Supporting Uncertain Predicates in DBMS Using Approximate String Matching and Probabilistic Databases

AMOL S. JUMDE AND RAVINDRA B. KESKAR
Department of Computer Science and Engineering, Visvesvaraya National Institute of Technology, Nagpur 440010, India
Corresponding author: Amol S. Jumde (amol.jumde@students.vnit.ac.in)

ABSTRACT Current relational database systems are deterministic in nature and lack the support for approximate matching. The result of approximate matching would be the tuples annotated with the percentage of similarity but the existing relational database system can not process these similarity scores further. In this paper, we propose a system to support approximate matching in the DBMS field. We introduce a ‘≈’ (uncertain predicate operator) for approximate matching and devise a novel formula to calculate the similarity scores. Instead of returning an empty answer set in case of no match, our system gives ranked results thereby providing a glance at existing tuples closely matching with the queried literals. Two variants of the ‘≈’ operator are also introduced for numeric data: ‘≈+’ for higher-the-better and ‘≈−’ for lower-the-better cases. Efficient approximate string matching methods are proposed for matching string-type data whereas numeric closeness is used for other types of data (date, time, and number). We also provide results of our system taken over several sample queries that illustrate the significance of our system. All experiments are performed using the MySQL database, whereas the IMDb movie database and European Football database are used as sample datasets.

INDEX TERMS Approximate string matching, probabilistic databases, uncertain predicate.

I. INTRODUCTION
In traditional databases, select, from, and where are the fundamental clauses of any SQL query. General SQL query takes relations specified in the from clause as an input, removes tuples which do not satisfy the predicates in the where clause, and selects the attributes specified in the select clause. In the end, the query returns a filtered relation as an output.

A. PATTERN MATCHING IN DETERMINISTIC DATABASE
Pattern matching in a deterministic database is performed using the like operator. Patterns are described using two special characters ‘%’ and ‘_’. The ‘%’ matches any string i.e., any number of characters and ‘_’ matches a single character. SQL uses the like operator to express the pattern. Consider the following query in Fig. 1. This query returns the title of books containing ‘Computer’ as a substring.

![FIGURE 1. like operator in traditional database.](image)

select book_title
from books
where book_title like ‘%Computer%’;

The associate editor coordinating the review of this manuscript and approving it for publication was Haanqing Wang.
of support for approximate matching. In this paper, we try to address this issue.

### B. NEED OF UNCERTAIN MATCHES

Instead of returning an empty result-set due to above-mentioned reasons like lack of knowledge about the existing data, misspellings, typing mistakes, etc, it will be better to return the result-set ranked according to the degree of similarity. This would help the casual user who has very little knowledge about the stored data. From the approximate results that the user would get, she would come to know about the actual values in the database and she may re-query with the exact keywords that she intended.

Let’s assume there exists an uncertain predicate operator, denoted as ‘≈’, which is used for approximate matching just like ‘=’ operator for exact matching. If somehow approximate matching is performed, resulting rows would have a percentage of matching, where a higher percentage will indicate higher similarity. In such a case, the result of the given query could be a relation annotated with the percentage of matching. These matching scores can be treated as the probability scores or the confidence scores for the respective tuples. Probability scores of filtered tuples need to be stored along with the tuples. The deterministic database has no support for managing probabilities of tuples during query processing. In that direction, the probabilistic database has been proposed by Dalvi and Suciu [1] for managing the probability of a tuple during query processing. Moreover, the queries with uncertain predicate operator using a probabilistic database also have been proposed. For a given SQL query, each tuple in an input database is assigned a probability based on its match with constants in uncertain predicates. Thus the probability for each intermediate tuple is derived. Finally, the result-set is ranked according to its derived probability, indicating the most credible (i.e. the highest probable) answer first. For string matching purposes, Approximate String Matching (ASM) methods can be used which are discussed in the next section.

### C. EXAMPLE OF UNCERTAIN PREDICATE

Our general aim is that using Approximate String Matching (ASM) techniques and probabilistic databases, we should be able to execute the kind of query as shown in Fig. 2.

```sql
select book_title, book_author, price
from books
where book_title like ‘%Computer%’
and book_author ≈ ‘Christopher’
and price ≈ 400;
```

**FIGURE 2. Uncertain predicate for approximate matching.**

As mentioned, the ‘≈’ operator is the proposed uncertain predicate operator. The expected result is depicted in Table 1 which shows the result-set arranged in the decreasing order of the tuples’ probability, topmost tuple being the most similar.

### D. OUTLINE AND CONTRIBUTIONS

As per our knowledge, database systems still do not have support for uncertain predicates and it is incompetent to perform approximate string matching. In this work, we extended the idea of queries with uncertain predicates introduced in [1]. In this paper, we try to provide the design of a complete system supporting the uncertain predicates in the database, which, we believe, will make database fields more informative. This will also enable the capabilities of information retrieval to be applied to the database fields. With a probabilistic database in hand, support for uncertain predicates is attainable.

The rest of the paper is organized as follows. We discuss the related work in Section II. In Section III, we take a brief overview of the concepts of the probabilistic database and its use for modeling the uncertainty. We also review various existing ASM methods and propose the best method of interest. Later, we cover the main contributions of the paper:

- We propose the design and implementation of a system to support uncertain predicates along with the regular “certain” predicates.
- We identified the deficiency of basic q-gram distance in our scenario and proposed the global q-gram distance. We provide the complexity and overhead analysis of the proposed formula (Section IV).
- We present the queries to calculate the above-mentioned distance by utilizing the in-built features provided by the DBMS (Section IV).
- We also introduce two variants (≈+ and ≈−) of uncertain predicate operator (≈) for the numeric type of data (Section IV).
- We formulate the procedures to implement our proposed method. We also describe details of the sample databases that we have used for the testing (Section V).
- We demonstrate the results of several queries on sample databases (Section VI).

Finally, we conclude in Section VII with possible future directions.

### II. RELATED WORK

SQL query with uncertain predicate is first proposed by Dalvi and Suciu [1]. But, the main focus of their work was on query evaluation in probabilistic databases. They have given intensional and extensional query evaluation to derive the probability of the filtered tuples. But, to the best of our knowledge, a full-fledged system supporting uncertain predicate is still missing. Probabilistic databases have been widely...
explored for storing uncertain data, instead of just replacing uncertain fields with nulls [2]. Hassanzadeh and Miller [3] have used probabilistic databases in order to remove duplicate records from the base dirty relation. They find out the similarity between the records and use clustering algorithms to identify the set of records that are probable duplicates. Kanagal et al. [4] proposed the sensitivity analysis to find the most influential input tuples for the query which predominantly affects the probability of the answer. Qin and Yu [5] proposed the dynamic programming algorithm to perform sensitive analysis for Inequality Query (IQ) efficiently. Sensitivity analysis can be utilized for ranking the query results [6].

Hakak et al. [7] presented a comprehensive survey on exact string matching algorithms that are commonly used for pattern searching. Unlike exact string matching, Approximate String Matching(ASM) methods allow errors in matching, which makes them useful for applications like spell checker, autocompletion, etc [8], [9]. ASM methods are also used to extract features for machine learning and to calculate intra-comment similarity to tackle the issue of cybercrimes which makes them useful for applications like spell checker, String Matching(ASM) methods allow errors in matching, exact string matching algorithms that are commonly used for

proposed the dynamic programming algorithm to perform sensitive analysis for Inequality Query (IQ) efficiently. Sensitivity analysis can be utilized for ranking the query results [6].

Hakak et al. [7] presented a comprehensive survey on exact string matching algorithms that are commonly used for pattern searching. Unlike exact string matching, Approximate String Matching(ASM) methods allow errors in matching, which makes them useful for applications like spell checker, autocompletion, etc [8], [9]. ASM methods are also used to extract features for machine learning and to calculate intra-comment similarity to tackle the issue of cybercrimes which makes them useful for applications like spell checker, String Matching(ASM) methods allow errors in matching, exact string matching algorithms that are commonly used for

III. PRELIMINARIES
In this section, we briefly take an overview of the probabilistic database and ASM methods which are the building blocks of our system.

A. UNCERTAINTY AND PROBABILISTIC DATABASES
To deal with uncertain data and probabilistic query processing, probabilistic databases are proposed [33]. Nowadays, many applications have to deal with uncertain data, such as object recognition, sensor networks, moving object monitoring, data extraction, and integration [21]. Uncertainty may arise during the data collection, data transmission, or data processing stages.

With traditional deterministic databases, uncertainty cannot be modeled. The use of nulls is proposed to handle the uncertainty by Codd [34]. But, there can be an information loss in using the nulls in place of doubtful values. For example, consider a classifier that classifies the objects. It could identify that object X belongs to class A with 0.6 probability and class B with 0.4 probability. Due to this uncertainty, if the class of X is recorded as null, the information that we had earlier is lost. Such type of uncertainty can be best modeled and processed using probabilistic databases [35]–[39]. It stores uncertainty in terms of a probability for that uncertain tuple or attribute. The probability of a tuple or an attribute is a measure of confidence that the system has on that reading.

Formally, probabilistic databases can be defined as follows:

Definition 3.1 (Probabilistic Database and Possible World [23]): Consider a schema with relations \( R_1, \ldots, R_n \). A probabilistic database is a finite set of structures \( W = \{ W^1, W^2, \ldots, W^n \} \), where each \( W^i = \{ R_1^i, \ldots, R_n^i, p^i \} \), is called a possible world and its probability of existence is \( p^i \).

Further, \( \sum_{1 \leq i \leq n} p^i = 1 \). Every relation \( R_i^j \) in world \( W^i \) contains one of the possible subsets of tuples from relation \( R_i \). If relation \( R_i \) contains only the certain tuples i.e., tuples with probability 1, it is called as certain relation, and \( R_i^1 = R_i^2 = \cdots = R_i^n = R_i \).

Let us consider the following example of probabilistic database.

1) EXAMPLE OF PROBABILISTIC DATABASE
Consider, an institute has installed an automated system for maintaining students’ attendance. The system recognizes a student using facial recognition with some error. The percentage of the match can be treated as a probability of that tuple. One particular instance of such a table is given in Table 2. The database with such uncertain relations is called a probabilistic database.

2) POSSIBLE WORLD SEMANTICS
As seen in the previous example, every tuple is assigned with some probability of existence. For instance, tuple \( t_1 \) is true with 0.9 probability and at the same time, it is false with
of all possible q-grams, and into three major categories, namely
have a distance of 0. ASM distance metrics are classified
measured in terms of distance. Exactly matched strings will
it is useful in cases of undesired corruption or inadvertent
the error between them. As ASM allows errors,
characters mismatches between close characters are more likely due
to the typing errors like martha-marhta or jonathon-jonathn.
Lastly, heuristic based distances have a particular application
in mind, but do not have a solid mathematical backbone.
For example, Jaro distance [45] was specifically designed to
pair the records between the census data and the independent
Post Enumeration Survey (PES) conducted by the U.S. Cen-
sus Bureau in 1985. The intuition behind the method is that
the types of errors like transpositions of characters or charac-
ter mismatches between close characters are more likely due
to the typing errors like martha-marhta or jonathon-jonathn.
These are legitimate variations and should have lesser dis-
tances. It is evident from the following examples:
\[ d_{jaro}(‘martha’, ‘marhta’) = 0.05555556 \]
\[ d_{jaro}(‘martha’, ‘arham’) = 0.1111111 \]
\[ d_{jaro}(‘martha’, ‘megan’) = 0.4222222 \]
Though there is a single transposition of characters in the
first two examples, the Jaro distance of the first example is
less as it may be more likely due to the typing error.
After this brief overview of uncertainty and probabilistic
databases, as well as Approximate String Matching (ASM),
our proposed methodology based on these two concepts is
described in the next section.

IV. METHODOLOGY
In order to find a similarity between the values in terms of dis-
tance, we introduce ‘≈’ operator for the uncertain predicate.
To implement such an operator, for string type data, we can
use the distance between the strings which is also a notion of
dissimilarity between them. For numerals, the distance could
be the indicator of numeric closeness. Fig. 3 shows the basic
modules of the proposed system which are preprocessing,

| TABLE 2. Example of probabilistic database. |
|-------------------------------------------|
| Record_No | Name           | Time     | Location | conf |
|-----------|----------------|----------|----------|------|
| t1        | Rakesh Kumar   | 15-09-02 | CSB-101  | 0.9  |
| t2        | Ankur Bhagat   | 15-11-02 | CSB-101  | 0.8  |
| t3        | Rekshi Trsag   | 12-12-21 | CSB-102  | 0.35 |
| t4        | Neeta Singla   | 15-15-31 | CSB-101  | 1.0  |
| ...       | ...            | ...      | ...      | ...  |

0.1 probability. Thus, in one instance, some of the tuples will
remain true and the rest will be false. This forms one possible
instance of the above database. In this way, the probabilistic
database can exist in multiple possible instances. In one of
the possible instances, all tuples will remain true. Similarly,
in one particular instance, a table may not have any tuple (this
can happen only if there is no certain tuple in a relation). Such
multiple instances are called possible worlds of a probabilis-
tic database.

A traditional relational database has only one state, con-
taining all the present tuples. In contrast, a probabilistic
database can be in one of several states. It is a set of multiple
possible certain databases each of which has some chance to
appear in the real world. The probability of a possible
world is determined by the probabilities of its tuples. For
example, probability of possible world pw containing three
tuples (say t1, t4, and t5) out of five tuples is,
\[ P_{pw}(t_1, t_4, t_5) = p_1 \ast (1 - p_2) \ast (1 - p_3) \ast p_4 \ast p_5, \]
where \( p_1, p_4, p_5 \) are the probabilities of presence of tuples
t1, t4, and t5, respectively, and (1 – p2) and (1 – p3)
represent the probabilities of the absence of tuples t2 and t3.

B. APPROXIMATE STRING MATCHING
Approximate String Matching (ASM) is a string matching
problem where two strings are matched approximately rather
than exactly. ASM aims to find the closest match for a query
string. The uncommon part between the two strings is con-
sidered as the error between them. As ASM allows errors,
it is useful in cases of undesired corruption or inadvertent
typing errors. Most often, the approximate string match is
measured in terms of distance. Exactly matched strings will
have a distance of 0. ASM distance metrics are classified
into three major categories, namely edit based, q-grams based
(sometimes called as n-grams), and heuristic based [9], [14].

The edit based metrics are measured in terms of the number of
edit operations required to make two strings equal. These edit operations include addition, deletion, and substitution of a character, or
transposition of characters. Based on the allowed operations, edit based distances are
categorized as Hamming distance [40], Longest Common
Subsequence (LCS) distance [41], Levenshtein (LV) dis-
tance [42], Optimal String Alignment (OSA) distance, and
damerau-levenshtein (DL) distance [43].

The q-gram based distances aim to find common q-grams
between two matched strings. The q-grams of string s is a set
of all possible q length substrings of s. The maximum number of
distinct q-grams for string s is \(|s| - q + 1\). For example, if
q = 2 and s = ‘VNIT’ then the q-grams are {'VN', 'NI', 'IT'}. Jaccard
distance, q-gram distance [44], and Cosine
distance are some examples of q-gram based distances.

Definition 3.2 (q-gram Distance): The q-gram distance is
the number of q-grams that are not common between the two
strings and can be calculated as:
\[ d_{qgram}(s, t; q) = \|v(s; q) - v(t; q)\|_1 = \sum_{i=1}^{||\Omega||} |v_i(s; q) - v_i(t; q)| \]
where, \( \Omega \) represents the finite set of alphabets.\(^1\) The Kleene
closure set (\( \Omega^* \)) is an infinite set of all possible strings over \( \Omega \).
The set \( \Omega^q \) represents all the strings of length q from \( \Omega^* \). The
vector \( v(s; q) \) is a q-gram profile of string s of dimension \( ||\Omega||^q \)
whose coefficients represent the number of occurrences of
every possible q-grams of length q. Let, a q-gram x \( \in \Omega^q \),
has an index i in the vector \( v(s; q) \) then the \( v_i(s; q) \) represents
the total occurrences of x in s. Thus, the q-gram distance is
equal to the L1-norm of vectors of two strings [44]. It varies
between 0 to \(|s| + |t| - 2q + 2\). For example, the q-gram distance between ‘Tu’ and ‘Thursday’ is 7 for q = 2 as
‘Tu’, ‘ue’, ‘es’, ‘Th’, ‘hu’, ‘ur’, and ‘rs’ are the q-grams that
are not common between them.

\(^1\)In the literature, the \( \Sigma \) symbol is generally used to represent the finite set of
alphabets. But here we used the \( \Omega \) symbol to avoid the confusion between
the set of alphabets and the summation operation.
distance calculation, distance normalization, and probability calculation. When the parser encounters ‘≈’ symbol in the where clause of the query, it performs approximate matching on the columns involved in uncertain predicates. A query may have other predicates along with the uncertain predicate, as present in our previous books example in Section I (again depicted in Fig. 4(a)). An instance of the books relation is shown in Fig. 4(b). All predicates, except uncertain predicates, are applied first and then uncertain predicates are applied on those filtered result-set. These steps are the preprocessing steps of our system. Fig. 4(c) illustrates the filtered result-set. Distance Calculation Module calculates the distance between the queried literal and each of the field values in the corresponding column to get the distance array (Fig. 4(d)). Distance array is normalized in the range [0, 1] by the Distance Normalization Module (Fig. 4(e)). The above steps are repeated for all the uncertain predicates. Probability Calculation Module combines the probabilities obtained from the uncertain predicates to calculate the final probability of filtered tuples (Fig. 4(f)) and finally, we get a probabilistic database as an output (Fig. 4(g)).

Procedures to find distances differ based on the data type of a column. Most of the standard databases support various built-in data types, such as number for integer type and real type data, char for a character, varchar for a string. Along with these basic types, databases also support types for a date (date) and time (time).

Firstly, we focus on string-type data. Out of all the ASM techniques discussed in the previous section, we believe q-gram distance (with required modifications) is best suited for our purpose. Edit based distances do not fare well in case of swapped words cases. For example, ‘John Watson’ may get recorded as ‘Watson, John’. The edit based distance between these two strings is 10 (for OSA) out of a possible range of [0, 12], which is significant. In the case of q-gram distance, it would be just 5 (for q = 2) out of a possible range of [0, 21]. We have neglected heuristic-based methods as they are very much application-specific.

In Section IV-A we describe a naive solution to find the distance between two strings. In Section IV-B we propose a more accurate distance formula that overcomes the drawbacks of the naive solution.

A. NAIVE SOLUTION TO CALCULATE THE DISTANCE BETWEEN TWO STRINGS

As noted earlier, q-gram distance needs $O(|\Omega|^q)$ storage. To avoid such high space complexity, we use the following technique to calculate the q-gram distance between the two strings (s and t) in RDBMS. Here one can try to utilize in-built features already available with the RDBMS like efficient joins methods, clustered index, etc. Auxiliary tables, $A_1(q\text{gram})$ and $A_2(q\text{gram})$, with qgram as the only attribute,
are created on-the-fly respectively for the two strings and deleted upon use. The example given in Fig. 5 depicts auxiliary tables $A_1$ and $A_2$ tables storing q-grams of $s = \text{`abdhg'}$ and $t = \text{`abcdehgh'}$ strings. Query Q1 in Fig. 5 is the SQL query to find the q-gram distance between the strings $s$ and $t$.

### B. PROPOSED SOLUTION TO CALCULATE THE DISTANCE BETWEEN TWO STRINGS

The problem in using q-gram distance in its purest form is that the user may query a part of the original string present in the table. For example, she may query ‘Harry Potter’ instead of ‘Harry Potter and the Order of the Phoenix’ while searching for the movie titles. Such extra uncommon parts of one of the strings increase the distance. To overcome this problem, we introduce the concepts of Local q-gram distance and Global q-gram distance. We define them as follows:

**Definition 4.1 (Local q-Gram Distance):** The local q-gram distance is the q-gram distance between the smaller length string and one of the substrings of the same length of the larger string.

The larger string (say $y$), among the two strings to be matched, is divided into the set of substrings ($y_i$)'s) where length of each substring is equal to the length of smaller-length string (say $x$). Let, $l_x$, $l_y$ be the length of strings $x$ and $y$, respectively. Then, $n = l_y - l_x + 1$, which will be the number of possible substrings ($y_i$)'s of length $l_y$ of a string $y$. The local q-gram distances ($d_i$) between $x$ and each of $y_i$'s are calculated using query Q1. The proposed net distance between the strings $x$ and $y$ is derived from their local q-gram distances to be termed as ‘Global q-gram distance’.

**Definition 4.2 (Global q-Gram Distance):** The global q-gram distance between strings $x$ and $y$ is proposed to be calculated as:

$$d(x, y) = \min(d_1, d_2, \ldots, d_n) + \frac{l_y - l_x}{l_x} + \frac{\sum_{i=1}^{n} d_i}{\text{maxdist}(x, y)}$$

Here $d_1, d_2, \ldots, d_n$ are the local q-gram distances between two strings calculated as mentioned above. The next two terms are the fine-tuning terms that help to calculate a more accurate distance. $\text{maxdist}(x, y)$ is the sum of maximum distance possible between the $x$ and each of $y_i$’s and therefore $\text{maxdist}(x, y) = 2n(l_x - q + 1)$.

Now we explain the significance of each term in the global q-gram distance formula proposed above. The overall aim is to reduce the increased distance between two strings due to the missing part of the query string, the lacuna in the naive solution mentioned earlier. We first note that the first string may be contained anywhere in the second string. Hence, to minimize the distance between two strings, minimum of local q-gram distances is considered ($\min(d_1, d_2, \ldots, d_n)$) as the first term in the formula to calculate the global q-gram distance. But, it can be quickly seen that this is not enough. Even if $d(‘aaa’,’aaa’) = 0$ and $d(‘aaaa’, ‘aaaaa’) = 0$ (for $q = 2$), still, the first string should get precedence over the second in terms of similarity. Hence, there is a need for a penalty for the difference in the relative lengths of the strings, which is computed by $(l_y - l_x)/l_x$ as the second term in the formula. Hence, higher the length differences (amongst the matched strings), higher will be the penalty. Now a penalty with a fractional value (0.667) gets added in the calculation of $d(‘aa’, ‘aaaa’) as against $d(‘aaa’, ‘aaa’) which remains at 0. Even then, $d(‘aa’, ‘aaaaa$) is 0.667 and the $d(‘aa’, ‘baaad$) is also 0.667. Here we believe ‘$aa’ is more closer to ‘aaaaa’ than ‘baaad’ as ‘aa’ gets matched multiple times in ‘aaaaa’ than in ‘baaad’. To factor in this observation, the third term ($\sum_{i=1}^{n} d_i/\text{maxdist}(x, y)$) in the formula is added in the calculation of global q-gram distance. It denotes the ratio of summation of actual distances of the substrings ($\sum_{i=1}^{n} d_i$) to the maximum possible distance between the strings $x$ and $y$ ($\text{maxdist}(x, y)$). This ratio measures the amount of cumulative dissimilarity between the two strings measured relatively against the maximum distance. Distance is maximum when there is no common character or common q-gram between the strings $x$ and $y$. In the example discussed above, the sum of distances with respect to ‘baaad’ ($0.667 + 0.333 = 1$) is more as against ‘aaaaa’ ($0 + 0.667 + 0 = 0.667$) due to this third term, and hence ‘aaaaa’ will be considered as a better match of ‘aaa’ compared to ‘baaad’, as required.

Table 3 depicts some sample strings and their distances when the formula is applied in a part-wise manner. The distances calculated using the global distance formula in the table justifies the significance of each term of the formula.

### Table 3. Result of proposed formula on some sample strings.

| String 1 | String 2 | Considering only the 1st term of the formula | Considering 1st & 2nd term of the formula | Considering 1st, 2nd & 3rd term of the formula |
|----------|----------|---------------------------------------------|-------------------------------------------|---------------------------------------------|
| aaaa     | aaaa     | 0                                           | 0                                         | 0                                           |
| aaaa     | aaaaaa   | 0.5                                         | 0.5                                       | 0.5                                         |
| aaaa     | baaacc   | 0.5                                         | 0.7222                                    | 0.7222                                      |
| aaaa     | aaabc    | 0.5                                         | 0.8333                                    | 0.8333                                      |
| aaaa     | aaaa aaaa| 2.75                                        | 3.25                                      | 3.25                                        |
| aaaa     | abcdle   | 6.25                                        | 7.25                                      | 7.25                                        |

Let AUX be an auxiliary table used to store local q-gram distances. To calculate the global q-gram distance, we execute query Q2 in Fig. 6 on the AUX table. Before executing Q2,
we need to set constants like \( q \), \( l_1 \), \( l_2 \) and \( \text{maxdist} \) using MySQL commands as shown in Fig. 6.

For other data types like numeric or date, a simple absolute difference of the values would be the distance between them. For example, a unit of distance between two date values is the number of days separating them, like \( d(1973-12-30', '1973-10-15') = 76 \) days. In this way, approximate matching is performed for all the columns involved in uncertain predicates and corresponding distance arrays are obtained.

### C. Complexity Analysis

In case of a naive solution, the q-gram distance between the strings \( x \) and \( y \) can be computed in \( O(|x| \cdot |y|) \) time and the memory requirement is \( O(Q(y; q)) \) storage, where \( Q(y; q) \) denotes a set of q-grams existing in string \( y \), if \( l_2 > l_1 \).

Overhead due the creation of substrings in global q-gram distance will increase the time complexity to \( O(|x|^2 \cdot |y|) \) but, memory requirement will reduce to \( O(|Q(x; q)|) \). This overhead comes with improved accuracy of the system.

### D. Variants of ‘\( \approx \)’ Operator for Numeric Data

Along with ‘\( \approx \)’ operator, we introduce ‘\( \approx+ \)’ and ‘\( \approx- \)’ operators. The operator ‘\( \approx+ \)’ indicates the higher-the-better case. Sometimes the higher distance between the values is beneficial. For example, if we are searching for books with discount nearly equal to 20%. It is better to show books with higher discount at the top. Similarly, ‘\( \approx- \)’ indicates the lower-the-better case. In our book example, we were searching for books with a price nearly equal to 400. Here, it is better to show books with lower prices at the top. For ‘\( \approx \)’ operator we take the absolute difference between the values. But in the case of ‘\( \approx+ \)’, we need to take the specific difference (constant - column_value) and in the case of ‘\( \approx- \)’, the (column_value - constant).

### E. Normalization of Distance Array

All distance arrays may differ in the range of values. For the uniformity, distance arrays are normalized in the range \([0, 1]\) using the following formula to obtain the probability array:

\[
p[i] = 1 - \frac{\text{dist}[i] - \text{min}}{\text{max} - \text{min}}
\]

The probability of a match is equal to one minus the normalized score, because \( d = 0 \) indicates the exact match, whereas, in probability notion, \( p = 1 \) indicates the exact match. When we get the distance array for data type other than string, we assume the highest element in an array to be the maximum calculated distance (\( \text{max} \)). In this way, the probability array for each of the columns involved in uncertain predicates is calculated.

### F. Final Probability Calculation

A query may have multiple uncertain predicates combined with and and or operators. Moreover, each uncertain predicate is an independent probabilistic event, which means the occurrence of one predicate will not affect the probability of other predicates. Therefore, the conditional probability of a predicate \( A \), in the presence of predicate \( B \), is equal to the probability of the predicate \( A \) i.e., \( p(A|B) = p(A) \). Also, the precedence of and operator is higher than the or operator. Therefore, the probability of ‘\( n \)’ predicates combined with and operator is first calculated using the following formula:

\[
p = (p_1 \cap p_2 \cap \cdots \cap p_n) = p_1 \times p_2 \times \cdots \times p_n
\]

The probability for ‘\( m \)’ predicates combined by or operator is calculated by using the principle of Inclusion-Exclusion as given below:

\[
p = (p_1 \cup p_2 \cup \cdots \cup p_m) = \sum_{1 \leq i \leq m} p_i - \sum_{1 \leq i < j \leq m} (p_{i1} \cap p_{j2}) + \sum_{1 \leq i < j < k \leq m} (p_{i1} \cap p_{j2} \cap p_{k3}) - \cdots + (-1)^{m-1} (p_1 \cap \cdots \cap p_m)
\]

In case of a complex expression containing multiple operators like and and or, operations are performed based on the operator precedence. Parenthesis can be used to override the actual precedence of operators. For nested parenthesis, innermost parenthesis is evaluated first.

For example, consider a query with four uncertain predicates combined with one and and two or operators applied sequentially without explicitly resolving the precedence. After performing approximate matching, let \( 0.7, 0.6, 0.4, \) and \( 0.5 \) be the probabilities of the corresponding four columns for the resultant tuple \( t_1 \). Then, the final probability of tuple \( t_1 \) is \( p(t_1) = 0.7 \cap 0.6 \cup 0.4 \cup 0.5 = (0.7 \cap 0.6) \cup 0.4 \cup 0.5 = 0.42 \cup 0.4 \cup 0.5 = 0.826 \).

In the next section, we summarize the entire methodology in the form of set of procedures. We discuss the implementation details and the datasets used for the experimentation, as well.

### V. Implementation, Datasets and Experimentation

To support uncertain predicate operator ‘\( \approx \)’, the query parser should recognize newly introduced ‘\( \approx \)’ operator. For experimentation, we parsed the input query in the C code itself to check whether the given query has ‘\( \approx \)’ operator. If the input query has a ‘\( \approx \)’ operator, we first filter out tuples by executing a query with certain predicates. Afterward, we apply uncertain predicates on those filtered result-set by invoking procedure \( \text{UncertainPredicate} \) given in Procedure 1.
In Procedure 2, the value of 'q' is obtained using getValueOfQ() function. Using calGlobalQgramDist() function, we calculate the global q-gram distance. It issues query Q2 to the database which itself makes use of query Q1.

In Procedure 3, calDaysDifference() function is used to calculate the distance in days, for a column of date type. Distance for the time data type is calculated using calMinsDifference() in the units of minutes. For the numeric data type, distance is the difference of values and it is calculated using calDifference() function.

For experimentation, we have used the MySQL database. MySQL C API library is used to communicate with MySQL server. The libmysqlclient is a client version of the library that C applications use to communicate. This C API offers several data structures for database connection, for storing the result of a query, like, MYSQL, MYSQL_RES, MYSQL_ROW. It also provides various C API functions like mysql_init() for initializing MYSQL structure, mysql_real_connect() for connecting to the MySQL server, mysql_query() for executing the query and many more [46].

### A. DATABASES USED FOR TESTING

IMDb movie [47] and European football [48] databases were used for testing. IMDb movie database has information about 5000+ movies. IMDb database has only one relation named ‘Movies’. Original relation had too many attributes related to the movies. In our version of ‘Movies’ relation, we inserted only 11 important attributes amongst them. These are Movie_Title, Director_Name, Duration, Actor_1_Name, Actor_2_Name, Actor_3_Name, Gross, Genres, Budget, Title_Year, and IMDb_Score.

Football database contains information of 11000+ players and their attributes, 25000+ matches, and 11 leagues. ‘Player’ relation stores personal information about the player and his attributes values are stored in ‘Player_Attributes’ relation. Attributes in ‘Player’ relation are id, player_api_id, player_name, player_fifa_api_id, birthday, height, and weight. ‘Player_Attributes’ relation has 42 columns that cover all attributes of a football player. player_api_id and player_fifa_api_id attributes are the foreign keys referring to the identical attributes in the ‘Player’ relations. Some fields in these databases were missing or incorrect. Our modified versions of these databases are available at [49].

### VI. RESULTS AND DISCUSSION

In this section, we present the results of different experiments. As part of the experimentation, we compared the performances of the naive approach with the proposed solution.
We also compared the results of the proposed global q-gram distance with the classic edit-based distance. We analyzed the effect of the value of q on accuracy. We have also presented the results of additionally proposed operators ($\approx+$ and $\approx-$).

\[
\text{select Movie_Title} \\
\text{from Movies} \\
\text{where Movie_Title } \approx \text{‘X-Men Origins’};
\]

\textbf{FIGURE 7.} Sample query on the IMDb database.

**A. NAIVE SOLUTION VS. PROPOSED SOLUTION**

Here we compare the performance of the naive solution as mentioned in Section IV-A with the proposed solution in Section IV-B. One sample query on the IMDb movie database is shown in Fig. 7. Table 4 shows their results on some random strings. Queried strings and the expected strings are shown in the first two columns of the table. The third and fourth columns of the table represent the result of a naive approach and the proposed technique, respectively. A naive method fails when there is a significant mismatch between the lengths of two strings.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
Queried Strings & Expected Strings & Result of naive method & Result of proposed method \\
\hline
ex1 X-Men Origins & X-men Origins: Wolverine & I Origins & X-men Origins: Wolverine \\
\hline
ex2 Life of Pets & The Secret Life of Pets & Life of Pets & The Secret Life of Pets \\
\hline
ex3 Most Wanted Man & A Most Wanted Man & Wanted & A Most Wanted Man \\
\hline
ex4 Twilight Saga: New Moon & Moonlight Mile & Twilight \\
\hline
ex5 Karate Kid & The Karate Kid & The Karate Kid \\
\hline
\end{tabular}
\caption{Results of naive method and the proposed method (q = 2).}
\end{table}

In the proposed method, there is an overhead of matching individual substrings which resulted in the change of the complexity from $O(|x| \cdot |y|)$ to $O(|x|^2 \cdot |y|)$. The actual increase in execution times is depicted in Fig. 8. The Movies relation contains 5043 tuples. Each of the execution times shown in Fig. 8 is the average of 10 experiments. An average of execution times of all the experiments ($5 \times 10 = 50$ experiments) shows a 46.3% increase in the execution time. But, this trade-off comes with improved accuracy as shown in Table 4 and Fig. 8.

**B. EDIT BASED DISTANCE VS. GLOBAL Q-GRAM DISTANCE**

As mentioned earlier, we noted that the edit-based distance would fail to identify similarities between the swapped-words strings (i.e. ‘John Watson’ vs ‘Watson, John’). Here, we illustrate it further using a few examples. In query 1 (see Fig. 9), we intend to get a film from the Harry Potter film series, which has its IMDb score is near to 7.0 but let’s say we (wrongly) suppose, is directed by ‘David Bates’. In reality, some parts of this film series are directed by David Yates (and not Bates). Tables 5-7 show the correct part of the movies at the top of the tables. We tried OSA and LCS methods from the edit-based category. We obtained the intended results for all the methods, as shown in the tables mentioned above.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
Movie Title & Director Name & IMDb Score \\
\hline
Harry Potter and the Half-Blood Prince & David Yates & 7.5 & 0.720278837 \\
Harry Potter and the Order of the Phoenix & David Yates & 7.5 & 0.716742963 \\
Pury & David Aver & 7.6 & 0.432680666 \\
Home Run & David Boyd & 6 & 0.37913609 \\
Dune & David Lynch & 6.6 & 0.364126543 \\
\hline
\end{tabular}
\caption{Result of Query 1 using the global q-gram method (q = 2).}
\end{table}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
Movie Title & Director Name & IMDb Score \\
\hline
Harry Potter and the Half-Blood Prince & David Yates & 7.5 & 0.720278837 \\
Harry Potter and the Order of the Phoenix & David Yates & 7.5 & 0.716742963 \\
Redbelt & David Mamet & 6.8 & 0.417505585 \\
Astro Boy & David Bowers & 6.3 & 0.394052863 \\
Harsh Times & David Aver & 7 & 0.37913609 \\
\hline
\end{tabular}
\caption{Result of Query 1 using the OSA method.}
\end{table}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
Movie Title & Director Name & IMDb Score \\
\hline
Harry Potter and the Half-Blood Prince & David Yates & 7.5 & 0.720278837 \\
Harry Potter and the Order of the Phoenix & David Yates & 7.5 & 0.716742963 \\
Hard Candy & David Slade & 7.1 & 0.46103402 \\
Pury & David Aver & 7.6 & 0.45884952 \\
Redbelt & David Mamet & 6.8 & 0.4306562 \\
\hline
\end{tabular}
\caption{Result of Query 1 using the LCS method.}
\end{table}
to ‘Hunters of Ghost’ in which an actor named ‘Steve’ has acted. But the real name of the movie is ‘Ghost Hunters’ and the complete name of the actor is ‘Steve Gonsalves’.

IMDb database has three columns for storing the names of three actors and we are not sure which column of actor names contains ‘Steve’. Notice that we have used parenthesis to override the default precedence of the operators and to associate these three conditions. In the q-gram distance, a set of all possible q-grams is formed. So, inherently, the sequence of the sub-strings does not affect the distance. Thus, it is expected that the q-gram method would give the intended result at the top. On the other hand, OSA and LCS perform sequential matching. Hence, even if two strings have exact same substrings and if one has them swapped, distance is not zero. For example,

\[
d_{qgram}(\text{‘Ghost Hunters’}, \text{‘Hunters of Ghost’}, q = 2) = 7 \text{ out of a possible range of } [0-27],
\]

\[
d_{osa}(\text{‘Ghost Hunters’}, \text{‘Hunters of Ghost’}) = 14 \text{ out of a possible range of } [0-16] \text{ [Operations: SSSDDMMSSSSSIIMDD],}
\]

\[
d_{lcs}(\text{‘Ghost Hunters’}, \text{‘Hunters of Ghost’}) = 15 \text{ [Operations: DDDDDDMMMMMMIMSSSSSIIMIII],}
\]

where ‘S’ stands for substitution, ‘D’ for deletion, ‘I’ for Insertion, and ‘M’ for match of the character. Results of Query 2 on the IMDb database are as shown in Tables 8-10.

**TABLE 8. Result of Query 2 using the global q-gram method (q = 2).**

| Movie_Title | Actor_1_Name | Actor_2_Name | Actor_3_Name | prob  |
|-------------|--------------|--------------|--------------|-------|
| Ghost Hunters | Amy Bruni    | Steve Gonsalves | Jason Hawes   | 0.6002946823 |
| Khumba      | Liam Neeson  | Steve Bucersu | Loretta Devine | 0.364523379 |
| Minions     | Steve Carell | Jon Hamm     | Steve Coogan  | 0.356277755 |
| Ghost       | Demi Moore   | Tony Goldwyn | Phil Leeds    | 0.535126168 |
| Ghost World | Scarlett Johansson | Steve Bucerus | T.J. Thyne   | 0.532182172 |

**C. ADDITIONAL OPERATORS**

To illustrate the utility of the proposed uncertain operator ‘≈’, two additional operators (‘≈+’ and ‘≈−’) are also proposed as an extension. We conducted several experiments to test the practicability of the proposed operators. We executed the same query first with ‘≈’ and then using ‘≈+’/‘≈−’ operators and examined the results. In query 3 (see Fig. 11), we tried to utilize our system as a predictor. We want to find the best striker from the Football database. There are several attributes that represent the player’s skills and his/her performance. Key attributes of any striker are finishing, dribbling, acceleration, composure, and pace. To get the name of the players, we joined ‘Player’ and ‘Player_Attributes’ relations on the ‘player_api_id’ attribute. Table 11 shows the result of Query 3 which has today’s leading strikers at the top.

**TABLE 9. Result of Query 2 using the OSA method.**

| Movie_Title | Actor_1_Name | Actor_2_Name | Actor_3_Name | prob  |
|-------------|--------------|--------------|--------------|-------|
| Dinner      | Steve Gutenberg | Daniel Stern | Ellen Barkin | 0.583222671 |
| Yeni        | Miriam Margolyes | Amy Irving   | Steven Hill  | 0.579948426 |
| Ghost       | Demi Moore   | Tony Goldwyn | Phil Leeds   | 0.57555179 |
| Good        | Viggo Mortensen | Jodie Whitaker | Steven Mackintosh | 0.570624901 |
| Minions     | Steve Carell | Jon Hamm     | Steve Coogan  | 0.559427524 |

**TABLE 10. Result of Query 2 using the LCS method.**

| Movie_Title | Actor_1_Name | Actor_2_Name | Actor_3_Name | prob  |
|-------------|--------------|--------------|--------------|-------|
| Dinner      | Steve Gutenberg | Daniel Stern | Ellen Barkin | 0.715666305 |
| Good        | Viggo Mortensen | Jodie Whitaker | Steven Mackintosh | 0.685238459 |
| Yeni        | Miriam Margolyes | Amy Irving   | Steven Hill  | 0.642897913 |
| Wings       | Steven Weber  | Tim Daly     | Amy Yazbeck  | 0.637446788 |
| Ghost       | Demi Moore   | Tony Goldwyn | Phil Leeds   | 0.602204576 |

**FIGURE 11. Query 3 on the Football database.**

In query 4 (see Fig. 12), we demonstrated the use of the ‘≈+’ operator. Players with better finishing, acceleration and dribbling scores should be ranked first. Table 12 shows the result of query 4, where we can observe that Messi now has jumped to the top of the table.

**TABLE 11. Result of Query 3.**

| player_api_id | player_name | finishing | acceleration | dribbling | prob  |
|---------------|-------------|-----------|--------------|-----------|-------|
| 37412         | Sergio Agiero | 90         | 92           | 89        | 0.954149049 |
| 30893         | Cristiano Ronaldo | 95         | 91           | 93        | 0.953682394 |
| 19533         | Neymar      | 88         | 91           | 94        | 0.931503733 |
| 30981         | Lionel Messi | 93         | 95           | 96        | 0.971275953 |
| 328912        | Paulo Dybala | 85         | 90           | 89        | 0.897924639 |
| 30834         | Arjen Robben | 85         | 90           | 93        | 0.897667059 |
| 40636         | Luis Suarez  | 90         | 88           | 88        | 0.894923597 |

In query 5 (see Fig. 13), we tried to find out the highest-grossing movies. Therefore, the gross amount should be higher than the constant in the condition as much as possible, and the budget should be as low as possible. Values of gross and budget specified in the conditions are the average values of the gross and budgets of all the movies. Table 13 shows the results of Query 5 and indicates Avatar as the highest-grossing movie of that period.
result at the top. By providing support for uncertain predicates in this work, we are, in a way, trying to achieve the same features in traditional databases as well.

VII. CONCLUSION AND FUTURE WORK

In this paper, a database management system to support the uncertain predicates is proposed. The similarity between the values (of strings) is measured in terms of a modified q-gram distances, i.e., local q-gram distance and global q-gram distance. A novel formula to calculate these distances between the strings is proposed and explained in detail. A set of sample queries indicate how this could be beneficial. Probabilities of filtered tuples are then meaningfully derived from the normalized distance arrays, which are treated as a confidence score for that tuple. Finally, the result-set is ranked according to the decreasing values of probabilities, indicating the most likely matches at the top. As an extension, we have also introduced two additional variants of the uncertain predicate which add to the practicability of the system. We demonstrated the significance of our proposed system through the various sample queries. We believe that our system will surely be helpful for the users having a minimal or superficial knowledge of the actual existing data in the database. The proposed global q-gram distance may find its application in other fields like record linkage, data deduplication, DNA sequence alignment.

The output of our proposed system is a probabilistic relation. The derived column named ‘prob’ is augmented to the result-set as an additional column. A concrete system that could be useful on a wider scale can be implemented by incorporating all proposed procedures in an open-source database system. For further query processing, query evaluation techniques of probabilistic databases can as well be utilized for processing uncertain relations.

REFERENCES

[1] N. Dalvi and D. Suciu, “Efficient query evaluation on probabilistic databases,” VLDB J., vol. 16, no. 4, pp. 523–544, Aug. 2007.
[2] D. Suciu, D. Olteanu, C. Ré, and C. Koch, “Probabilistic databases,” Synth. Lectures Data Manage., vol. 3, no. 2, pp. 1–180, 2011.
[3] O. Hassanzadeh and R. J. Miller, “Creating probabilistic databases from duplicated data,” VLDB J., vol. 18, no. 5, pp. 86–99, Jan. 2009.
[4] B. Kanagal, J. Li, and A. Deshpande, “Sensitivity analysis and explanations for robust query evaluation in probabilistic databases,” in Proc. Int. Conf. Manage. Data (SIGMOD), 2011, pp. 841–852.
[5] B. Qin and J. X. Yu, “Efficient sensitivity analysis for inequality queries in probabilistic databases,” IEEE Trans. Knowl. Data Eng., vol. 29, no. 1, pp. 86–99, Jan. 2017.
[6] J. Chen, Y. Li, and L. Feng, “Sensitivity analysis of answer ordering from probabilistic databases,” in Proc. Int. Conf. Database Expert Syst. Appl. Berlin, Germany: Springer, 2013, pp. 243–258.
[7] S. I. Hakak, A. Kamsin, P. Shivakumara, G. A. Gilkar, W. Z. Khan, and M. Imran, “Exact string matching algorithms: Survey, issues, and future research directions,” IEEE Access, vol. 7, pp. 69614–69637, 2019.
[8] P. Jokinen, J. Tarhio, and E. Ukkonen, “A comparison of approximate string matching algorithms,” Softw., Pract. Exper., vol. 26, no. 12, pp. 1439–1458, 1996.
[9] G. Navarro, “A guided tour to approximate string matching,” ACM Comput. Surv., vol. 33, no. 1, pp. 31–88, Mar. 2001.
