TRIE BASED METHODS FOR STRING SIMILARITY JOINS

Venkat Charan Varma Buddharaju

#10498995

Department of Computer and Information Science

University of MIssissippi

ENGR-654

INFORMATION SYSTEM PRINCIPLES

RESEARCH TRACK FOR FIRST HALF
ABSTRACT:

A string similarity join finds similar pairs between two collections of strings. It is a metric that measures similarity or dissimilarity between two strings for approximate string matching or comparison in fuzzy string search. Its major applications include data integration, data cleaning and matching searches in search engines. Conventional string similarity join approaches using trie-based algorithms the process is very efficient for short length of strings. The existing processes calculate the prefix nodes known as active nodes for every search operation and these get piled up when string are of larger lengths. My research work is a study of two different techniques that have been discussed in two different conference proceedings. The two efficient Trie-based string similarity join that have been studied and evaluated are Pre-join and Trie-join. Pre-join finds all similarity string pairs using a new active-node generation method, and dynamic preorder traversal of the Trie index. Trie-join uses a trie structure to index the string and utilize the trie structure to efficiently find similar string pairs based on subtrie pruning. Additionally, trie-join algorithms and trie algorithms are used to gain higher performance. All these algorithms are discussed in this paper. Later the experiments performed are analyzed to get a clearer picture of these methodologies.

INTRODUCTION TO TRIES:

A Trie also known as digital data structure is an ordered data structure that is used to store a dynamic set or associative array where the keys are usually strings. Values are normally not associated with every node, only with leaves and some inner nodes that correspond to keys of interest.

In the figure shown, keys are listed in the nodes and values below them. Each complete English word has an arbitrary integer value associated with it. A tree can be seen as a deterministic finite automation with loops. Though tries are most commonly keyed by character strings, they don’t need to be.
Tries can be more advantageous in many scenarios when compared to Hash tables and Binary Search Trees. Unlike Binary Search Trees, no node in the trie stores the key associated with that node, instead, its position in the tree defines the key with which it is associated. There are no collisions of keys in tries that happen with hash tables. There is no need to provide a hash function or to change hash functions as more keys are added to a trie. Some of the major applications of tries are:

1. Dictionary Representation
2. Sorting
3. Full Text Search

**INTRODUCTION TO SIMILARITY JOINS:**

Similarity measure is a metric that quantifies the similarity between two text strings. String similarity join is the task of finding similarities between two given texts by making the minimal number of alterations like additions, deletions, swaps so that the first given string becomes the second given string. String similarity join is an important operation in data integration and cleansing that finds similar pairs from two collections of strings. The vast range of applications of string similarity joins include fraud detection, fingerprint analysis, plagiarism detection, ontology merging, DNA & RNA analysis, image analysis, evidence-based machine learning, data mining, web interfaces, semantic knowledge integration. String similarity algorithms
commonly used are Levenshtein distance or edit-distance, Needleman-Wunsch distance, Smith-Waterman distance, Gotoh distance or Smith-Waterman-Gotoh distance, Block distance or L1 distance, or City block distance, Jaro-Winkler distance, Soundex distance, Dice's coefficient, Tversky index, Overlap Coefficient, Variational distance, Skew Divergence, Confusion Probability, Maximal matches, Lee Distance, etc. These are the metric used for string similarity calculation. The widely used technique for measuring the similarity of strings is Edit distance. It operates between two input strings, returning a score equivalent to the number of subscriptions and deletions required to transform one input string to another input string.

The conventional methods of Similarity joins have the following drawbacks that lead to the introduction of new techniques like Trie-join and Pre-join:

1. They are inefficient for the data sets with short strings.
2. They involve large indices.
3. They are expensive to support dynamic updates of data sets.
4. They create many active nodes that need to be removed again in future.

Calculating the active node:

Given two strings \( r = r_1r_2 \ldots r_n \) and \( s = s_1s_2 \ldots s_m \), let \( D \) denote a matrix with \( n+1 \) rows and \( m + 1 \) columns, and \( D(i, j) \) be the edit distance between the prefix \( r_1r_2 \ldots r_i \) and the prefix \( s_1s_2 \ldots s_j \). We use the dynamic-programming algorithm to compute the matrix: 

\[
D(0, j) = j \text{ for } 0 \leq j \leq n, \quad D(i, j) = \min\{D(i-1, j)+1, D(i, j-1)+1, D(i-1, j-1)+\theta\}
\]

where \( \theta = 0 \) if \( r_i = s_j \); otherwise, \( \theta = 1 \).

Here, \( D(i,j) \) is called an active entry if \( D(i,j) \leq \tau \).

There is an example illustrated in (Feng, Wang, & Li, 2012) that gives a better calculation of active node entries for the given trie structure. It calculates the edit-distance for every element of the given input string and enters it into the \((n+1)*m+1\) matrix.
The \((n+1 \times m+1)\) matrix gives the values of edit distance between any two elements of the given input string. The active nodes can clearly be noted by picking the values that are below the given threshold.

**TRIE-JOIN**

Trie-join works basically on two observations, one being subtrie pruning and the other being dual subtrie pruning. Subtrie pruning states that in a given trie and string \(s\), node \(n\) in said to be active node of the string \(s\) if the edit distance, \(ED(s,n) \leq \tau\). If \(n\) is not an active node of every prefix of string \(s\), then all the contents of \(n\) cannot be similar to \(s\). Dual subtrie pruning states a relation between two trie \(u\) and \(v\), where the string under \(u\) and \(v\) cannot be similar to each other if \(u\) is not an active node of every ancestor of \(v\) and \(v\) is not an active node of every ancestor of \(u\).

There are three different algorithms that have been discussed in (Feng, Wang, & Li, 2012)- Trie-Trversal, Trie- Dynamic and Trie-Pathstack. There are other algorithms that have been explained that work for supporting dynamic data updates and also for similarity joins for two different sets. In addition, there are algorithms that have been developed for improving the already developed Trie- Pathstack algorithm to work for larger edit- distances.

**Trie-Traverse:**

To be more particular, Trie- Traverse has been developed in the intension of improving the standard Trie-Search algorithm. In general, Trie-Search computes duplicate active nodes and this computation overload is avoided in Trie-Traverse by dual subtrie pruning. The process of Trie-Traverse is that it first draws the trie structure for both the given input strings. After construction
of the tries, the active-node set is computed for every node only once in the whole process irrespective of the node being a prefix of large number of strings. The process of Trie-Traverse can be explained with the expansion of an algorithm. With the given string collection $S$ and a given edit distance threshold, a new Trie is constructed for a given string. Active node set is computed for the root node $r$ and the output variable $P$ is united with a function findSimilarPair of child node $c$, root node $r$ and active node set of root node $r$. Now findSimilarPair is described with an input of $c$, a tire node or a child node, the root node and active node set of root node. Active node set of the child node is computed and pruning of the set is performed. This is a recursive function until the current node on which operations are done is a leaf node. If the current node is a leaf node, then outputSimilarPair function is called on the current node and active node set of the current node. Pruning is performed again in the outStringPair function which is called for the output pair representation.

**Trie-Dynamic:**

In this part, a new algorithm has been designed with the consideration of symmetry property of active nodes. The main idea is that if $u$ is an active node of $v$ then $v$ must and should be an active node of $u$. If the computations are done under the basis of this property, unnecessary computations can be avoided. Trie-Dynamic is an algorithm with a collection of strings, given edit distance threshold and an output of similar string. A trie structure has been constructed and a tire node has been found among the given input stings which is the longest prefix among the given strings. A for loop has been devised for repeating the process until the current node of the trie reaches the child node. Active node set of the current node is computed and appended to the existing set. The function outputSimilarPair is called upon the leaf node that is when the loop reaches the leaf node.

**Trie-Pathstack:**

The above proposed two algorithms are effective but only in one dimension i.e. Trie-Traverse uses very little memory but computes unnecessary active nodes. In contrast Trie-Dynamic computes just required number of active nodes but uses more memory. Trie-Pathstack has been devised in such a way that it can overcome the above mention problems. It integrates both the ideas of Trie-Traverse and Trie-Dynamic. For achieving this, while traversing the trie nodes,
virtual partial subtrie is maintained to keep a record of visited nodes. When active nodes are computed for unvisited nodes, they are first considered as visited nodes, and the active node set is computed by assuming it as a part of the virtual partial subtrie which avoids the redundant computation. For less memory usage, we traverse the nodes in preorder and a stack is maintained for the nodes that need to be updated. For current node active node computation, the stack is visited as the stack contains active node set of all nodes from the parent node to the current node. For computing the active node set of current node, the active node set of its parent is viewed and it helps the active node computation. Since, the Trie-Pathstack uses the symmetry property of active nodes, it has the same time complexity of Trie-Dynamic and the space complexity is same as Trie-Traversal.

Trie-Pathstack is an input of collection of strings and a given edit-distance threshold. Trie is computed for the given collection of strings and a new stack is initialized. The root node $r$ is considered and it is set as visited in the virtual partial and active node is computed and it is pushed into the stack. First child of the root node is considered as $c$ and when stack is not empty and the current first child node is not null the top element of the stack are the parent node and the active node set of the parent node and it is set to visited and the active node set is calculated for the first child and pruning is done simultaneously. The active node set is updated in the stack and then the process is repeated for its child nodes till the child node is reached.

**Pruning Techniques:**

Dual subtrie pruning has been used to develop three trie based algorithms. Now to further improve the algorithms to reduce the active node set, three types of pruning have been further introduced:

**Length Pruning:**

Considering two subtries $u$ and $v$, a range is maintained for both the nodes with the length of the shortest string and the length of the largest string. If the difference between these lengths is greater than the edit-distance threshold, then pruning for second node $v$ can be done by using the active nodes for the first node $u$. 
Single-Branch Pruning:

Considering two nodes $u$ and $v$, and say that $v$ is an ancestor of $u$ and their nodes have same leaf nodes, then second node $v$ can be pruned from active node set of first node $u$, even if $v$ is a node of $u$. Since, there is only a single branch from first node $v$ to second node $u$, new active nodes are not generated for $v$. This is called single-branch pruning.

Count Pruning:

Given two nodes $u$ and $v$, if there is only one string that has both nodes $u$ and $v$ as prefixes, node $u$ can be safely pruned from $A_v$ because we cannot find two strings in their subtries.

There are three other algorithms that have been developed for improving the performance of the above mentioned Trie-Join algorithms.

1. Incremental Trie-Join Algorithm that supports dynamic data update.
2. Trie-Pathstack+ algorithm for two different data sets.
3. Incremental Trie-Pathstack+ pr BI-Patjstack algorithm for larger edit distance thresholds.

All the above mentioned algorithms have been explained with theorem, algorithm and example in (Feng, Wang, & Li, 2012).

PRE-JOIN

In general, Trie-based similarity join approaches do the computation of active nodes in a different phase and also the generated active nodes are false candidates. Both these issues turn out to be computation overload for the process. To overcome this, a new active node generation method has been devised that just computes the required active nodes and the pruning phase can be eliminated. This can be very helpful for larger string lengths. So, Prejoin is a combination of preorder traversal and the new active node generation method. There are three steps in Pre-join. First, the active node set is calculated for the siblings of the current node while it is computing the active node set of the next to-be visited node. Next, Prejoin has its own order of traversal in contrast to conventional preorder traversal. Finally, the new active node generation method is employed that avoids the adding false active nodes into the set.
Presjoin is an algorithm with a collection of strings and a given edit distance threshold. Trie is constructed for the given set. PreorderTravers is performed on the root node which imposes an order on the children and they are considered as set. If the current node is an EOS, then Out_Similar function of the current element, children of current node, set of active nodes of that node, its value and edit-distance threshold is called. If the element is leaf node, a function of Gen_ActiveNode of the same variables mentioned above is called. Pre Traverse of the current element is performed. In the Gen_ActiveNode() function, for each node belonging to the active node set at a given distance, if cth power of ith node is equal to cth power of the current node, then a Push down operation is performed on the task with the ith node, current node, distance, threshold and ‘1’ else Push down operation is performed on ith element, current element, distance, threshold and ‘0’.

While computing the active nodes, false nodes can be avoided by imposing a couple of rules that are a part of the new active node generation method.

Rule 1: Apply the symmetry property of edit distance early in the generation process.

Rule 2: During the generation of the active node set from parent’s active node set, remaining siblings are not added to the set.

**COMPARING THE EXPERIMENTAL RESULTS**

**TRIE-JOIN:**

The experiment has been performed on the following datasets:

1. **English Dict-** Contains English words that have been derived from Aspell spell- Checker.
2. **DBLP Author-** Author names from DBLP dataset.
3. **AOL Query Log-** One million random distinct queries
4. **DBLP Authors+Title-** Strings are concatenation of Authors names and title of publication.
The experiments are performed on the four different algorithms: Trie-Search, Trie-Traversal, Trie-Dynamic, Trie-Pathstack. The latter three algorithms are compared to the standard Trie-Search algorithm. As observed in the statistics provided in (Feng, Wang, & Li, 2012), the three Trie-based algorithms work better than the standard Trie-Search. All these algorithms Trie-Dynamic and Trie-Pathstack use the symmetry property of active node computation, which lacks in Trie-Traversal. That is, the computation overload has been decreased in the Dynamic and Pathstack algorithms. Graphs stated also show that Trie-Traversal is approximately two times slower than Trie Dynamic and Trie Pathstack for this very reason. The number of active nodes for Trie-PathStack is smaller than that of Trie-Search and Trie-Traverse, since Trie-PathStack utilizes the symmetry property of two active nodes. The below graph shows the comparison of four algorithms.
The graphical representation of comparison of running time between Trie-Pathstack and Bi-Trie-Pathstack can be shown in the following figure. Bi-Tire-PathStack performed very well with higher edit-distance threshold for which it has been designed. Conventional Trie-Pathstack worked well for smaller edit-distance thresholds.

The comparison of three algorithms namely Ed-join, Trie-Path Stack and Bi-Trie-PathStack have been represented in graphs on the basis of Length of strings and Time scale with different edit distance thresholds. With different, edit distance thresholds, the algorithms performed similarly except for small changes. Ed-Join and Bi-Tire_PathStack were more efficient with larger string sizes and less efficient for smaller string sizes. The performance of Trie-PathStack has been very good with smaller string size and its performance degraded with larger string size by gradual increase in time.
**PRE-JOIN:**

The experiment has been performed on the following datasets: DBLP Authors, DBLP Authors+Title, AOL Query Log.

The Pre-join algorithm with the novel active node generation algorithm is compared against the Trie- traverse algorithm and Trie- Pathstack algorithm for different edit distance thresholds. The graphs have been drawn from (Gouda & Rashad, 2012) and they give a clearer picture of the experimental results.

Comparing Pre-Join, Trie- Pathstack and Trie- Traverse on DBLP Authors dataset with different edit distance thresholds 1, 2 and 3.

Comparing Pre-Join, Trie- Pathstack and Trie- Traverse on AOL Query Log dataset with different edit distance thresholds 1, 2 and 3.
Comparing Pre-Join, Trie- Pathstack and Trie- Traverse on DBLP Authors+Title dataset with different edit distance thresholds 1, 2 and 3. If all the above graphs are observed, it clearly shows that Pre-join works very efficiently by taking the minimum of time among Pre-join, Trie-Pathstack and Trie- Traverse. As the string size increases, the latter two algorithms started to work less efficiently whereas Pre-join is working at the shortest time.

CONCLUSION

This paper analyzed the techniques followed by two different groups of researchers. Both these techniques have studied the problem of string similarity joins with edit-distance constraints. Trie-join states a trie- based similarity join framework and also introduced many pruning techniques to enhance the performance of the state algorithms. To effectively serve the scenarios with large thresholds, improved algorithms have been devised with Trie- Search. Experimental Results show that the Trie- Join algorithms like Trie- traversal, Trie- dynamic, Trie- Pathstack work more efficiently compared to standard Trie- Search algorithm showing their superiority over the latter one. Pre-join proposes techniques that finds all similar pairs using a new active-node set generation method which helps in reducing the computation overload of active nodes in conventional trie based algorithms. It also proposes a dynamic preorder traversal of the Trie index. The experimental results show that the Pre-join serves very well for large and small datasets and works very efficiently for large edit distance threshold. Both these techniques work very well on paper and look interesting when the results produced my different research groups have been analyzed. No real time applications have been found that have used these methodologies to get interesting results.

References


