DEIM Forum 2011 B5-3

A Study on Similar String Matching

Zhenglu YANG and Masaru KITSUREGAWA
Institute of Industrial Science, the University of Tokyo
Tokyo, Japan
{yangzl, kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract Approximate querying on string collections is an important data analysis tool for many applications, and it has been exhaustively studied. However, the scale of the problem has increased dramatically because of the prevalence of the Web. In this paper, we aim to study the efficient top-k similar string matching problem. Several efficient strategies are introduced, such as length aware and adaptive q-gram selection. We present a general q-gram based framework and study the efficient strategies introduced experimentally on real data sets.
Key words Top-k, Similar String Search, Q-gram

1 Introduction

Similar string searching is an important problem because it involves many applications, such as query suggestion in search engines, spell checking, similar DNA searching, and so forth. From a given collection of strings, such queries ask for strings that are similar to a given string, or those from another collection of strings [5].

We focus on similarity search with edit distance thresholds in this paper. Many algorithms have used q-gram based strategies [9, 7, 6] to measure the edit distance between two strings. A q-gram is a consecutive substring of a string with size q that can be used as a signature of the string. To hasten the search process, many approaches pre-construct some index structures (e.g., suffix tree [10]) and then search these compact indices in an efficient way. However, most of the works are rooted in a threshold-based framework (i.e., error threshold predefined), and there are not many works on exploring the top-k issue of the problem.

In this paper, we study several efficient strategies to address the top-k similar string matching problem. Specifically, we introduce an adapted string length-aware technique and optimally select the q-gram dictionary. For the second strategy, we construct a set of q-gram dictionaries in the preprocessing. In each iteration, we select an appropriate value of q (hence, a q-gram dictionary is chosen) based on the top-k similarity score of the last iteration. The new value of q is guaranteed to increase, which indicates that the size of the inverted lists is decreasing (i.e., large q leads to shorter inverted lists). Hence, the cost of the frequency counting is reduced.

2 Problem Statement

Let Σ be an alphabet. For a string s of the characters in Σ, we use "$|s|$" to denote the length of s, "s[i]" to denote the i-th character of s (starting from 1), and "s[i,j]" to denote the substring from its i-th character to its j-th character.

Q-Grams: Given a string s and a positive integer q, a positional q-gram of s is a pair (i, g), where g is the q-gram of s starting at the i-th character, that is $g = s[i, i+q-1]$. The set of positional q-grams of s, denoted by $G(s, q)$, is obtained by sliding a window of length q over the characters of string s. There are $|s| - q + 1$ positional q-grams in $G(s, q)$. For instance, suppose q = 3, and s = **university**, then $G(s, q)$ = {(1,**uni**), (2,**niv**), (3,**ive**), (4,**ver**), (5,**ers**), (6,**rsi**), (7,**sit**), (8,**ity**)}.

Top-k Approximate String Queries: We denote the edit distance between s_1 and s_2 as $ed(s_1, s_2)$. For example, ed("blank", "blunt") = 2. In this paper, we consider the top-k approximate string query on a given collection of strings S. Formally, for a query string Q, finding a set of k strings R in S most similar to Q, that is, $\forall r \in R$ and $\forall s \in (S - R)$ will yield $ed(Q, r) \leq ed(Q, s)$.

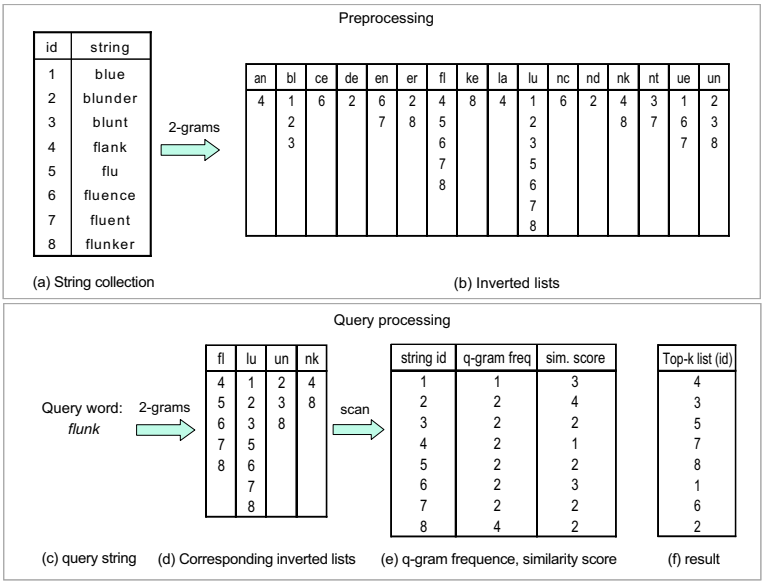


Figure 1: A general framework for approximate top-k string search with q-gram

To extract the top-k similar strings efficiently, we apply the q-gram strategy [9] with deliberate optimization, as will be discussed shortly. The q-gram similarity of two strings is the number of q-grams shared by the strings, which is based on the following lemma.

Lemma 1 (q-gram lemma [3])**.** *Let P and S be strings with the edit distance τ. Then, the q-gram similarity of P and S is at least $t = max(|P|, |S|) - q + 1 - q \cdot \tau$.*

2.1 A General Framework for Top-k Similar String Query Processing

We introduce the general q-gram based top-k querying system in this section. The main issue is that the cost of counting the frequency of the q-grams is high. An example illustrating the details of the framework is shown in Fig. 1, which presents the preprocessing and the query processing.

Example 1. Suppose a user wants to obtain the top-1 similar word to $flunk$. While querying, the query word $flunk$ is parsed into four 2-grams as fl, lu, un, and nk; their corresponding inverted lists are $\{4, 5, 6, 7, 8\}$, $\{1, 2, 3, 5, 6, 7, 8\}$, $\{2, 3, 8\}$, and $\{4, 8\}$ respectively. The lists are merged to count the frequency of the strings in the collection resulting in 1:1, 2:2, 3:2, 4:2, 5:2, 6:2, 7:2, and 8:4, where each pair of values is of type sid:freq. Sid and freq represent the string ID and the frequency value of the n-grams included by the string, respectively. Finally, the edit distances between the candidate strings and the query are computed as $blue$:3, $blunder$:4, $blunt$:2, $flank$:1, flu:2, $fluence$:3, $fluent$:2, and $flunker$:2, where each pair of values is of type string:distance. The top-1 similar string to the query $flunk$ is thus $flank$.

The most time consuming step in the query processing is to count the frequency of q-grams, which has to scan a large amount of inverted lists. Recently, [8, 4] have proposed efficient strategies (i.e., divide-merge-skip) to count the frequency of q-grams. The key idea is that it first divides the inverted lists into short lists and long lists. The whole short lists are then scanned, and the elements are collected. Finally, it binary searches these elements in the long lists to find strings whose frequencies are larger than the threshold. Refer to [8, 4] for details. In this paper, we apply this efficient technique.

3 Top-k Similar Search Strategies

In this section, we study two algorithms for top-k approximate string search. We first introduce two filtering strategies (i.e., count filtering and length filtering) for the first algorithm and then we present the adaptive q selection strategy for the second algorithm.

3.1 Count Filtering

The intuition of count filtering is that strings within a small edit distance of each other share a large number of q-grams in common. The q-gram count filtering for the top-k similar string search is derived from that of the traditional threshold-based framework [9].

Lemma 2 (q-gram lemma for top-k approximate string search)**.** *Let P be the query string and S be the top-k similar string in the candidate string list so far. P and S have the edit distance τ'_{top-k}. Then, the q-gram similarity t'_{top-k} of P and S is at least*

$$t'_{top-k} = max(|P|, |S|) - q + 1 - q \cdot \tau'_{top-k} \quad (1)$$

3.2 Length Filtering

The intuition of length filtering is that if there are two strings within a small edit distance τ of each other, then the difference of their lengths will not exceed τ.

When scanning the inverted lists, we choose to test the candidate strings in ascending order of the difference between their lengths and that of the query string. The process can be early terminated based on the following lemma.

Lemma 3 (Early terminate)**.** *Let P be the query string and S be the top-k candidate similar string so far. We denote the set R of the remaining untested strings. If $\forall r \in R, ed(P, S) \leq ||P| - |r|| + 1$, then the search process can be early terminated.*

The length filtering for the top-k similar string search is derived from that of the traditional threshold-based framework [2, 1].

The branch and bound manner (BB) algorithm is constructed based on count filtering and length filtering. For simplicity and without loss of generality, we assume that a specific q-gram set is constructed in the preprocessing. The query processing is executed as follows:

- Initialization: Set the frequency threshold t'_{top-k}=1, and the length difference ld=0. P is parsed into a set of q-grams. The following steps are executed iteratively until k similar strings are output.
- Branch: Extract the corresponding inverted lists in which any string S has $||S| - |P|| = ld$.[1] Scan these inverted lists and discover the candidate similar strings whose frequencies are no less than t'_{top-k}.[2]
- Bound: Rank the candidate strings and obtain the temporal edit distance threshold τ'_{top-k}. Terminate the process if the top-k string Q in the candidate list so far has $ed(P, Q) \leq (ld + 1)$. Compute the temporal frequency threshold t'_{top-k} based on τ'_{top-k} (Lemma 2) and set $ld = ld + 1$.

3.3 Adaptive q-gram Selection

We assume from above that the value of q is static, which indicates that we only use one set of q-gram inverted lists in the entire process. However, as well known in the literature, a larger value of q results in a smaller size of inverted lists, which may reduce the cost of the frequency counting. We select an appropriate q-gram dictionary (may be varying) in each iteration, which is calculated by the top-k similarity score in the last iteration. As illustrated in the experiments, this strategy can reduce the size of the inverted lists to be scanned, therefore improving the query performance.

Lemma 4 (Adaptive q selection)**.** *Let P be the query string and S be the top-k similar string in the candidate string list so far. P and S have the edit distance τ'_{top-k}. Then, the adaptive value of integer q, q'_{top-k}, is selected in the following formula.*

$$q'_{top-k} = \begin{cases} \frac{max(|P|,|S|)}{\tau'_{top-k}+1} & , \quad if \frac{max(|P|,|S|)}{\tau'_{top-k}+1} \geq 2 \\ 2 & , \quad else \end{cases} \quad (2)$$

The adaptive q-gram algorithm (AQ) is constructed based on the three strategies introduced above (i.e., count filtering, length filtering, and adaptive q-gram selection).

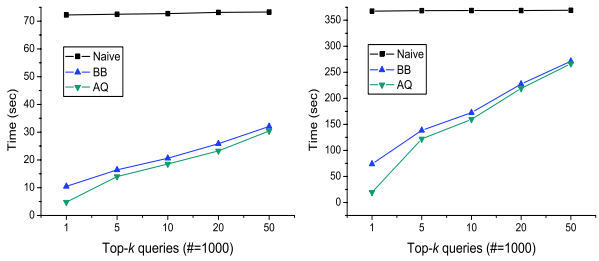
[1] We pre-build an index to record strings based on their lengths.

[2] Divide-merge-skip strategy is applied as aforementioned.

4 Performance Analysis

We performed the experiments using a Intel(R) Core(TM) 2 Dual CPU PC (3GHz) with a 2G memory, running linux. The $Naive$ algorithm was implemented based on the general top-k querying framework. All the algorithms were written in C++. The default value of q is set to 2 for $Naive$ and BB, while for AQ algorithm the default dictionaries are [2,3]-grams. We conducted experiments on the real life data sets, $Dict$[3], and $Person$[4]. Due to space limitation, more experimental results on other real data sets are avoided and will be shown in the presentation.

4.1 Efficiency of Query Processing



(a) **D**ict dataset (b) **P**erson dataset

Figure 2: Query performance of different algorithms

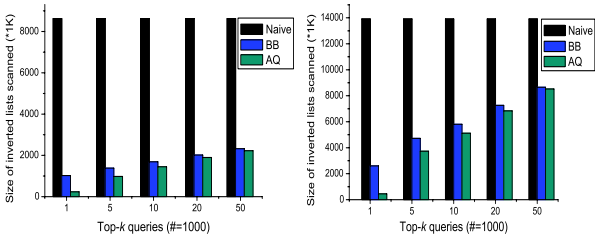
We evaluate the query performance of BB and AQ when varying the value of k. To test the effect of top-k query, we randomly selected 1000 different queries from the data sets. The result is shown in Fig. 2, from where we can see that BB and AQ always perform better than $Naive$, especially when k is small. This is not surprising because of the length filtering and count filtering strategies employed. Between the two algorithms, AQ is superior to BB because the former can reduce the cost by adaptively selecting a relative large value of q.

4.2 Size of inverted lists scanned

We evaluate the size of inverted lists scanned in different algorithms when varying the value of k. As illustrated in Fig. 3, we can see that BB and AQ scan smaller sets of inverted lists compared with the other algorithm. This is the intrinsic reason why the algorithms performed better on query processing (Fig. 2).

[3]www.aspell.net/

[4]www.trec.state.tx.us/LicenseeDataDownloads/trecfile.txt



(a) **D**ict dataset (b) **P**erson dataset

Figure 3: Inverted lists scanned among algorithms

5 Conclusions

In this paper we have studied the efficient top-k approximate string searching problem. Several efficient strategies are studied experimentally, i.e., length filtering and adaptive q-gram selection. The results show that the approaches can efficiently answer the top-k string queries.

References

[1] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, 2001.

[2] M. Hadjieleftheriou, A. Chandel, N. Koudas, and D. Srivastava. Fast indexes and algorithms for set similarity selection queries. In *ICDE*, pages 267–276, 2008.

[3] P. Jokinen and E. Ukkonen. Two algorithms for approximate string matching in static texts. In *In Proc. 2nd Ann. Symp. on Mathematical Foundations of Computer Science*, pages 240–248, 1991.

[4] C. Li, J. Lu, and Y. Lu. Efficient merging and filtering algorithms for approximate string searches. In *ICDE*, pages 257–266, 2008.

[5] C. Li, B. Wang, and X. Yang. Vgram: improving performance of approximate queries on string collections using variable-length grams. In *VLDB*, pages 303–314, 2007.

[6] G. Navarro. A guided tour to approximate string matching. *ACM Computing Survey*, 33(1):31–88, 2001.

[7] G. Navarro and R. A. Baeza-Yates. A practical q - gram index for text retrieval allowing errors. *CLEI*, 1(2), 1998.

[8] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *SIGMOD*, pages 743–754, 2004.

[9] E. Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science*, 92(1):191–211, 1992.

[10] E. Ukkonen. Approximate string-matching over suffix trees. In *CPM*, pages 228–242, 1993.