A Hash-based labeling technique using Dynamic Programming for String Similarity Search

[1]Monika D. Randhavane, [2]Chandrakant R. Barde

Department of Computer Engineering
Gokhale Education Society's R. H. Sapat college of Engineering, Management Studies and Research, Nashik-05,

***Abstract*: Data over the search engine is developing tremendously so whenever client will give a query to a search engine it will find lots of match for that query. The results which are generated by the search engine are sometimes relevant or sometimes irrelevant. So, we need design such a framework which can shows the correct results for any query given by the client. For string similarity search we propose two hashing approaches namely ox label and xx label. It is the edit distance based approach and based on that we are measuring the similarity between two strings and shows the similar results. we also propose a hypergraph based approach in which compute the exact edit distance of two string with help of Levenshtein distance and calculate the similarity so that it will reduces the index size and index construction time.**

Index Terms: String similarity, similarity search, hash, edit distance, Hypergraph.

I. Introduction

String Similarity search:

In string similarity search we have given with a dataset, query string and threshold and we need to find similar strings to that query. It can be used in variety of applications which are data integration, Error checking, Database Management system, Bio logical sequence analysis, Data cleaning, Pattern recognition. Normally Strings consist of data in the form of text and which is come from different sources in different format. In existing, string similarity can be measured in two way first is token based similarity metric and another is character based similarity metric. Varieties of string similarity functions have been used to measure the similarity edit distance, Jaccard similarity, and cosine similarity. Edit distance between two strings can be computed depending upon the minimum number of edit operations.

Existing approaches uses the filter and verify framework. In filter step it can built an index for the dataset and verify those strings one by one in verification step. In existing approaches when strings are much longer it can built larger index and therefore it will take more time as compare to the short strings and hence performance gets decreases. The hash based approaches can efficiently deal with the both i.e short as well as long strings.

The challenge is we need to design an effective index so that it will take less time for query processing and also the index size gets reduce.

Similarity semantics:

Similarity is a word and which can have various implications with regards to PC documents. We take the view that all together for two records to be comparable they should share content. In any case, there are various approaches to characterize that sharing.

II. Related work

Most of the existing approaches for string similarity search uses the filter and verify approach. To find similar string in string dataset it requires query string and threshold for finding similar results to the query. The existing approaches can build large index when string becomes long. Large index require more time for processing the query hence, performance gets decreased.

Gravano et al. [1] Proposed n approach which can be applied to both full string as well as approximate string matching. The approximate string match predicate, with an acceptable edit distance threshold, are often mapped into a vanilla relative expression and optimized by standard relative optimizers.

Chaudhari et al. [2] proposed a fuzzy edit distance based methodology, the main aim of this approach is to provide roust and efficient fuzzy match applicable to all kinds of domains. The author implements an efficient and exact fuzzy match so that it can easily clean an incoming token fron various sources. They used Generalized Edit Similarity algorithm in which they take every string as a tokens and for this every token it will match with another string and depending upon that it assigns a cost with the help of edit distance.

Chaudhari et al. [3] proposed a SSJoin operator which is used for similarity joins based on variety of string similarity functions. They proposed SSJoin operator and for this operator can be efficiently implemented using sql operator. SSJoin operator can map the strings to the set and from that it can measure the similarity using set overlap. This operator can return distinct values. If there overlap is high between two sets then it is similar to subsets of two set are also overlap.

Li et al. [4] proposed a system that is variable-length-grams for improving the exhibition for similitude search issue. In the fix length gram a few grams may be some are normal and some are uncommon. To manage this issue, they select excellent grams to abstain from producing some basic grams. This methodology describes a way how to choose high-quality grams from the given collection to Support queries on the collection. It also generates index structure for variable-length grams in the collection of strings. It defines the relationship between the preselected grams, similarity of two strings and edit distance.

Yang et al. [5] proposed methodology compute the lower bound based on common grams which are used by two strings and affect the query performance in two ways. In first how we use the inverted index for q-grams in the query string. Second is how many resulted strings for the inverted index of the q-gram given query. The query should contains at least 2 common q-gram then it will shows the resulted string as a answer. They use new list every time with new bound for finding the answer to the query. In this way they have to add new q-gram list to the existing inverted list and hence that will affect the query performance. They select only the high quality grams from the dictionary or inverted list.

Xiao et al. [6] proposed an approach based on mismatching q-grams to speed up the query processing. They propose new algorithm called Ed-Join, it consider two filtering techniques, first one is the location based mismatch filtering method and another is content based mismatch filtering method. Ed-Join takes two phases for performing the query first is the candidate generation and another is verification phase. With the help of two methods it can reduce the size of candidate strings. The content based mismatch filtering is used to detect clustered edit errors which are frequently occur in real dataset. This algorithm require less time as compare to existing one and reduces the index size for query.

Bem et al. [7] the author studied that how to help keyword search on spatial information. To efficiently answering such queries is tedious task for this websites for achieving high output when serving concurrent users. It focuses on the tree-based spatial index structure having ability for finding approximate keyword search. They also developed algorithm to construct the spatial index structure also minimize the time and improving the space efficiency.

Zhang et al. [8] proposed Bed-tree, based on edit distance indexing structure for supporting collection of queries based on edit distance constraints also designed three different string transformation based on query for answering the queries. It uses the transformation function that implicitly aligns string space to integer space so that searching query and verify those query can be efficient. In this way it can provide an index structure for answering selection and join queries. This methodology shows good performance for long strings with larger threshold as well as for large dataset.

Li et al. [9] proposed PASS-JOIN method based on partition for similarity search. Previous approaches are not well supported for short as well as long string so, to address this issue they presented PASS-JOIN approach. In this approach it can select string and partition them into set of segments that is substring for each string and using that substring it can find out candidate string in an inverted index structure and hence minimize the number of substring which is selected. They also proposed an extension based method for verifying the candidate's string for improving the performance. This approach is best suited for both the short as well as long strings.

Feng et al. [10] proposed a straightforward trie search based methodology for similarity join based which only consider subtrie pruning, and used that framework for efficiently finding similar string for that query string. They given a two set of strings and try to find the similar strings from that given collection. This structure is used to efficiently finding the near duplicated string pairs and achieve high performance. They proposed partitioning approach when the threshold is large enough, in that case they partition it into two sub part first is the left-half part and another is the right-sub part. This approach requires less space as well as small index size.

Wang et al. [11] proposed a replacement approach edit similarity be part of supported chunk within the string. Initially they split the complete string into small number of chunks and from that chunk extract only the nonoverlapping chunk for the sub string. They also proposed the class of chunking which have tight lower bound for two strings. They also proposed another methodology called VChunk-join with several powerful filters with the existing filter in order to improves the performance in terms of efficiency and index size. They also used the CBD algorithm for finding a good chunking scheme.

Xiao et al. [12] proposed duplicate removal technique i.e, IncNGTrie for extracting query results. The neighborhood generation introduces algorithm for error tolerant query which is based on edit distance constraints. They likewise proposed indexing, searching, fetching techniques for processing the query and also developed optimization procedures to decrease index size as well improves the query processing time.

Deng et al. [13] proposed PIVOTALSEARCH method which includes two phases one is to build index offline and another is online query processing phase. They developed dynamic programming method for selecting high quality pivotal prefix signatures to prune the dissimilar strings with nonconsecutive errors. When there are consecutive errors in the query then it will generate many false positive. To address this issue they propose an alignment filter that considers the alignment between two signature and prune the dissimilar one with the consecutive errors.

J. Wang et al. [14] proposed cluster based B+-tree which reduces the I/O to minimum and performed faster than the existing one. They proposed iDistance method to partition the dataset into set of clusters that is indexing high dimensional data points in one dimensional space. They assign a partition for each string from the string that will reduce the number of strings in the candidate string.

Yang et al. [15] proposed a new filtering which considers the problem of approximate query processing based on edit distance metric. They propose the concept of local distance to identify minimum number of error which are occurred while processing that query string with dataset string. The local distance computes the dissimilarity between two strings. This approach will consider only the fixed length partitioning scheme. They also proposed index tree called Bit tree that indexes all the strings.

Wang et al. [16] proposed local similarity approach to find the duplicated text or duplicated documents for unstructured text. The replication is detected by using similarity search for technique in which they partition the text into number of tokens. They use the cost aware algorithm for finding good partitioning of token. It will select the token which having high signature. This approach can compare two strings if edit distance between them is within the given or specified threshold.

III. Proposed Methodology

3.1 Hash Based Approach:

In string similarity search, based on edit distance concept it will compare the edit distance of a given string with the threshold value if it is less than the given threshold then query string should be returned as an answer. For that purpose we propose a hash labeling technique in which we can create two hash labels namely OX label and XX label. Here, Hash label is simply a binary vector which containing only bit sequence that is 0 and 1 and which can be generated for any length of string. Initially we assign that hash label to a string and according to that pruning is done for the hash labels.
Basically the string is partition into given q-gram and then assign the memory locations to each q-gram in this way we can build an index for query string. In hash based labeling technique we consider only the dissimilar hash bits between two hash labels. In existing approaches, if there are occurrences of hash collisions then it is very difficult to manage it. Hence we propose a hash based labeling technique in which we can efficiently handle the hash collisions, so that no two gram will set to the same position.

3.2 System Architecture:

The framework of our hash based approach for string similarity search shown in Fig 1. In our proposed approach we used various steps for string similarity search purpose first we generate hash labels for strings, and then searching based on hash label and finally verify the two strings which are dissimilar by comparing two hash labels. Initially, required review dataset is selected and loaded. As well as give the query string as a input to the data preprocessing step. In second step of methodology, we give the value of q-gram and generate the q-gram for dataset as well as the query string. Then select unique q-gram from both and allocate the memory location for each q-gram.

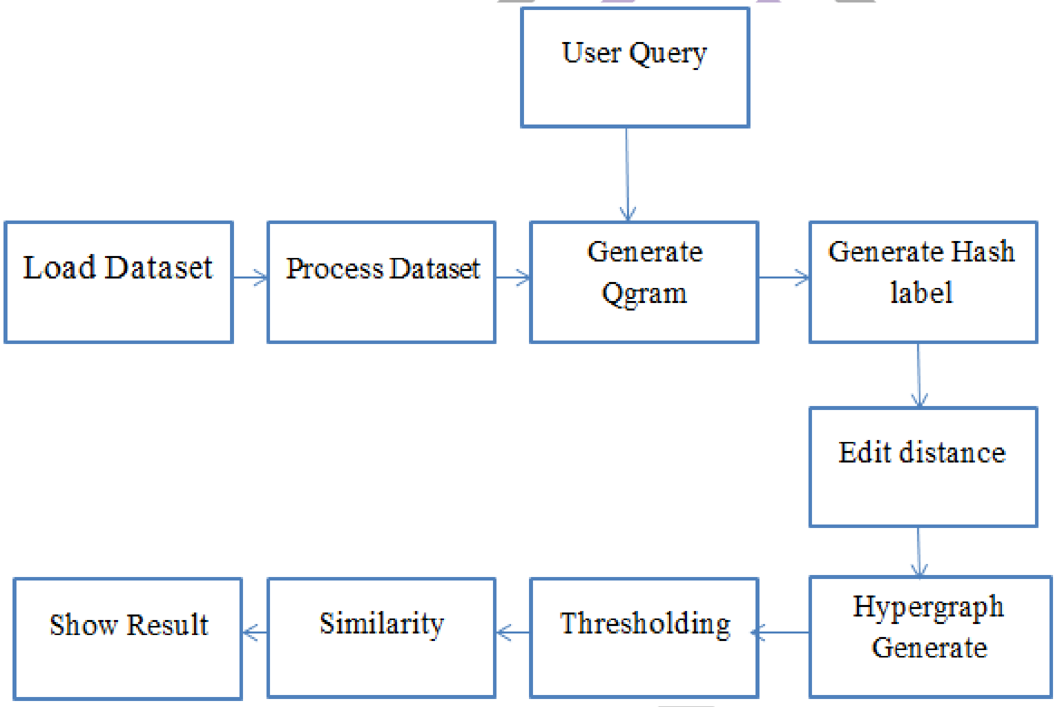


Fig.1 System Architecture

3.3 Algorithm:

In our system we use different algorithms which are as follows:
- Hash Label creation
- Hash Label comparison
- Verification

Hash label creation:

We use a hash function and assign that hash function to an input q-gram and place that q-gram in certain memory location. This function takes input as a character and from that we can build a hash table. It may be the possibility that more than one q-gram will occur in the same memory location so in that case we need to avoid this computation for that purpose we use a rolling hash function for avoiding repeated entries for hashing an input q-gram.
The rolling hash function can rotate the bit vector to the left by specified position and final hash value can be calculated. We saves

all that q-gram in the hash table and also use it by looking up in hash table and from that create a label for an input query. In making q-gram we can remove the stop words also so that time to build a hash table is becomes effective and size also compact.
In hash label creation, it is simply a bit vector. Initially all bits are set to 0 in the hash label. The q-gram is padded with the delimited character $ and ended with the same character. In this way we can create a q-gram for dataset as well as for the user query. Then will match query q-gram with the dataset q-gram a match found then it will set the bit position by 1 in the hash label if no match found then this bit becomes 0.
For label creation, it mainly consists of bitwise operator that means we consider only the exclusive-or operation. In this way we can generate label for query string and dataset string. After that performing the ex-or operation on that two labels that means if two bit input are same then result is 0 otherwise the result is 1 and finally generate one label from that two labels. For ox label we use bitwise ex-or operator and generate the label. For xx label we also use the ex-or operation but the collision problem can be efficiently handled by this label.

Hash label comparison:

After creating hash label this algorithm will takes both the hash label that is ox and xx labels as an input. Then for each hash label of a string in a dataset it checks the following condition:

$$\#(\oplus(H_s, H_t) \leq 2q\tau - ||s| - |t||)$$

Where,
# (.) is a function ,which the number of 1 in the hash label.

In hash label comparison we compute the Edit distance between two hash label.

Verification:
In verification step we use the concept of Levenshtein distance, in which we compute the exact edit distance between two strings. In verification it only checks the edit distance between two strings with the given threshold. Finally calculate the similarity and depending on that similarity similar results will retrieve. Similarity should be 1 when there is exact matching otherwise it will vary in between 0 and 1.

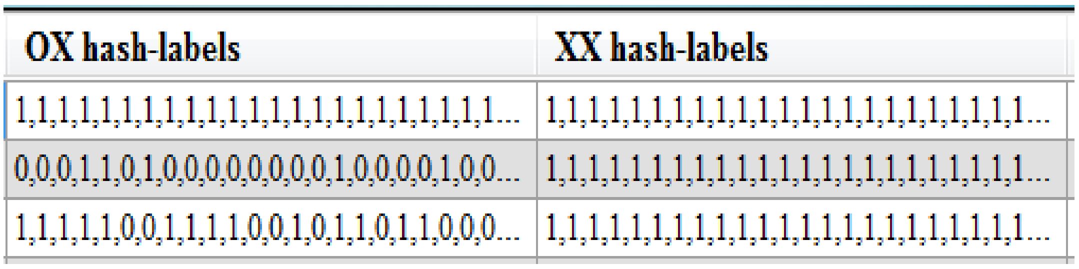
IV. RESULTS

The dataset contains different type of text which is given bellow. Food Review consists of reviews of fine foods from Amazon. The statistics of the dataset, including the product and user information, rating and a plain text review. It also includes reviews from all other Amazon categories.

Table 1 Dataset

Dataset	Product	User information	Ratings	Review
Food Review	74,258	256,059	Between 1 and 5	568,454

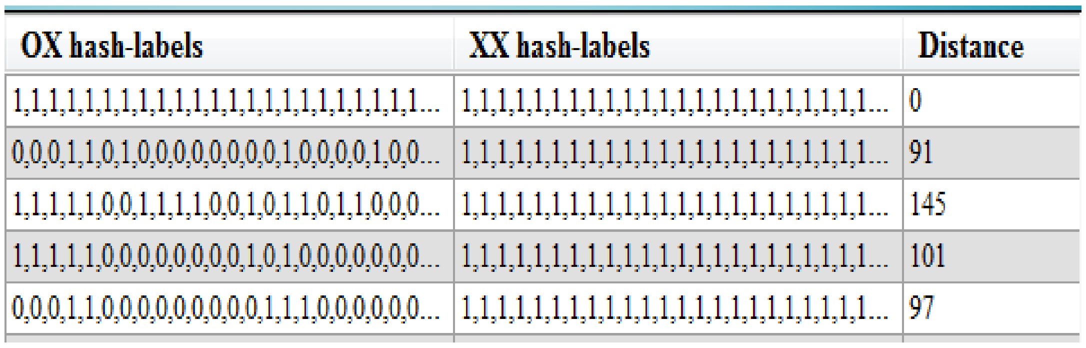
Here we are giving review dataset as an input to the system as well as query string for generating hash labels. The labels are in the form of vector composed 0 and 1. The figure 2 shows the ox label and xx label.



OX hash-labels	XX hash-labels
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1...	1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1...
0,0,0,1,1,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0...	1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1...
1,1,1,1,1,0,0,1,1,1,1,0,0,1,0,1,1,0,1,1,0,0,0...	1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1...

Fig.2 OX Label Creation

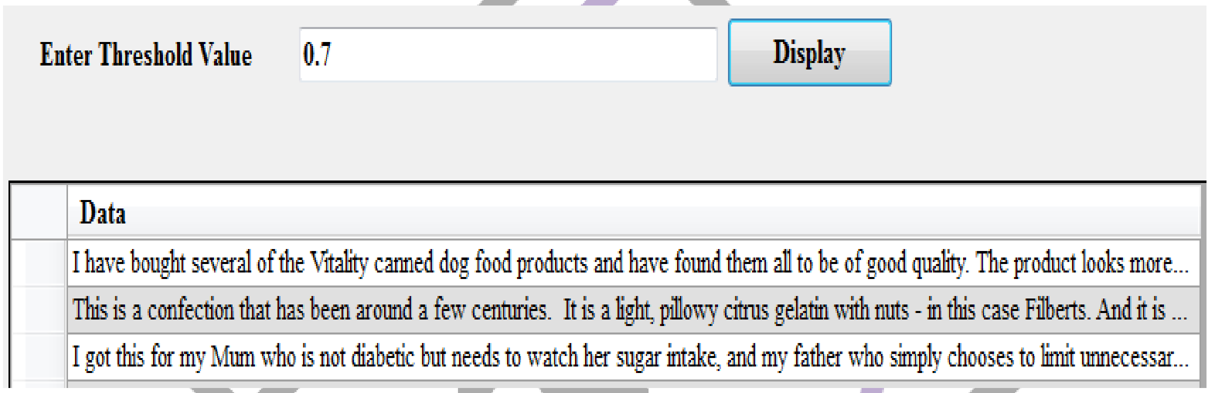
We will use both labels for comparison and count the number of different bits. That means calculating the edit distance between two strings which is shown in figure 3.



OX hash-labels	XX hash-labels	Distance
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1...	1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1...	0
0,0,0,1,1,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0...	1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1...	91
1,1,1,1,1,0,0,1,1,1,1,0,0,1,0,1,1,0,1,1,0,0,0...	1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1...	145
1,1,1,1,1,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0...	1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1...	101
0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0...	1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1...	97

Fig.3 Edit distance calculation

After that applying a dynamic programming approach by using Levenshtein distance for calculating exact edit distance between two strings and calculate similarity score. Then applying a threshold value and retrieve only those records that are similar to the query string which is shown in the figure 4.



Enter Threshold Value 0.7 Display

Data
I have bought several of the Vitality canned dog food products and have found them all to be of good quality. The product looks more...
This is a confection that has been around a few centuries. It is a light, pillowy citrus gelatin with nuts - in this case Filberts. And it is ...
I got this for my Mum who is not diabetic but needs to watch her sugar intake, and my father who simply chooses to limit unnecessar...

Fig.4 Applying threshold

The based approaches require more time for creation of hash label and then calculate the edit distance. The proposed approach requires less time as compared to the based one which results is shown in the figure 5.

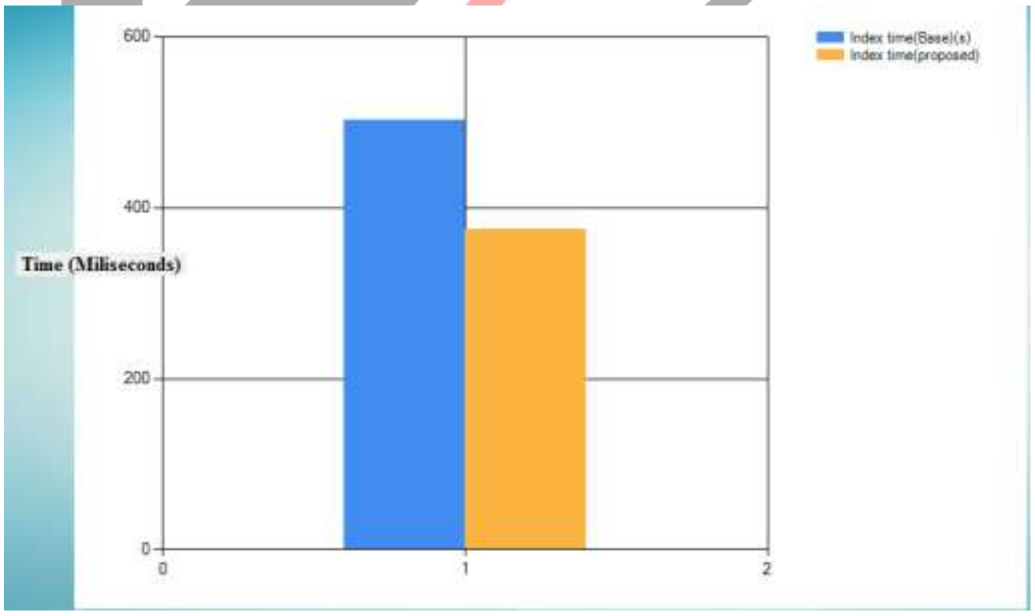


Fig. Index time

V. CONCLUSION

We developed two labels for string similarity search based on edit distance. Both can be used for prune the dissimilar strings. Pruning power of Ox label is effective when length is large for two strings. Pruning power of xx label is depends on different set of q-grams and effectively used for short and long strings. Index size and index construction time is less. Here we developed a hash function which contains group of hash keys and data string to determine the string similarity. We define the string similarity based on the hash label.

VI. ACKNOWLEDGMENT

I have a tremendous pleasure in presenting the project "A Hash based labeling technique using dynamic programming approach for String Similarity Search" under the guidance of Prof. C. R. Barde and PG coordinator Prof. A. S. Vaidya. I am really obligated and appreciative to Head of the Department Dr. D. V. Patil for their significant direction and consolation. I might likewise want to thank the Gokhale Education Society's R. H. Sapat College Of Engineering, Management Studies Research, Nashik-5 for giving the required offices, Web get to and vital books. At last I must express my sincere heartfelt gratitude to all the Teaching Non-teaching Staff members of Computer Department of GESRHSCOE who helped me for their important time, support, remarks, thoughts.

REFERENCES

[1] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *Proc. of VLDB'01*, 2001.
[2] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In Proc. of SIGMOD'03, 2003.
[3] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In Proc of ICDE'06, 2006.
[4] C. Li, B. Wang, and X. Yang. VGRAM: improving performance of approximate queries on string collections using variable-length grams. In Proc. of VLDB'07, 2007.
[5] X. Yang, B. Wang, and C. Li. Cost-based variable-length-gram selection for string collections to support approximate queries efficiently. In Proc. of SIGMOD'08, 2008.
[6] C. Xiao, W. Wang, and X. Lin. Ed-join: an efficient algorithm for similarity joins with edit distance constraints. PVLDB, 1(1), 2008.
[7] A. Behm, R. Vernica, S. Alsubaiee, S. Ji, J. Lu, L. Jin, Y. Lu, and C. Li. UCI Flamingo Package 4.1, 2010.
[8] 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 Proc. of SIGMOD'10, 2010.
[9] G. Li, D. Deng, J. Wang, and J. Feng. PASS-JOIN: A partition-based method for similarity joins. PVLDB, 5(3), 2011.
[10] J. Feng, J. Wang, and G. Li. Trie-join: a trie-based method for efficient string similarity joins. VLDB J., 21(4):437–461, 2012.
[11] W. Wang, J. Qin, C. Xiao, X. Lin, and H. T. Shen. Vchunkjoin: An efficient algorithm for edit similarity joins. TKDE, 25(8), 2013.
[12] C. Xiao, J. Qin, W. Wang, Y. Ishikawa, K. Tsuda, and K. Sadakane. Efficient error-tolerant query autocompletion. PVLDB, 2013.
[13] D. Deng, G. Li, and J. Feng. A pivotal prefix based filtering algorithm for string similarity search. In Proc. of SIGMOD'14, 2014.
[14] W. Lu, X. Du, M. Hadjieleftheriou, and B. C. Ooi. Efficiently supporting edit distance based string similarity search using B+-trees. *TKDE*, 26(12), 2014.
[15] X. Yang, Y. Wang, B. Wang, and W. Wang. Local filtering: Improving the performance of approximate queries on string collections. In Proc. Of SIGMOD'15, 2015.
[16] P.wang, C. Xiao, J. Qin, W. Wang, X. Zhang, and Y. Ishikawa. Local similarity search for unstructured text, In Proc. of SIGMOD'16, 2016.