Performance Optimization for Top-K String Similarity Search Using Inverted Index

Ibrahim Musa Conteh[1], Gibril Njai[2]

[1,2]*Department of Information and Communication Technology, Faculty of Engineering and Technology, Ernest Bai Koroma University of Science and Technology, Magburaka, Sierra Leone*

---***---

Abstract – *With the rapid growth in internet technologies, the optimization of web search and information retrieval in a large dataset repository is of a serious challenge. The filter-and-verify framework for non-candidates pruning with some lower bounds of edit distance has been widely used. However, this framework is generally inefficient due to their breadth first search algorithm that results in repeated computations and it answers are based on approximations. To solve this problem, we proposed a weight-less top-k fast search algorithm that largely avoids repeated computations and can achieve better results regardless of the scales of similarity search. The efficiency and effectiveness of the proposed algorithm are empirically demonstrated with Java and Microsoft excel using real-world datasets as simulated data. Moreover, we compare our algorithm with the filtering methods to verify its efficiency and early termination of a query when the results were found using the same device specifications. Experimental results show that our approach performs 2 times faster than traditional methods with more than 97% of the results returned.*

Key Words: - Appgram, Apriori, Flamingo, HS-top-K, WL-search, string similarity search; top-k.

1. INTRODUCTION

As the demand for data increases, particularly in this era of big data, the optimization of web search and information retrieval in a large dataset [1] repository becomes serious research challenge [2]. In the real-world, string similarity search have many applications such as spell checking, web search [3], data cleaning, DNA sequence search, and plagiarism detection. Given a set of strings and a query, string similarity search aims to find all strings from the string set that are similar to the query. The more fact you have from the data the more self-assured you will be in taking a decision. We used data mining techniques to know the hidden information by analyzing the huge and complex dataset at hand. There are many metrics to quantify between strings like Jaccard, Dice, cosine and edit distance [3][4]. To tolerate typographical error, the edit distance is the most widely used metric [5] In this paper we consider the later to quantify string similarity.

In string similarity searching studies, authors in [6][7] techniques mostly emphasize efficient filter performance with an effective and efficient indexing structure. The authors in [8][9] studies threshold-based similarity search problem using filter-verification model. Such method requires more computational time and hence very expensive to perform.

Chen Li et al in [10] studied ScanCount, DivideSkip, and MergeSkip algorithms for string similarity. Their focus was on how to identify similar strings efficiently in a set of strings given a query string. Again, recent studies on top-k similarity search in [11][12] are inefficient for threshold-based search because they cannot make full use of the given threshold to do pruning. These existing methods are efficient either for threshold- based search or for top-k search and most of them can't efficiently support both of the two problems. Thus, the need for an optimized framework to efficiently support the two variants of string similarity search with inverted index.

To address this problem, we propose an optimization scheme for top-K similarity search with inverted index. We defined n-grams as possible finite number of strings. The more candidates are fetched, the more accurate our algorithm will be. We also consider the trade-off between the query performance and the accuracy of the answers. The experimental study shows that our algorithm can achieve the best performance compared with other algorithms with very high accuracy.

The rest of the paper is organized as follows. In section II, we presented our algorithms and how its implemented. Preliminaries in section III. Framework description in section IV. Experimental results and analysis in section V. We conclude in section VI.

2. ALGORITHM AND IMPLEMENTATION

In this paper, like most algorithms dealing with the string similarity search problem, a filter-and-verify framework was also used. We adopt the index structure and propose a light-weight top-k search algorithm. It was implemented in Windows 10 operating system with 64 bits computer. The processor used: Intel(R) Core (TM) i5 – 3337U @ 1.80GHz, 1.80GHz with 4 GB RAM. We evaluated this algorithm by varying K-value and varying the datasets as well. We compared our algorithm performance "Weight-Less top-k fast search", with different algorithms like Flamingo, Apriori, HS-Top-K, and Appgram.

Algorithm 1: *Weight-Less Top-K Fast Search String Similarity Algorithm*

```
 start
 Declare input Variable: Q = Query; Tds = Training Dataset; K = Number of result
 Declare output Variable: R =Top-K Result
 Get Tds from source
   Decompose strings to N-Gram
     Building of inverted index for Tds
     Store the index (Key & values)
     Repeat step 5 for each incoming Tds
         Get Q from user
           Decompose Q; by Repeat step 3
             Change Q to N-Gram
                 Use Gn(q) to get N-Gram set
 Using the key for generating the inverted list
 Stop
 Assigning K Sequence
 Assigning max-heap H and a threshold (t) using first visited K sequence
 Generate an inverted list
 Combined the retrieved list
 Sort by F for both exact & approximate N-Gram list
      Count F for each string Tds
          For each unprocessed string s in Tds
          Select candidate if F >= Lower Bound (t)
            If F[s]>= t then
           Calculate ED (q, s)
         If ED (q, s) < (t) then
      Update H and t
   Stop
  Rank result by similarity to Q
 Output Top-K result
      Stop
```

3. METHODOLOGIES

We integrated an index structure that has a one-level inverted index, and design a faster search algorithm call Weight-Less top-k fast search algorithm. Here, these technologies were used: N-Gram technology, indexing technology, Filtering, and Verification technology and Edit distance

3.1 The N-Grams and Inverted Index

N-grams scenario in this paper, if you are given S as a query string and n is a positive integer, we obtained a set of substrings by sliding window of size n over string S starting from the first component in the string to the latter component in the string. which is the n-gram set of string S, and it is represented by;

Gn (s) For string "Conteh",

its 3-gram set is G3 (Conteh) = {" con", "ont", "nte", "teh"}.

The most essential aspect of using this n-gram based inverted index method was for the use of the inverted lists to find all candidate strings that have a certain number of same n-grams with the given query string. We constructed our inverted index due to n-gram, and assign a key to each of the n-grams and have the string that comprises the n-gram into the equivalent inverted list. Algorithm 2 is responsible to decompose the raw data and the issued query from the user into grams. These decomposed 5-grams will be stored in the index for easy retrieval of the inverted list. If the length of the input has fewer or same gram length (less or equal 5 gram) in this case, there was no need for the decomposition of the query and we will just return the input as a gram. When the user is trying to retrieve an answer without issuing a query, this will throw an error "the input is empty" to notify the user must enter a query before searching.

Algorithm 2: *N-Gram processing operations*

```
Line 1: public static List<String> getNGram(String input) throws Exception {
if (input == null
throw new Exception ("The input is empty");
List<String> indexes = new ArrayList<>();
    if (input.length() <= GRAM)
    indexes.add(input);
    return indexes;
for (int i = 0; i < input.length() - GRAM + 1; i++)
    StringBuilder buffer = new StringBuilder ();
    for (int j = i; j < GRAM + i; j++)
    buffer.append(input.charAt(j));
    indexes.add(buffer.toString());
    return indexes;
```

3.2 Index and Compression Technique

We used one level inverted index and introduced some ideas of Wang et al. in [13] that used a two-level inverted index. For the recovery of the top-k string, we store with the string id's, key, and each word's frequency in the document will be calculated and store in a sorted form which could aid in improving searching performance. We constructed this Inverted Index through the n-gram set of our data, preparing every n-gram as the key and place the string that contains the specific n-gram into a matching inverted list, when the user issue the query, likewise, this query will also decompose using the same principle. This was used to recover the matching inverted list from the index according to exact n-grams.

Different from the ordinary inverted index, after getting the inverted list from the index, you could consider either exact n-grams or using the approximate n-grams for result retrieval. Therefore, to know which string comprises the certain n-gram (substring), we need just to acquire the inverted list by its key. Example, given four strings "approximant", "approximate", "Approximation", and "Approximately" we can update the inverted index by adding their information into the inverted index and get a simple one when we want to add information of other strings. We just have to look for the corresponding n-gram and update this information to the inverted list.

Compression technique helps to improve the searching performance through an algorithm that codes information. We used compression also for keeping more information in cache memory. Their practicality determines their power to exploit newly or regularly used data.

3.3 Edit Distance

Edit distance was used as the similarity function. The gram length used was 5 for decomposing datasets. This is a technology used to measure the dissimilarities between strings S1 and S2 denoted by ED (S1, S2), using Substitution, Insertion, and Delete (SID) operations. This is applicable in natural language, detection [8][14] to quantify the similarity and spelling correction in both data and the query, by selecting the word with less edit distance. This technique was implemented for the selection of the most appropriate candidate from the retrieved candidate strings, which are more similar to the issued query for result processing. In this process, the retrieved candidate that satisfied the threshold condition will be considered in the selection of the top-k result. Algorithm 3 show how we implemented edit distance operation in this paper. It is responsible to compute the similarities between the input and the existing grams.

Algorithm 3: Edit Distance processing operations

```
Line 1   private int similarity(String query, String word)
  return GRAM - distance(query, word);
private float similarity(String word)
  int distance = distance(query, word);
  return (2f * distance) / (query.length() + word.length());
public Object2FloatMap<Pair> call() throws Exception
  return getCandidates();
```

3.4 Query Result Ranking

The ranking was done for both exact match and non-exact match (similarities ranking) with edit distance techniques between the datasets and the issued query. The candidates were ranked by their exactness or similarity and only top-k results were returned. This algorithm will continue to search the data until it fetches all the top-k result [15], it is terminated immediately and present the fetched result.

There are certain word operations to perform between the datasets and the query. There are other sets of queries like the Boolean query, a function that assigns a 0 number to represent non-matching and number 1 represents matching, but this is not the case in this study. The simple approach is to total the term frequency of the selected candidates, the one with the highest frequency will be ranked higher for easy selection of the top-k. Algorithm 4 is responsible for ranking the selected candidate base on similarities, using their ids. We created a map that responsible for holding the ids and their similarities. Moreover, the k-value there is the top-k outputted result and it was returned in a ranked manner. Were the candidate with higher similarities will be at the top of the ranking.

Algorithm 4: Ranking Process

```
public static List<Pair>
   sortBySimilarity(Object2FloatMap<Pair> map,
 int kValue)
  {
   int size = map.size();
   return map.object2FloatEntrySet().stream()
   sorted(comparing(Map.Entry::getValue))
.map(Map.Entry::getKey).collect(toList()).subList(0,(size
> kValue ? kValue : size));
```

4. FRAMEWORK DESCRIPTION

We divide the filter-and-verify framework into two processes according to their different functions: filtering process and verification process. After building the inverted index for dataset S. Given a query string q, we can decompose it and obtain an n-gram set in which n, is the same as the setting for the inverted index. Thus, according to the n-gram set, we can retrieve several inverted lists and many filter techniques [16] can be applied to prepare the candidate set; in the verification process, all the candidates will be verified and the n most similar strings will be returned as the top-k results.

In the Gram-based method, we considered the following three key steps:

1. The Indexing step: We assemble an inverted index from the string set, which takes all the grams to be the keys and the strings that comprise the gram as the values
2. Candidate generation step: We inspect the inverted index through the grams in the issued query string and the interim counts the accuracy of the strings in the chosen lists. We could select a string to become a candidate if its appearances are above or equal to the lower bound. Length filter is one of the filter techniques used in this step for reducing the number of candidates.
3. Verification step: we calculate the exact distance between the candidate and the issued query, we added a candidate to the final result only if the distance is less than or equal to the given threshold (t) denoted as ED (s, q) <= t

5. EXPERIMENTAL STUDY

In this section, we conduct an extensive set of experiments to evaluate the efficiency of our algorithms and compare with other methods.

5.1 Experiment Setup

We used three real datasets in our experiments: eBay, Author, IMDB, which are widely used in previous studies [11]. Author contains short strings, Query Log contains medium-length strings, and DBLP contains long strings. Our first experiment was done using different N values, while the second experiment was to compare our algorithm with others like Flamingo, HS-TopK, Appgram and Appriori. During the experimental process, we selected the average query time and the average maximum edit distance for evaluation measures, and execute all different algorithms at these K-values: 5, 10, 20, 30, 40, 50, and 60 on all our three datasets used in this study. We presented the excremental results of our proposed methods. All the algorithms were implemented in Java and MS excel on a Windows 10 operating system with 64 bits . The processor used: Intel(R) Core (TM) i5 – 3337U @ 1.80GHz, 1.80GHz with 4 GB RAM. We implemented our algorithm in the following datasets as shown in Table I.

Table – 1: Summary Dataset

Dataset	Records	Size	Format
eBay	258,589	72.9 MB	Csv
Author	194,788	47.8 MB	Csv
IMDB	153,428	32 MB	csv

5.2 Performance Evaluation

1. ***Evaluation at different N-Value:*** We evaluate the performance of N-Value on the three datasets. The experimental analysis in Chart 1-3 is shows a clear performance of different algorithms in different candidate size (N value) on all three used datasets. In this, we differ the size of N as 1000, 2000, 4000, 8000, 16000, 32000 and 64000 while setting the K value as 5, 10, 20, 30, 40, 50 and 60 with query time in milliseconds (ms). Thus N-Value will terminate within a less ms.

2. ***Comparison on Different Datasets****:* We compared our WL-Search algorithm with other algorithms as shown in Chart 4-6 by varying different edit distance metric on the three datasets. Chart 4-6 shows an explicit performance comparison between five algorithms and our WL-search algorithm including four others, with an N value of 16000 for all three different datasets. Moreover, varying the K value as 5, 10, 20, 30, 40, 50, and 60. Results shows our algorithms returns search result in less computational time.

3. ***Verification and Filtration Time:*** In Chart 7-9, we show results of the comparison of records of WL-Search and Appgram algorithms in their verification and filtration time at different K-Value. This is to demonstrate the time each of these two best algorithms took the verification and filtration in a bar chat representation. The analysis was done in all the three different datasets and each diagram contains four different colored bars, except the eBay dataset that has three different bars, which means no filtration time for Appgram. The blue and brown represent our WL-Search.

*4. **The Top- K Results***: The top-60 result was retrieved in the eBay dataset within 401 milliseconds (ms). To obtain this time, we issued many queries with a threshold of 3. We also increased the size of the records in each dataset. Chart 7 is showing the query time of the Weight-Less top-k fast search algorithm took to execute the top-60 results. The query “legend” was issued in the eBay dataset for the top 60.

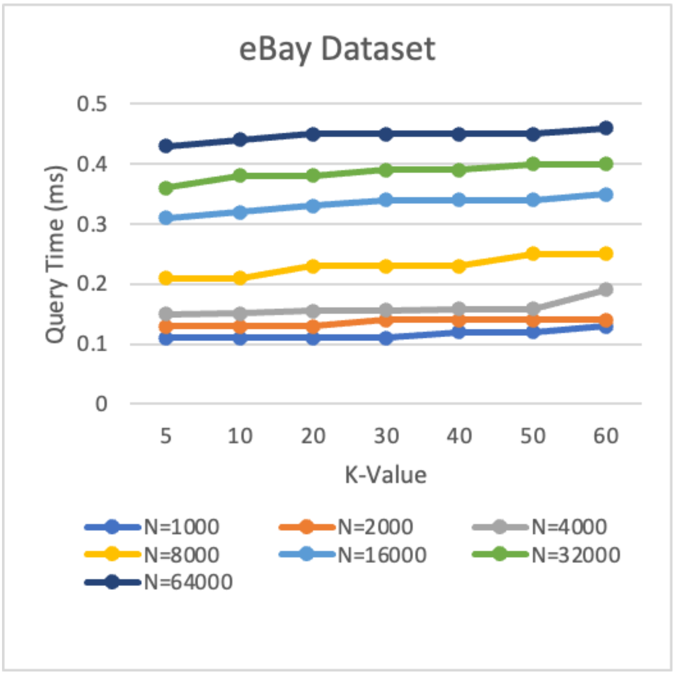

Chart -1: Evaluation on eBay dataset – K-Value N-Value

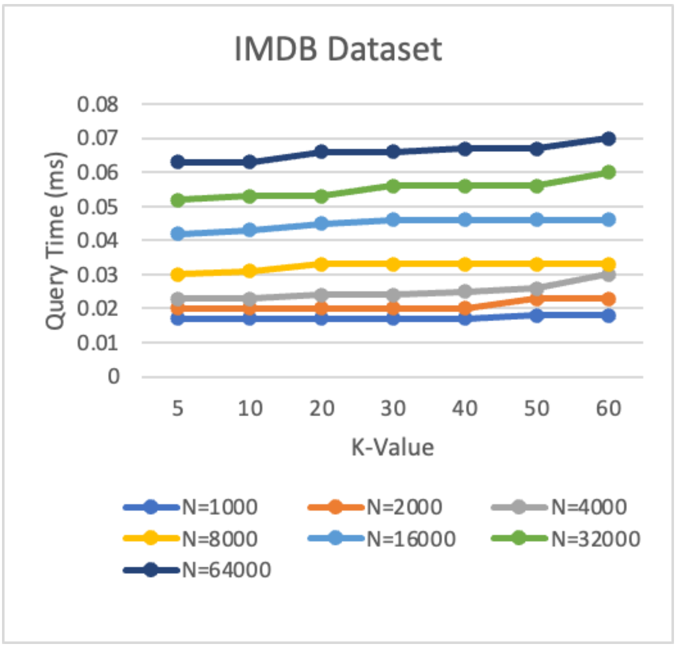

Chart - 2: Evaluation on eBay dataset – K-Value N-Value

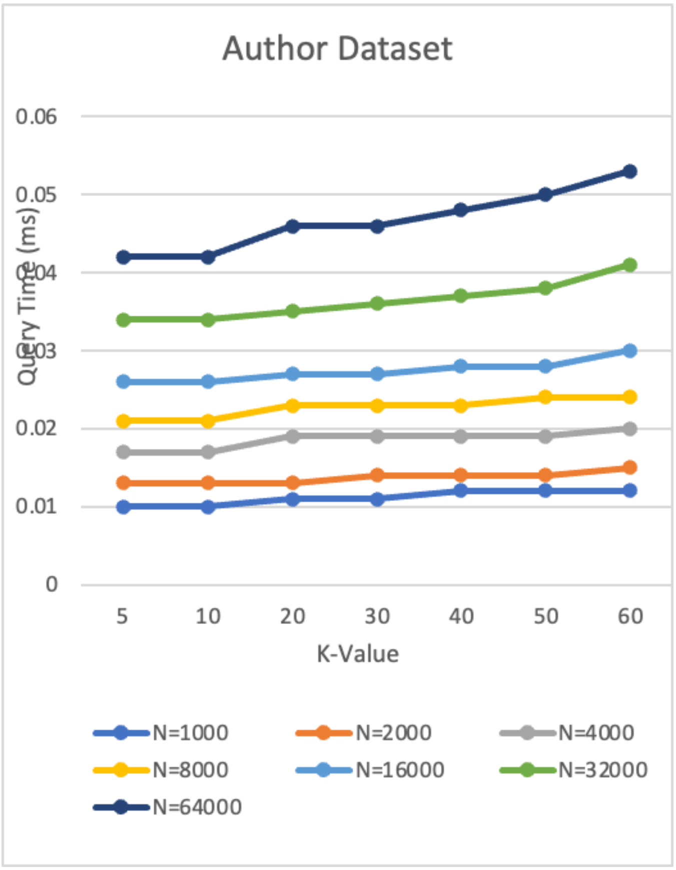

Chart - 3: Evaluation on Author Bay dataset – K-Value N-V

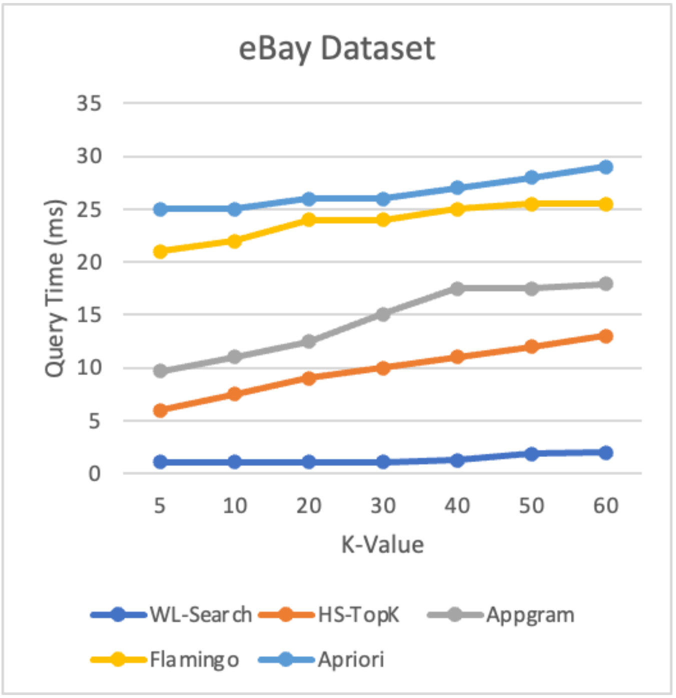

Chart - 4: Algorithms Comparison on eBay Datasets

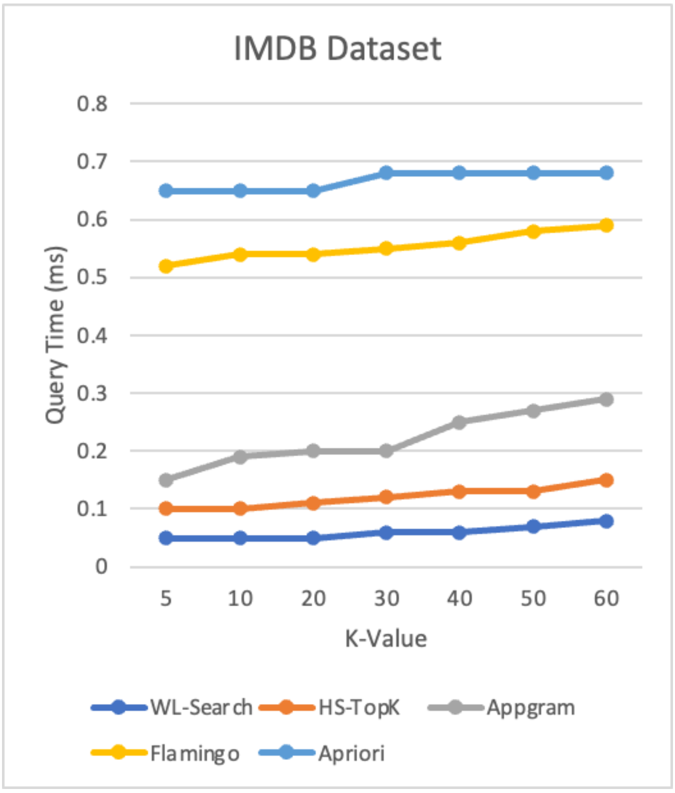

Chart -5: Algorithms Comparison on eBay Datasets

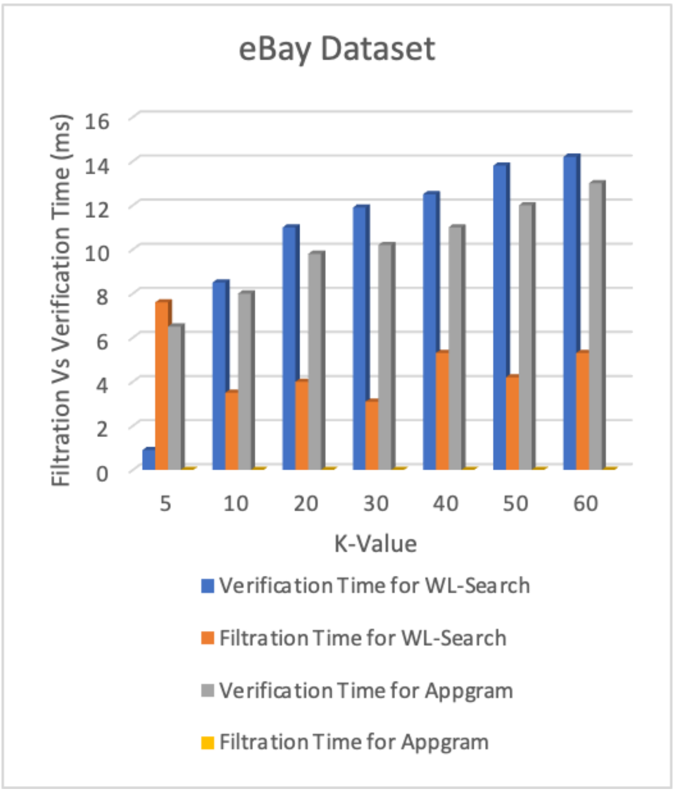

Chart -7: Verification and Filtration Time on eBay dataset

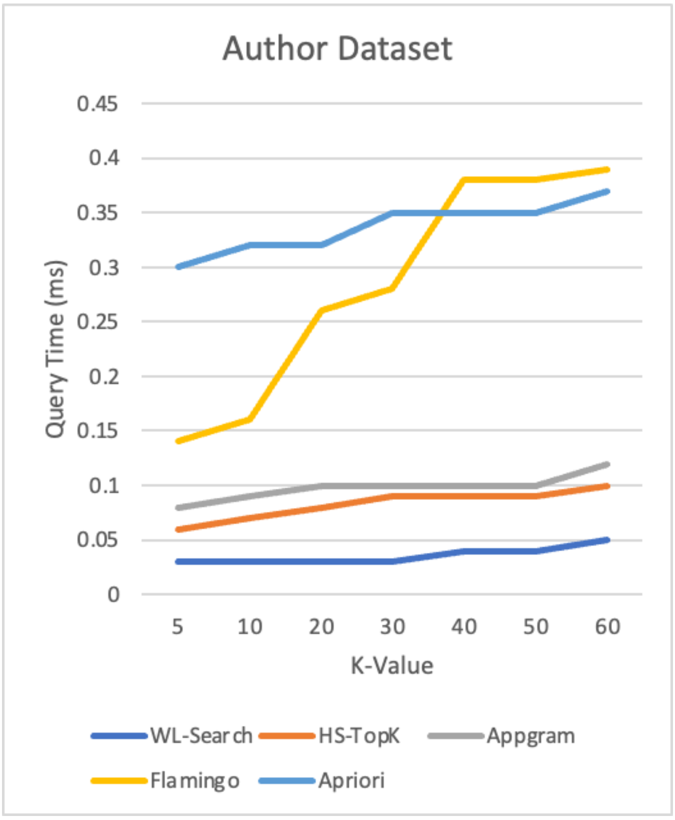

Chart - 6: Algorithms Comparison on eBay Datasets

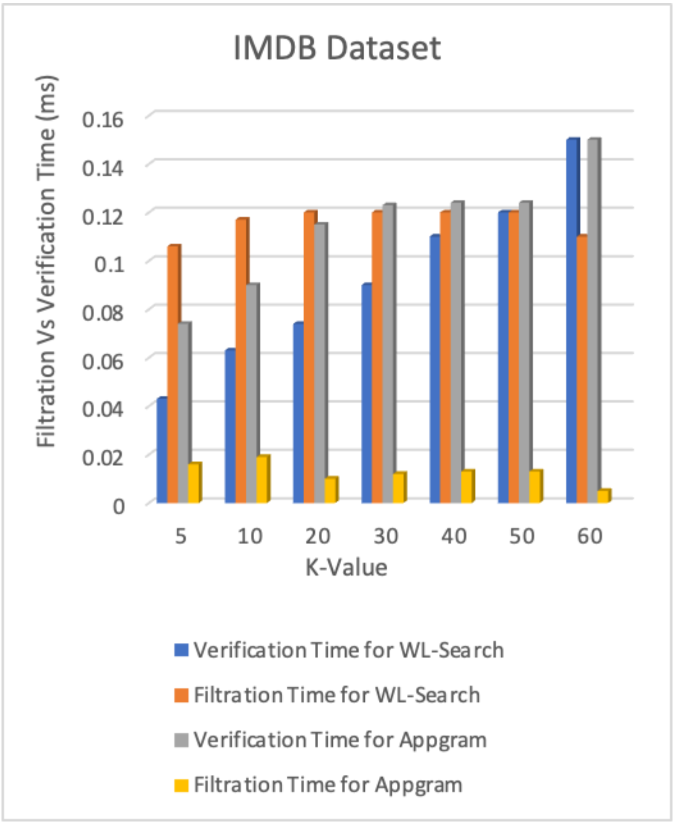

Chart - 8. Verification and Filtration Time on IMDB dataset

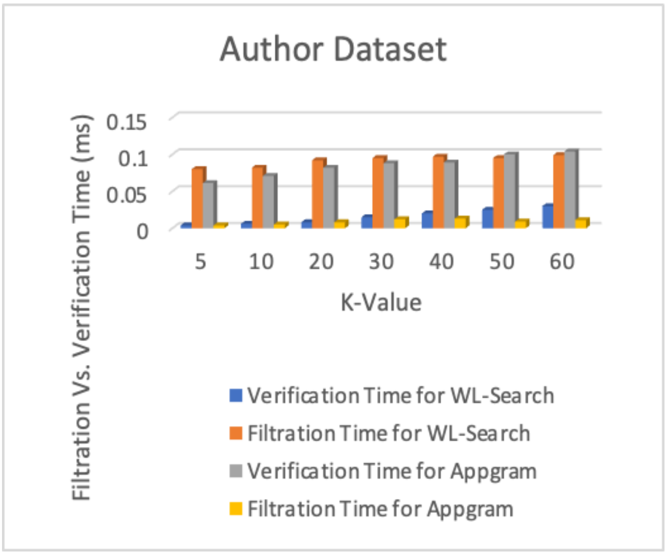

Chart – 9: Verification and Filtration Time on Author dataset

TABLE:2 ACCURACY OF WL-SEARCH ALGORITHM

Dataset	Top-5	Top-10	Top-20	Top-30	Top-40	Top-50	Top-60
eBay	99.5%	98.9%	97.2%	97.7%	97.8%	98.2%	98.9
IMDB	99.4%	99%	96.7%	96.7%	95.3%	96.8%	97.5%
Author	99.6%	99.7%	98.3%	98.4%	98.2%	98.9%	97.8%

6. CONCLUSIONS

In this paper, we studied the problem of string similarity search. We find the top-k string similarity search problem using edit distance. Although existing approaches usually try to reduce the number of candidates by building index structure and using a powerful filter with a high time expense to optimize the query performance, we tackle this problem by proposing a new weight-less approach which mainly decreases the number of candidates. By choosing a proper N and verifying the string which has a high frequency, we got good performance both on efficiency and accuracy of our algorithm. The results of our algorithm are ranked according to a defined similarities function. We then integrate current filtering methods with the algorithms, and finally design our algorithm for early termination when the results were found. The algorithm has proven to have higher accuracy and best performance in string searching compare to other methods.

7. REFERENCES

[1] A. Behm, C. Li, and M. J. Carey, "Answering approximate string queries on large data sets using external memory," in *ICDE*, 2011, pp. 888–899.

[2] Duan, W. Zhai, and C. Cheng, "A Spatial Grid Index Based on Inverted Index and Its Query Method 1," no. 2, pp. 6189–61922017.

[3] Alberga C N. String similarity and misspellings J. Communications of the ACM, 1967, 10(5): 302-313.

[4] K. Balhaf, M. A. Alsmirat, M. Al-ayyoub, Y. Jararweh, and M. A. Shehab, "Accelerating Levenshtein and Damerau Edit Distance Algorithms Using GPU with Unified Memory," pp. 1–5, 2017

[5] S. U. G. Darsan, "Event extraction & Image retrieval for Story Board Generation using GLCM," 2017.

[6] J. Lu, C. Lin, W. Wang, C. Li, H. Wang String similarity measures and joins with synonyms Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, ACM (2013), pp. 373-384

[7] W.H. Gomaa, A.A. Fahmy A survey of text similarity approaches Int. J. Comput. Appl., 68 (13) (2013), pp. 13-18 J. Qin, W. Wang, Y. Lu, C. Xiao, X. Lin

[8] D. Deng, G. Li, J. Feng, and W.-S. Li. Top-k string similarity search with edit-distance constraints. In *ICDE*, pages 925–936, 2013.

[9] J. Qin, W. Wang, Y. Lu, C. Xiao, and X. Lin. Efficient exact edit similarity query processing with the asymmetric signature scheme. In *SIGMOD Conference*, pages 1033–1044, 2011.

[10] S. Zhang, Y. Hu, and G. Bian, "Research on String Similarity Algorithm based on Levenshtein Distance," no. 1, pp. 2247–2251, 2017.

[11] Z. Zhang, M. Hadjieleftheriou, B. C. Ooi, and D. Srivastava. Bed-tree: an all-purpose index structure for string similarity search based on edit distance. In *SIGMOD Conference*, pages 915–926, 2010

[12] G. Li, J. Wang, C. Li, and J. Feng. Supporting efficient top-k queries in type-ahead search. In *SIGIR*, pages 355–364, 2012.

[13] Z. Zhang, M. Hadjieleftheriou, B. C. Ooi, and D. Srivastava. Bed-tree: an all-purpose index structure for string similarity search based on edit distance. In *SIGMOD Conference*, pages 915–926, 2010.

[14] Z. Yang, J. Yu, and M. Kitsuregawa. Fast algorithms for top-k approximate string matching. In *AAAI*, 2010.

[15] C. Xiao, W. Wang, X. Lin, and H. Shang. Top-k set similarity joins. In *ICDE*, pages 916–927, 2009.

[16] C. Li, J. Lu, and Y. Lu. Efficient merging and filtering algorithms for approximate string searches. In *ICDE*, pages 257–266, 2008.

Biographies

Ibrahim Musa Conteh, M.Sc. Computer Science, Huazhong University of Science and Technology (HUST), Wuhan, P.R. China; B.Sc. Information Systems, Institute of Public Administration and Management, University of Sierra Leone.

Gibril Njai, M.Sc. Information Engineering, Chongqing University of Posts and Telecommunications (CQUPT), Chongqing, P.R. China; B.Sc. BIT, Njala University, Sierra Leone.