Synopsis of Fast Algorithms for Top-k Approximate String Matching

CS 518 Advanced Analysis of Algorithms
Project 2 - Fall 2017

Priya N Subramanian

Abstract

I have implemented two algorithms from the paper "Fast Algorithms for Top-k Approximate String Matching". This document briefly describes the goal of the algorithm, my empirical study of performance compared to the performance claims in the paper and my own thoughts on the paper.

1 Objective of the Algorithm:

In this paper, the authors have studied the efficient top-k approximate string searching problem. They have presented two algorithms by combining the length filtering and adaptive q-gram strategies. They have applied the divide-merge-skip and the dynamic count filtering techniques along with the adaptive length-aware and q-gram selection strategies to minimize the size of the inverted lists corresponding to the query string. They have conducted extensive experiments on real data sets, the results show that their approaches can efficiently answer the top-k string queries.

2 Implementation Details:

Since both **"Branch and Bound"** (BB) and "Adaptive q-gram"(AQ) algorithms boosts the performance of the Top-k approximate string matching by the same factor, I chose to implement the BB algorithm and concentrate more on the empirical studying task. I have also implemented the **"Similar String Search by Q-gram Matching"** algorithm (Naive algorithm without any filters) presented in the paper for the comparison purposes.

2.1 Similar String Search by Q-gram Matching :

They have illustrated the Naive algorithm with an example. But this q-gram based framework suffers from a serious disadvantage: counting the frequency of the q-grams has to scan a lot of inverted lists and consumes a lot of time. To overcome this, the authors have devised efficient algorithms for top-k approximate string matching.

2.2 Branch and Bound Algorithm :

The authors have proposed an algorithm based on the Branch and Bound manner which uses count filtering and length filtering strategies. I performed three main tasks for the query processing as mentioned in the paper: Initialization, Branch, and Bound.

- Initialization: Constructed the q-gram dictionary using trie datastructure (pygtrie) in the pre-processing.
- Branch: Initially length difference was set to 0 and the corresponding inverted lists were extracted in which any string S has $||S| - |P|| = ld$. The process is repeated by incrementing ld at each iteration.
- Bound: Ranked the candidate strings based on their levenhstein distance from the query and top-k values were obtained. once we have k candidates, frequency threshold is calculated and used to filter candidates in the subsequent iterations. The process was terminated if the top-k-th string Q in the candidate list so far has ed(P, Q) <= (ld + 1) (which means we are not going to find better matches with candidates whose size difference itself is greater than edit distance the worst candidate in our top-k list)

3 Empirical Study:

I've conducted experiments to evaluate the efficiency of both algorithms that was implemented. The Naive algorithm was implemented based on the general top-k querying framework. All the algorithms are written in Python. The default value of q is set to 2 for Naive and BB, like in the paper. The dictionary input consists of approximately 236,000 words out of which the top-k matches are obtained.

3.1 Efficiency of Query Processing:

In this section, I have evaluated the query performance of the proposed algorithms when varying the value of k. The result is shown in Fig. 1, from where we can see that my implementation of the BB algorithm always performs better than Naive, especially when k is small just like in the paper. This is mainly contributed to the dynamic length filtering and count filtering strategies used.

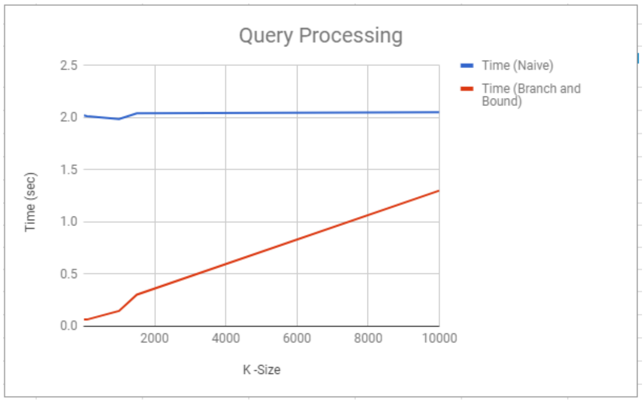

Figure 1: Query performance of Naive and BB algorithms

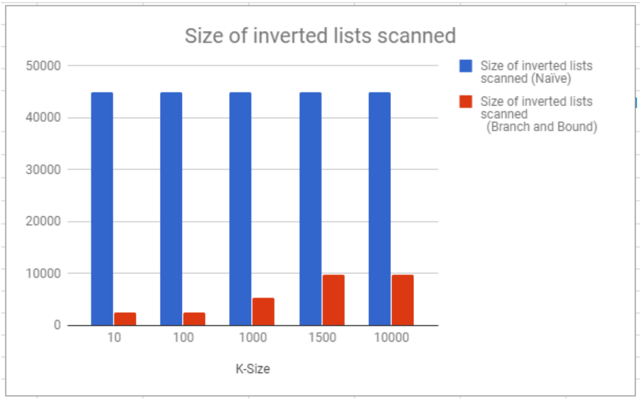

Figure 2: Inverted lists scanned among Naive and BB algorithms

3.2 Size of inverted lists scanned:

In this section, I have evaluated the size of inverted lists scanned in the two algorithms when varying the value of k. As illustrated in Fig. 2, we can see that BB scans smaller sets of inverted lists compared with the other algorithm.

4 Personal Thoughts :

4.1 Applications of String Matching :

Because of the prevalence of the web, the number of applications that require efficient querying of a large database of texts to search for similar strings or substrings has dramatically increased. The algorithms proposed in this paper has reduced space consumption and computes faster. String matching algorithms, also known as String searching algorithms, is variably used in a wide range of real time applications like Database schema, Network systems, Spell Checkers, Spam Filters, Intrusion Detection System, Search Engines, Plagiarism Detection, Bioinformatics/ DNA Sequencing, Digital Forensics and Information Retrieval Systems. Approximate String matching can be defined as the problem of string matching that allows inaccurate results. And for most of the applications cited above, it is not imperative to get the exact top-k match while what's more important is rather how efficiently we can arrive at a solution that's agreeable.

4.2 Suggested Improvements :

When K is significantly large, Branch and Bound performs worse than the Naive approach. We can come up with a combination of both these approaches which would choose the best approach based on the K. This would give better performance in cases of larger k values

The proposed algorithms increases the ld threshold every iteration and tunes the gram length dynamically. It still needs to build redundant inverted indexes for different gram lengths. They could further improve the performance by identifying the strings with smaller edit distances to the query string and use the edit distances of these strings as bounds to facilitate computing top-k answers.

4.3 String Matching in Sentiment Analysis :

Sentiment Analysis is the process of determining whether a block of text is depicting a positive or negative sentiment. I am currently working on performing sentiment analysis on tweets from Twitter, using Neural networks. Preprocessing the data is the process of cleaning and preparing the text for classification. Online texts usually contain lots of noise and uninformative parts such as HTML tags, scripts and advertisements. Approximate String matching along with regular expression techniques can be used to transform the noise in the tweets to an analyzable text input. Also, a lot of times, users commit a typo in their tweets. This could be a valuable piece of information in determining the sentiment that we'd be missing out on. Fast Top-K string matching could be used to decipher what the user intended.