Abstract

Since a traditional database doesn't support approximate string matching, a query should return a result set that is more similar to the substring rather than returning an empty result set in the case that the substring doesn't match the string. This is done using an uncertain predicate operator and approximate string matching algorithm (Q- Gram). In order to provide more accurate results, the Q-gram algorithm has been changed, although this modification increases the time complexity (O (|x|2. |y|)) while decreasing the space complexity (O (|Q (x; q) |)) compared to the naive technique for string matching. In this article, the levenshtein distance algorithm is utilized by the rapidfuzz package to calculate the differences between strings for approximate string matching. The result set is sorted and presented to the user based on the similarity value and length of the strings. Python is used for the ASM's implementation. All studies use modified versions of the IMDB movie dataset as their sample dataset. In addition to having a far lower time complexity than the prior methodology utilized for string comparison, the proposed methodology's accuracy is also significantly higher. Through certain sample queries, the time complexity and accuracy of the approach under various instances have been noticed and reported, demonstrating the performance difference between the prior and present systems.

Specific Learning

- Q-Gram string matching algorithm.
- Levenshtein distance string matching algorithm.
- Text mining.

CHAPTER 1

INTRODUCTION

1.1 String Matching in Deterministic Database

The core clauses of each SQL query in conventional databases are select from and where. A general SQL query accepts the relations listed in the from clause as input, eliminates tuples that don't match the criteria listed in the where clause, and then picks the characteristics listed in the select clause. By employing the like operator to conduct pattern matching in a deterministic database, the query ultimately produces a filtered relation as an output. Two special characters, "%" and "_," are used to describe patterns. The letter "_" matches a single character, while "%" matches any string, i.e., any number of characters. The like operator in SQL is used to express the pattern. Consider the following query select book_title from books where book_title = '%Computer%'; this search returns, as a substring, the titles of books that contain the word "Computer." The where clause's conditions are used to compare the expressions. The expression might be a relational column or a constant, like an alphanumeric string, a number, a date, or a string literal (also known as a text literal or text constant). In the where clause, we can employ the comparison operators, =, and > as well as like operators. The approximate matching of the values is obviously not supported by the standard database.

1.2 Need of Approximate String Matching

The rough matching of the values is obviously not supported by the conventional database. Only the pattern defined by the like operator, which is identical to substring matching, can be matched by it. In order to prevent the query from returning an empty result-set, the user must have in-depth knowledge of the database's data. A single typographical error or misspelt word can potentially result in the same outcome. Due to the homophony of the words, issues could occur. Consider the words Cristopher or Kristopher, which are pronounced similarly yet have different spellings. Exact string matching fails when strings differ by even one character. As a result, the absence of support for approximation matching in the current deterministic database is a concern. It would be preferable to return the result set ranked by the degree of resemblance rather than returning an empty result set due to the aforementioned reasons, such as ignorance of the current data, misspellings, typing errors, etc. The casual user, who knows very little about the data that is kept, might benefit from this. The user would learn about the actual values in the database from the approximative answers he would receive, and he could then re-query with the precise keywords she had in mind.

1.3 Probabilistic String Matching

If there isn't an exact match, one can use approximate string matching to locate a string that nearly resembles the input string. It is a technique for evaluating a measure (metrics) of match between two strings by comparing them.

1.3.1 Application areas of string matching algorithms

- Plagiarism Detection
- Bioinformatics & DNA Sequencing
- Spelling Checker
- Search Engines
- Spam Filters

1.3.2 String matching algorithms can be classified into two types of algorithms -

1. Exact String Matching - Exact string matching algorithms is to find one, several, or all occurrences of a defined string (pattern) in a large string (text or sequences) such that each matching is perfect.

2. Approximate String Matching - Approximate String Matching Algorithms (also known as Fuzzy String Searching) searches for substrings of the input string.

1.3.3 Various approximate string matching techniques -

- Hamming Distance Algorithm.
- Levenstein Distance Algorithm.
- Damerau Levenstein Distance Algorithm.
- N-Gram Algorithm.
- BK Tree Algorithm.
- Bitap Algorithm.
- Boyer Moore Algorithm.
- Karp-Rabin Algorithm.
- Smith Waterman Algorithm.

CHAPTER 2

OBJECTIVE

According to our understanding, database systems still lack support for uncertain predicates, making approximate string matching impossible. In this study, I expanded the concept of searches by ensuring that strings with the greatest resemblance to searched strings should be displayed even if the sought string is not present in the dataset. I will suggest the optimal way of interest after examining different existing ASM approaches. I will make the necessary adjustments to the suggested algorithm to improve its accuracy and time complexity in comparison to the q-gram technique. In all of my research, I'll utilize a modified version of an IMDB movie dataset as a sample dataset. I'll display the outcomes of various searches on the sample dataset together with the precision and speed of the suggested solution and the original solution.

CHAPTER 3

METHODOLOGY

3.1 Previous System Methodology

It introduces the "" operator for the uncertain predicate in order to compare the values in terms of distance. Distance between the strings, which is also a concept of dissimilarity between them, can be used to create such an operator. Preprocessing distance computation, distance normalization, and probability calculation are the system's fundamental modules. The parser performs approximation matching on the columns involved in uncertain predicates when it comes across the symbol "" in the where clause of the query. Prior to applying uncertain predicates to the filtered result set, all predicates, with the exception of uncertain predicates, are applied. These procedures make up our system's preprocessing phases. To determine the distance, the Distance Calculation Module measures the separation between the literal that was requested and each of the field values in the relevant column. The Distance Normalization Module normalizes the distance array in the [0, 1] range. For each of the uncertain predicates, the aforementioned procedures are repeated. In order to determine the final probability of filtered tuples and, ultimately, to provide a probabilistic result set, the Probability Calculation Module integrates the probabilities obtained from the uncertain predicates.

3.1.1 Local q-gram distance

The local q-gram distance is the q-gram separation between a substrings of the bigger string that is the same length as the smaller string. In order to match the two strings, the bigger string (y) is divided into a collection of substrings (yi's), each of which is the same length as the shorter string (x). Let the lengths of the strings x and y be lx and ly, respectively. Let n be the number of potential substrings (yi's) of length lx of a string y, where n = ly lx + 1.

3.1.2 Global q-gram distance

It is suggested to calculate the global q-gram distance between strings x and y as follows: d(x, y) = min (d1, d2,..., dn) +(ly - lx) / lx + sum of di's / maxdist (x, y). Here, the local q-gram distances between two strings have been determined using the method described above. The fine-tuning terms, which contribute to a more precise distance calculation, are the next two terms. A maxdist(x, y) is the total of the maximum distances that may be travelled between each x and each yi, so maxdist(x, y) = 2n (lx q + 1).

3.1.3 Normalization of distance array

The range of values for each distance array may vary. The following formula is used to normalize distance arrays in the range [0, 1] for the homogeneity in order to obtain the probability array: p[ ] = 1 – ((dist[ ] – min)/(max – min)). Because d = 0 indicates an exact match but p = 1 indicates an exact match in the probability idea, the probability of a match is equal to one minus the normalized score.

3.2 Related Work

Table 3.1 Literature Survey

Research Paper	**Year**	**Approximate String Matching Algorithms**	**Time Complexity**
Survey and comparison of string matching algorithms chayapathi a r, g sunil kumar, Manjunath swamy be, thriveni j, venugopal K.R.	2021	Levenstein Distance	O(N+M)
		Damerau Levenstein Distance	O(M*N)
Supporting uncertain predicates in DBMS using approximate string matching and probabilistic Databases, AMOL S. Jumde & ravindra b. Keskar , 0.1109/access.2020.3021945.	2020	N-Gram	$(O(\|x\|^2.\|y\|))$
Exact string matching algorithms: survey,issues, and future research directions saqib iqbal Hakak, amirrudin kamsin, palaiahnakote shivakumara, gulshan amin gilkar, Wazir zada khan, and muhammad imran, 0.1109/access. 2019.2914071.	2019	Bitap	O(N)

3.3 Proposed System Methodology

The rapid fuzz library has been used in this article to approximate string match, while the rapid fuzz library uses the Levenshtein distance approach to determine the difference between strings. The user is presented with a sorted result set according to the length of the strings and the similarity value. Such ASM is implemented using the Python programming language. The IMDB movie dataset includes data on more than 5000 movies, with some tweaks, such as fewer features to lessen dataset complexity and new attributes to test algorithm effectiveness in various scenarios. In addition to having a significantly lower time complexity than the previous method utilized for string comparison, the proposed method's accuracy is also improved. Through several sample queries, the method's time complexity and accuracy have been noticed and reported, which demonstrates the performance difference between the past and present methodologies.

3.3.1 Rapidfuzz

Another string matching library that computes string differences and has a lot more to offer is called RapidFuzz. This was primarily created in C++ to speed up the string matching process. Two primary modules are:

1. Fuzz Module
2. String Metric Module

3.3.1.1 Fuzz module

It provides all the many options for contrasting two strings. Among them are:

1. **Ratio:** It calculates the normalized distance.
2. **Partial Ratio:** It calculates the ratio similarity measure between each substring of length m of the longer string and the shorter string, and it then returns the highest of these similarity measures.

3. **Token Set Ratio:** Compares the words in the strings based on unique and common words between them. It takes a set of all the tokens in the string and then compares it.

4. **Token Sort Ratio:** Sorts the words in the strings and calculates the fuzz ratio between them.
5. **W Ratio**: Calculates a weighted ratio based on the other ratio algorithms. It depends on the number of times a word occurs, order of the tokens, etc.

3.3.1.2 String metric module

This module is responsible for the type of various edit distance, more similar the metric, greater the edit distance between two strings.

1. **Levenshtein:** Calculates the minimum number of Levenshtein insertions, deletions, and substitutions needed to transform one sequence into the other while accounting for unique costs for each.

3.3.2 Filtering result set

After getting resultant set of strings we will sort it in ascending order based on the similarity value & in descending order based on the number of words in strings matched with searched strings.

3.3.3 GUI

I have provided the graphical user interface (GUI) for system which is not only user friendly but also help the users with the minimal knowledge of database to use our system & benefited by it.

CHAPTER 4

RESULTS AND DISCUSSION

Table 4.1 Previous v/s Proposed System

Queried String	**Expected String**	**Result of Modified Q-gram Algorithm**	**Result of Proposed ASM Algorithm**	**(Qgram) Execution Time(sec)**	**(P-ASM) Execution Time (sec)**
X-Men Origins	X-Men Origins:Wolverine	X-Men Origins: Wolverine	X-Men Origins: Wolverine	1.327	0.0490
Life of Pets	The Secret Life of Pets	The Secret Life of Pets	The Secret Life of Pets	1.221	0.0486
Most Wanted	A Most Wanted Man	A Most Wanted Man	A Most Wanted Man	1.229	0.0379
Twilight Saga Moon	The Twilight Saga: New Moon	The Twilight Saga: Eclipse	The Twilight Saga: New Moon	0.815	0.0446
Karate Kid	The Karate Kid	The Karate Kid	The Karate Kid	1.263	0.0414

CHAPTER 5

CONCLUSIONS AND FURTHER WORK

In contrast to the modified version of the q-gram technique used in this paper, Levenshtein approximation string matching algorithms are proposed in this paper, and the quick fuzz module of the Python programming language has been employed. Modified Levenshtein distances are used to determine how similar the values (of strings) are to one another. Examples of queries show how this might be advantageous. The normalized distance arrays are then used to compute probabilities of filtered tuples, which are subsequently interpreted as a tuple's confidence score. The result set is then ordered, with the most likely matches at the top, according to the decreasing probabilities.

I think the suggested system will be beneficial for users who only have a cursory understanding of the actual data already present in the database. By providing findings in less than a second, it also helps consumers save valuable time.

The proposed global q-gram distance may be used in different disciplines such as text mining, record linkage, data duplication, and DNA sequence alignment. My proposed system's execution time is significantly faster than the modified q-gram algorithm's execution time, and it also performs with more accuracy.

Patterns should match with more than one attribute, the generated set should also have those attributes, and the system can support datasets from multiple databases. Query evaluation techniques can be utilized for further query processing.

CHAPTER 6

REFERENCES

1. Amol S. Jumde And Ravindra B. Keskar, “Supporting Uncertain Predicates In DBMS Using Approximate String Matching And Probabilistic Databases”, IEEE, 0.1109/Access.2020.3021945.

2. Chayapathi A R, G Sunil Kumar, Manjunath Swamy Be, Thriveni J, Venugopal K.R., “Survey and Comparison Of String Matching Algorithms”, Turkish Journal of Computer & Mathematics Education ,Vol.12 No.12 (2021), 1471-1491.

3. Krishna Prakash, Kalyanathaya, Dr.D. Akila, Dr.G. Suseendren, “A Fuzzy Approach To Approximate String Matching For Text Retrieval In Nlp”, Journal of Computational Information Systems 15: 3 (2019) 26-32.

4. Saqib Iqbal Hakak, Amirrudin Kamsin, Palaiahnakote Shivakumara, Gulshan Amin Gilkar, Wazir Zada Khan And Muhammad Imran, “Exact String Matching Algorithms: Survey,issues, and Future Research Directions”, IEEE, 0.1109/Access.2019.2914071.