# *q*-Gram Tetrahedral Ratio (qTR) for Approximate Pattern Matching

Dr. Greg Holland, Dr. John R. Talburt - University of Arkansas at Little Rock
greg@gregholland.com jrtalburt@ualr.edu

## Abstract

*Text-based data values often exhibit variations due to differences in valid spellings, misspellings, abbreviations, transpositions, deletions, aliases, etc. Traditionally, exact matching is used to facilitate record linkage or other combinations of data values. Exact matching does not allow for the variations which are common to data values, leaving many gaps in record linkage efforts. Approximate pattern matching techniques such as edit distance and Soundex are common approaches, however, both methods can fail to differentiate the variety of results. This research provides a simple yet systematic approach to approximate pattern matching and text comparison utilizing tetrahedral numbers and a comprehensive q-gram algorithm.*

Keywords: Pattern Matching, Approximate String Matching, *q*-Gram, *n*-Gram, Tetrahedral Number, Triangular Pyramidal Number

## 1. Introduction

String matching is based upon exact matching techniques, most often associated with database statements written in SQL, either in the form of the GROUP BY clause or the WHERE String1=String2 expression. This type of matching is valid but incomplete.

The text string "NICK" cannot be matched to "NICHOLAS", though the two names are often interchangeable for the same person. A table of known aliases or nicknames may be available which could allow for NICK-to-NICHOLAS matching, however, these tables are always incomplete and do not normally include transpositions, such as "JOHN" to "JONH". Infrequently used names are not likely to be present in a standard nickname table, though there may be multiple variations of the name for the same individual..

Furthermore, nickname or alias tables are domain-specific and often field-specific, which is an undesirable requirement if attempting any general-purpose text comparisons. When the strings being compared are alphanumeric, the concept of "nicknames" may have no meaning. For example, consider "T8R9X" compared to "TBR9X". It may be possible to create a table of aliases for domain-specific alphanumeric values, however, it is unlikely that all possible errors could be anticipated in advance.

## 2. Background

### *2.1 Standard approaches*

Standard approaches to approximate string matching include both Edit distance and Soundex. Soundex immediately fails when two strings do not begin with the same character.[1] Additionally, Soundex was designed for words that "sound alike", and not words which may be typed incorrectly. The transposition of two consonants in a string are likely to result in two different Soundex values, which makes approximate matching very difficult.

Edit distance, in its most basic form, returns an integer value representing the minimal number character insertions, deletions, or substitution necessary to transform one string into the other. By its definition, edit distance is not sensitive to the position of where string differences occur. For example, all pairs of strings differing in only one character all return an edit distance of 1. Weightings may be incorporated into the edit distance formula to differentiate results by position, but this process increases complexity even further. [3]

### *2.2 q-Grams*

The term '*q*-gram' (also called '*n*-gram') refers to a subsequence of *q* items from a given sequence. [4] With respect to data values such as the earlier examples of the text value "CHRIS", all possible *q*-grams where $q = 3$ would be the subsequences "CHR", "HRI", and "RIS". The value of *q* can be any number between 1 and the length of the sequence. A variety of existing methods for pattern matching utilize *q*-grams in their approach. [5]

### *2.3 Tetrahedral Numbers*

A tetrahedral number, also called a triangular pyramidal number, is a figurate number corresponding to the number of discrete points arranged into a tetrahedron (triangular base pyramid). Calculation of a tetrahedral number follows the formula: [6]

$$T_n = \frac{n(n+1)(n+2)}{6}$$

For example, the tetrahedral number calculated for $n = 4$ is $T_n = 20$, illustrated as 20 discrete points in Figure 1.

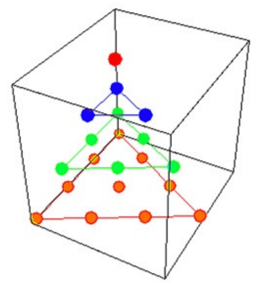
Figure 1: A tetrahedral arrangement for $T_n$ with side length $n = 4$, represented by 20 discrete points.

## 3. *q*-Gram Tetrahedral Ratio (qTR)

### *3.1 Triangular Arrangement*

Consider the text string "JOHN". Possible *q*-grams for "JOHN" are "J", "O", "H", and "N", when $q = 1$; "JO", "OH", and "HN", when $q = 2$; "JOH" and "OHN", when $q = 3$; and "JOHN" when $q = 4$. These 10 subsequences constitute all possible *q*-grams for "JOHN". If each letter in these ten subsequences is considered a discrete point, there are 20 discrete points for the text string of length 4, identical to the calculation of $T_n$ for $n = 4$.

A triangular arrangement can be used to illustrate all possible subsequences for a pattern (string of characters) of any length. In each case, the length $n$ of the pattern will represent the number of subsequences on each side of the triangle, the number of subsequences in total will be $\Sigma n$, and the total number of characters will be $T_n$.

Figure 2 has a triangular shape with each side consisting of 4 subsequences, there are 10 total subsequences, and the number of discrete points (individual characters) is 20. Though the display is two-dimensional, the 20 discrete points could theoretically be positioned to match Figure 1.

| | | | |
|---|---|---|---|
| J | JO | JOH | JOHN |
| O | OH | OHN | |
| H | HN | | |
| N | | | |

Figure 2: Triangular arrangement of subsequences for "JOHN".

### *3.2 String Pattern Comparisons*

Combining the *q*-gram subsequences for character strings with the mathematical aspects of tetrahedral numbers allows for a comparison of any two string patterns. Rather than selecting a particular value for *q*, this more comprehensive approach would include all possible values for *q*.

To compare any two strings, S1 and S2, the comprehensive *q*-gram subsequences for S1 are located within S2, if possible. Suppose that S1 = "JOHN" and S2 = "JONH". Each of the ten possible *q*-gram subsequences in Figure 2 for "JOHN" are indicative of pattern similarity if those subsequences can also be located as a subsequence of "JONH". Figure 3 highlights the subsequences from Figure 2 in common with subsequences of string "JONH".

| | | | |
|---|---|---|---|
| J | JO | JOH | JOHN |
| O | OH | OHN | |
| H | HN | | |
| N | | | |

Figure 3: Highlighted subsequences of "JOHN" which are shared with subsequences of "JONH". *{"J","O","H","N","JO"}.*

### *3.3 Calculation of Tetrahedral Similarity*

In order to determine the similarity of two strings (S1 and S2) utilizing the tetrahedral aspects of *q*-gram subsequences, $T_n$ is calculated where $n$ is the length of S1, and Q is the count of discrete points (characters) shared by the *q*-gram subsequences in common to both S1 and S2.

The simple ratio *qTR* can be calculated as:

$$qTR = \frac{Q}{T_n}$$

Given that $0 \leq Q \leq T_n$, it is true that $0 \leq \text{qTR} \leq 1$. For the example where S1="JOHN" and S2="JONH", qTR = 0.3. This qTR value is obtained because:

- Q = 6, as illustrated by the 6 discrete points (individual characters) in the 5 subsequences shared by S1 and S2, highlighted in Figure 3
- n = 4, the length of S1= "JOHN"
- $T_n$ = 20, the tetrahedral number when $n = 4$, and the total number of discrete points (individual characters) in Figures 2 and 3
- qTR = Q / $T_n$ = 6 / 20 = 0.3

Another example mentioned in the introduction of this research was that of "T8R9X" and "TBR9X". In this example, qTR = 11 / 35 = 0.314. There are 11 characters in the 7 subsequences highlighted in Figure 4 reflecting the same process used for Figure 3, allowing S1="T8R9X" and S2="TBR9X".

| | | | | |
|---|---|---|---|---|
| T | T8 | T8R | T8R9 | T8R9X |
| 8 | 8R | 8R9 | 8R9X | |
| R | R9 | R9X | | |
| 9 | 9X | | | |
| X | | | | |

Figure 4: Highlighted subsequences of "T8R9X" which are shared with subsequences of "TBR8X". *{"T","R","9","X","R9","R9X"}.*

### *3.4 Software Implementation of qTR*

Despite the comprehensive *q*-gram aspects of the qTR approach, the coded implementation of qTR is surprisingly simple. Utilizing Visual Basic for Applications (VBA), the implementation of the proposed qTR function would consist of only a dozen lines of code as a simple nested loop.

```
i = 1
Do While i <= Len(S1)
    j = 1
    Do While j <= Len(S1) - i + 1
    Tn = Tn + j
        If InStr(S2, Mid(S1, i, j)) > 0 Then
        Q = Q + j
        End If
    j = j + 1
    Loop
i = i + 1
Loop
qTR = Q / Tn
```

Figure 5: VBA implementation of qTR

The qTR implementation would also include the function declaration including the input variables S1 (String 1) and S2 (String 2) as well as the declaration of integer variables i, j, Q, and Tn with initial values of zero. Though other modifications or error checking may be optionally included, no other code would be necessary to implement qTR.

## 4. Experimental Results

### *4.1 Limited Commutability*

It is possible to obtain a qTR result which is applicable even if the order of S1 and S2 values are reversed. “JOHN” and “JONH” can be interchanged with no effect on the qTR result of 0.3. Likewise, “T8R9X” and “TBR9X” can be interchanged with no effect on the qTR result of 0.314. This string commutability is only a special case of the qTR algorithm, limited to cases where the lengths of S1 and S2 are identical.

In the event that S1 and S2 have differing string lengths *n*, the respective values of $T_n$ also differ. As a consequence, the qTR result are expected to depend upon the ordering of the two comparison strings. Observe the difference in qTR results when evaluating S1=“NICK” and S2=“NICHOLAS”:

- Q = 10 due to the shared subsequences of “N”, “I”, “C”, “NI”, “IC”, and “NIC”.
- $n = 4$, corresponding to the length of S1.
- $T_n = 20$ when $n = 4$.
- qTR = Q / $T_n$ = 10 / 20 = 0.5.

Compare to S1=”NICHOLAS” and S2=”NICK”:

- Q = 10 due to the shared subsequences of “N”, “I”, “C”, “NI”, “IC”, and “NIC”.
- $n = 8$, corresponding to the length of S1.
- $T_n = 120$ when $n = 4$.
- qTR = Q / $T_n$ = 10 / 120 = 0.083.

Because the interchange of the two strings can cause the value for qTR to differ, it may be desirable to devise a method to ensure consistent results for calculating qTR regardless of string ordering.

### *4.2 Adjusted qTR Methodology*

The qTR calculation is determined by the values of Q and $T_n$. The value of Q for any two strings is a constant, determined by the particular subsequences shared regardless of string order. Consequently, the differing values in qTR occur when the length values differ. A simple adjustment for qTR would incorporate both possible values of *n* in order to eliminate the impact of string order. Utilizing a length-weighted average for the two qTR results effectively produces the desired order-independent effect. If *n*1 represents the length of S1 and *n*2 represents the length of S2, an *n*-weighted average for qTR would be calculated as:

$$qTR(adjusted) = \frac{\frac{n1 \times Q}{T_{n1}} + \frac{n2 \times Q}{T_{n2}}}{n1 + n2}$$

Although the formula appears significantly more complicated than the earlier version, the change to the code is minor, requiring only the introductions and assignments of a few new variables.

```
i = 1
Do While i <= Len(S1)
    j = 1
    Do While j <= Len(S1) - i + 1
        If InStr(S2, Mid(S1, i, j)) > 0 Then
        Q = Q + j
        End If
    j = j + 1
    Loop
i = i + 1
Loop
n1 = Len(S1)
n2 = Len(S2)
Tn1 = (n1)(n1+1)(n1+2)/6
Tn2 = (n2)(n2+1)(n2+2)/6
qTR = (n1*Q / Tn1 + n2*Q/Tn2)/(n1+n2)
```

Figure 6: VBA implementation of adjusted qTR.

Revisiting earlier examples, the adjusted qTR values would be:

- qTR = 0.3 for ("JOHN","JONH")
- qTR = 0.314 for ("T8R9X","TBR9X")
- qTR = 0.222 for ("NICK","NICHOLAS")

The standard approaches to approximate string matching do not demonstrate this precision in result differentiation. For the three examples above, Soundex results in a "match" for ("JOHN","JONH") and a "no match" for the other two cases. Edit distances for the three examples are 1, 1, and 5, respectively.

## 5. Conclusion and Future Work

### *5.1 Executive Summary*

Summarizing the findings of this research, the proposed qTR methodology:

- utilizes all possible *q*-gram subsequences for two strings
- incorporates the mathematical concept of tetrahedral numbers
- determines a similarity ratio for any two strings
- is not dependent upon the order of the two strings
- requires minimal code to implement

Additionally, the qTR methodology as described appears to:

- have no limitations to any particular set of characters
- be applicable for both left-to-right (LTR) and right-to-left (RTL) directional text situations, provided both strings are written in the same manner

### *5.2 Recommendations*

A minimum qTR value should be determined through subject matter expertise for the intended implementation. The purpose of approximate pattern matching is to increase automated record linkage. Valid linkages will be determined by the user and should represent those "near matches" that the user would approve if doing the comparison work manually. It may be necessary to determine multiple qTR value ranges corresponding to those string comparisons which are deemed highest-confidence, acceptable, unacceptable, or worthy of visual inspection.

Implementation of the qTR for non-Western alphabet/keyboard data would be beneficial to further research the "universal" aspects of the methodology. Subject matter expertise in those languages or data sets would enhance the research immensely.

Though the qTR as described is neither domain-specific nor field-specific, it is understood that the implementation of the qTR to specific applications may be enhanced by domain-specific or field-specific coding adjustments, such as an "extra credit" factor for strings which begin with the same letter. The further enhancements to the qTR may improve the performance of the approximate string matching by incorporating elements from alternate methods, such as the phonetic aspects of Soundex.

Comparison and modification related to additional techniques, such as the Jaro-Winkler String Comparison[7] may enhance the utility of qTR.

The qTR may be utilized to create nickname or alias tables for a particular implementation if frequently-occurring string combinations are determined to be acceptable as matches.

It is recommended that the qTR approach be introduced to both computer science and information science students, potentially during database application instruction, or whenever string matching is discussed. The relative simplicity of the qTR implementation should allow for in-depth discussion of the intent, statements, and execution of the qTR function. Additionally, students of number theory in mathematics may appreciate the practical application utilizing tetrahedral numbers as a key factor of the methodology.

## References

[1] National Archives and Records Administration, "The Soundex Indexing System", May 30, 2007, http://www.archives.gov/genealogy/census/soundex.html, retrieved January 10, 2010.

[2] Gilleland, M., "Levenshtein Distance, in Three Flavors", 2009, http://www.merriampark.com/ld.htm, retrieved October 10, 2009.

[3] Damerau, Fred J., "A Technique for Computer Detection and Correction of Spelling Errors," *Communications of the ACM,* 7 (3), 171-6, 1964.

[5] Christen, Peter, *A Comparison of Personal Name Matching: Techniques and Practical Issues,* Australian National University, TR-CS-06-02, 2006.

[5] Gravano, Luis, et al., "Using *q*-grams in a DBMS for Approximate String Processing," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering,* 24 (4), 28-34, 2001.

[6] Weisstein, Eric W. "Tetrahedral Number." http://mathworld.wolfram.com/TetrahedralNumber.html, Wolfram - Mathworld, retrieved February 10, 2010.

[7] LingPipe, "Code Spelunking: Jaro-Winkler String Comparison," *LingPipe Blog*, December 13, 2006, http://lingpipe-blog.com/2006/12/13/code-spelunking-jaro-winkler-string-comparison, retrieved November 22, 2009.