EECS 405 Project Proposal

for

Improving Efficiency of String Similarity Searches through Clustered Pruning

Prepared by

James Fitzpatrick
Kyle Patterson

Version 1.0
March 18, 2014

Contents

1. Introduction

String similarity search is a rapidly growing area of research within the computer science community due to its numerous overlapping applications in the modern world. From biology and genetics to computer security, improvements to current string searching techniques must be improved to handle the large amounts of data being gathered today. As presented in [2], pruning techniques were effective in reducing the time of a query to less than 50 milliseconds; however, these pruning techniques were computationally costly, requiring a minimum of 0.1 billion entries in generated triples in the range-based pruning methods. With the move towards big data sets, it may be more beneficial to include common data-mining practices that are already designed for large sets of data.

2. Problem Definition

We propose a method of pruning the search space that utilizes a common data mining technique, clustering. At the application's initialization, we will mine the database and identify similar groups of strings; these clusters will be indexed by longest common substring. When a search is performed at run time, we will search for the appropriate cluster of strings using the constructed cluster index and restrict our search space to the appropriate cluster of data. Although this method includes a significant amount of preprocessing, index construction is only needed when the data is modified and is not present in every search. This effectively reduces the number of strings that must be compared via the B^{ed}-Tree and Top-k techniques from [1] and [2], which should further improve search time among large sets of data.

As presented in [9], using longest common subsequence (LCS) is the second most effective technique in grouping strings (with the Levenshtein distance being the most effective). However, LCS is more conducive to key generation because it provides a testable string that can be used as the key. The best-matching LCS of a query and a key is guaranteed to identify the cluster that contains the query, if the query is present in the database.

3. Goals

Throughout the course of the project, the following goals will be accomplished:

1. Conduct a literature survey on String Similarity Search
2. Implement the B^{ed}-Tree and Top-k Algorithms
3. Build clustered datasets based on the Longest Common Subsequence on:
   - Movie Titles - IMDB
   - Publication Titles - DBLP
4. Test the effects of the proposed clustering techniques on the overall performance of the algorithm in terms of:
   - Correctness/Accuracy
   - Speed
5. Test the effects on the performance of the proposed clustered search space by comparing to the performance in an unclustered search space
6. Formally present findings through a final presentation

4. Project Members and Task Distribution

James Fitzpatrick will be implementing the Top-k Algorithim as documented in the papers below. James will also create an agnostic test harness to assess the differences between the two search algorithms on the clustered and unclustered datasets.

Kyle Patterson will be implementing the B^{ed}-Tree Algorithm and will be developing the clustering algorithm for the datasets.

5. Literature Review

1. Z. Zhang, et. al. "B^{ed}-Tree: An All-Purpose Index Structure for String Similarity Search Based on Edit Disctance", SIGMOD 2010.

2. Don Deng, Guoliang Li, Jianhua Feng, Wen-Syan Li, Top-k String Similarity Search with Edit-Distance Constraints, ICDE 2013.

   Top-K is a well-documented algorithm implemented both naively and using optimizations that reduce the number of comparisons used when comparing each string. In this paper, the authors propose a progressive framework that well help limit the number of comparisons performed. We find the strings that are of a set edit distance, starting at 0, in our set of string to match a query string. If we have our requisite number of strings, return the found set of strings. Else, we increment the edit distance and find the strings that are of a edit distance of x+1 away from our query string until k results are found. Using trie structures, we can accelerate this process by comparing many strings simultaneously.

   Note that the edit distances used here also allow for user error to an extent, which is a desirable feature for any searching agent to have.

   We will be using the algorithms defined in the paper as the basis for our comparisons of our backing structures to see how clustering our dataset will improve our search times.

3. A. Andoni and K. Onak. Approximating edit distance in near-linear time. STOC, pages 199-204, 2009.

4. M. Hadjieleftheriou, A. Chandel, N. Koudas, and D. Srivastava. Fast indexes and algorithms for set similarity selection queries. ICDE 2008, pages 267-276.

5. M. Hadjieleftheriou, N. Koudas, and D. Srivastava. Incremental maintenance of length normalized indexes for approximate string matching. *SIGMOD Conference*, pages 429-440, 2009.

   Indexes for datasets are usually updated at set time intervals. To prevent unnecessary reindexing, this paper recommends a lazy update propagation that allows for efficient incremental updates that immediately reflect new data. Since removing strings can cause a number of dead grams, particularly in the case of less frequent grams, we need to

be careful about when we update. The algorithm proposed creates a log of all of the changes to be made to the dataset, then will push all of the changes simultaneously when a memory or time restriction has been met. This will prevent unneeded reorderings from multiple small operations done to the dataset.

This article shows us the importance of safely maintaining our datasets while having manageable indexing structures. This should be taken into consideration when we construct our indexing algorithms, but will not be the main focus of our project.

6. W. J. Masek and M. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18-31, 1980.

7. T. Kahveci and A. K. Singh. An Efficient Index Structure for String Databases. VLDB, pages 351-360, 2001.

   This paper expresses the woes of many database managers, where the size of the database grows exponentially over time. As such searching larger and larger databases need to consider ways to utilize external memory to speed searches.

   With a focus on genetic string databases, the paper suggests mapping strings to integers to integer space with a wavelet function to reduce the problems caused by the sizes of dataspaces. The paper also utilizes Minimum Bounding Rectangles (MBRs) to help with range queries and nearest neighbor searches.

   While useful in the case of genetic structure and other substring sensitive data, the gains for extremely large strings with frequent substring may not hold in the case for movie title, publication title searches.

8. Z. Yang, J. Yu and M. Kitsuregawa. Fast algorithms for top-k approximate string matching. AAAI, 2010.

   While Top-k is a well-known algorithm, there are many ways to improve it from its nave form. This paper examines how to expedite the search. Before going into the exact algorithms, the authors discuss filtering techniques. Count filtering utilizes the intuition that short strings that are a small edit distance away have a large number of shared q-grams. Length Filtering reduces the number of strings compared by eliminating

strings that are of a certain threshold away from our query string. A string of length 13 is unlikely to be the best match for a string of length 4. As such, we dismiss it before performing the comparisons.

The Branch and Bound algorithm utilizes length filtering to find the top-k strings. First a frequency threshold is set to 1 and length difference set to 0. We then Branch (extract the inverted lists of our query string to find the candidates that match given the length and frequency with our limitations) then Bound our strings (rank the candidate strings based on edit distance). We then expand our boundaries of length differences between our stored set of strings and our query until k strings have been found

The Adaptive q-gram selection exploits the two filtering strategies as well as adaptive q-gram selection, which exploits that strings that have small edit distances share most of the same q-grams. By having longer grams, we can (to an extent) exploit these similarities using larger grams until we need to find lower ranking strings until we have found our top k grams.

These adaptations of the Top-k algorithm will prove useful in our consideration of how to best implement the top-k algorithm to exploit our backing clustering system.

9. R. Zafarani and H. Liu. Connecting users across social media sites: a behavioral-modeling approach. KDD, pages 41-49, 2013.

10. M. Rafsanjani, Z. Varzaneh, N. Chukanlo. A survey of hierarchical clustering algorithms. TJMCS 5(3):229-240, 2012.

11. C. Li, B. Wang, X, Yang. VGRAM: Improving Performance of Approximate Queries on String Collections Using Variable-Length Grams. VLDB, September 23-27, 2007.

    Many string comparison algorithms consider grams of set lengths. However, there are many advantages and disadvantages to having any particular length of n-gram. Rather than to have a one-for-all solution to the strings, the authors propose a solution to generate grams of variable length between a minimum and maximum length based on a set gram dictionary, which is constructed based on the data collection of strings prior to the query. When computing the gram dictionary, we should

only store grams that have a frequency above a certain threshold. The algorithm utilizes a trie to reduce the computations of grams that are substrings of longer grams, since the subgrams will necessarily occur when the larger grams are present. Since smaller grams are present in larger ones, the grams that survive pruning will more often than not be shorter grams. Longer grams will be present in data sets only if they are very frequent.

This algorithm exploits many of the same principles that we should see in our clustering techniques as our clusters should be identified by longest common substrings.