performance so to resolve inadequacies in the existing framework we propose a gram based filter-and-verification framework to achieve maximum filter performance in search engine's.The search engine we used is G-Filter search engine.We devise an effective criterion to sort grams based on their potential capability to reduce candidate size, which is used to select better grams efficiently and effectively.A new selection algorithm is developed to construct the high quality query-gram set.The selection process is proved to be NP-hard problem, we give a cost model to measure the filter ability of grams and develop efficient heuristic algorithms to an high-quality grams. We conduct extensive experiments based on multiple real datasets.

REFERENCES

[1] Haoji Hu, Kai Zheng, Xiaoling Wang, Aoying Zhou, "GFilter: A General Gram Filter for String Similarity Search", in IEEE Transactions on Knowledge and Data Engineering, vol.27, no. 4 , April 2015.
[2] J. Wang, G. Li, and J. Feng, "Can we beat the prefix filtering? An adaptive framework for similarity join and search", SIGMOD '12 Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, May 20 - 24, 2012.
[3] D. Deng, G. Li, J.Feng, and W.Li, "Top-k string similarity search with edit-distance constraints", DATA Engineering (ICDE) Conference, April 8 - 12, 2013.
[4] G. Navarro, "A guided tour to approximate string matching", ACM Computing. Surveys, vol.33, no. 1, pp. 31–88, 2001.
[5] H. Lee, R. T. Ng, and K. Shim, "Similarity join size estimation using locality sensitive hashing", in Proceedings of the VLDB Endowment,vol.4, no. 6, March 2011, pp. 338–349.
[6] A. Arasu, V. Ganti, and R. Kaushik, "Efficient exact set-similarity joins", in VLDB '06 Proceedings of the 32nd international conference on Very large data bases, September 12 – 15, 2006, pp. 918–929.
[7] M. S. Charikar, "Similarity estimation techniques from rounding algorithms", in STOC '02 Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, May 19 - 21, 2002, pp. 380–388.