# EECS 405 Progress Report 2

**for**

**Improving Efficiency of String Similarity Searches through Clustered Pruning**

**Prepared by**

**James Fitzpatrick**
**Kyle Patterson**

**Version 1.0**
**March 27, 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. Using the BIRCH clustering algorithm, we will create a heirarchical cluster within the data space based off of edit distance. These clusters will be indexed according to Longest Common Subsequence; when a string query is performed, we will search this cluster heirarchy to find an appropriate subspace of data. The query algorithms, namely the $B^{ed}$-Tree, Top-$k$, and VGRAM techniques from [1] [2] and [3], will then be restricted to this data subspace; this effectively prunes the search space according to edit distance, without requiring that the algorithms compute the edit distances at each query.

We will compare the execution times of our proposed pruning technique versus a naïve implementation of the $B^{ed}$-Tree, Top-$k$, and VGRAM algorithms from [1] [2] and [3]. We hypothesize that by pruning off large clusters of data, a large reduction in query time will be seen.

# 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 Execution

This project will progress as defined in the proceeding section, the preliminary timeline. Data is currently stored on an Amazon Web Service hosted virtual machine within a MySQL server; however, this is just for convenience in establishing the clustering algorithms. The tables within the server have not undergone any optimization; thus we can anticipate the access time to be bounded by I/O time of the server. This cost is only incurred upon original construction of the data clusters, and is thus negligable to the string query time. However, this access time may effect the performance of the query algorithms within the unclustered search space; we consider this a portion of the cost incurred without use of the clustering algorithms, and include it as such.

Development will take place on local machines, due to the restricted resources of the virtual machine (ie, limited memory of under 1 GB). The project will be stored on a public GitHub repository, and will utilise the git features for source control. James is responsible for the implementation of the Top-$k$ and VGRAM algorithms, as well as the testing framework. Kyle is responsbile

for implementing the $B^{ed}$-Tree algorithm, as well as the BIRCH clustering algorithm. Because the experimental portion of this project will compare the run time of a query algorithm in a clustered search space versus an unclustered search space, we will develop the query algorithms and the clustering algorithm independently. The query algorithms should include an additional feature to determine where the search space is obtained; that is, given a bit flag, the query should be capable of retrieving data from the unprocessed MySQL server or retrieving an appropriate subset of data from the clustered space.

Algorithms will be developed using the C# language. Our test data includes the titles of over 57,000 movies from IMDB and over 2.5 million publication titles from DBLP.

# 5. Adjusted Timeline

| Date | Objective |
|---|---|
| March 27th | Complete Literature Review |
| March 30th | Complete Workstation Setup |
| April 10th | Finish Top-k and $B^{ed}$ Tree Algorithms |
| April 12th | Finish implementing BIRCH Algorithm |
| April 14th | Finish Implementing VGRAM Algorithm |
| April 15th | Revise BIRCH Algorithm if necessary |
| April 16th | Finish Test Cases For Both Datasets |
| April 18th | Optimize Search Algorithms If Possible |
| April 20th | Finish Testing of Algorithms |
| April 21st | Finish Project |

# 6. Project Progress

## 6.1. Literature Review

We have completed our Literature review of 12 related articles to our future project. The information gathered will help us judiciously detemine which diretion we want to take our various algorithms and also raise many valid concerns that we should be aware of.

## 6.2. Project Server

Currently, we have set up our server for which we will be hosting our database. We will be utilizing Amazon AWS hosting service, using a Linux Machine in Amazon's Cloud. On this machine is a modified version of IMDB and the entirety of DBLP. We have abbreviated the IMDB database due to the size of the database compared to the size of our remote host. Due to hardware limitations on our remote machine, we will be developing our algorithms in C# from local machines for testing and then upload them to the server for our final tests.

## 6.3. Top-K Algorithm

Our Top-K Algorithm is nearly completed. We still need to improve many minor details. Often our numbers along edge cases have been reporting values of numbers that are off by one when considering the ranges of nodes. However, a complete Top-K Algorithm should be finished shortly.

## 6.4. $B^{ed}$-Tree Algorithm

The $B^{ed}$-Tree is operational and has almost cleared the debugging phase. It is on schedule to enter the experimental portion of this project upon completion of the test cases.

## 6.5. BIRCH Algorithm

Progress on the BIRCH Algorithm is under way, though it was delayed from the preliminary schedule due to external constraints. Debugging is scheduled for April 12th to remain on schedule. Revisions will take place during the following days to join the data flow from the output of the clustered structures to the input of the Similarity Search Algorithms.

## 6.6. Challenges

Many of the algorithms that we are examining use very different bases for constructing the backing strucutres. As the Tree Structure in Top-k and the B+ tree exhibit very different properties, using a clustered form of our data

will exploit these strengths and weaknesses in ways we did not originally anticipate. We will need to consider how to coorelate very different algorithms, if we choose to at all, with our backing clustering structure.

# 7. 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.

   This paper presents a modified $B^+$-tree designed to improve performance on edit distance and normalized edit distance queries. This modified tree was designed with four principle ideas: (1) it can support incremental updates efficiently, (2) it can handle arbitrary edit distance thresholds without having to specify a minimum threshold value, (3) it can support any query type over a normalized edit distance, and (4) it can be implemented on top of an existing $B^+$-tree structure. They identify a mapping function $\phi : \Sigma^* \rightarrow \mathbf{N}$ that has four properties (Comparibility, Lower Bounding, Pairwise Lower Bounding, and Length Bounding); these four properties enable it to handle point queries, range queries, range queries over normalized edit distance, and join queries. They then identify three different string orders that may be used in the mapping (Dictionary order, Gram Counting order, and Gram Location order) and explore the computational complexity and differences between each of these.

   It is important to note that in their experiment setup, they relied on both the DBLP and IMDB databases, and added a protein database. They found that the $B^{ed}$ with a gram counting $\phi$ (written **BGC**) was most effective at a top-$k$ range query using normalized edit distance. Because of this, we will be using this design in our implementation of the $B^{ed}$-tree.

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. 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.

4. 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.

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

   This paper presents an algorithm to approximate the edit distance between two strings of length $n$ to a factor of $2^{\tilde{O}(\sqrt{\log n})}$ in $n^{1+o(1)}$ time. They report this is the first sub-polynomial approximation algorithm that runs in near-linear time, beating the previous fastest algorithm that had a running time of $n^{1/3+o(1)}$. Currently, the best exact value algorithm has a complexity of $O(n^2/\log^2 n)$ time and requires constant-size alphabets.

   The algorithm presented in this paper may be used to enhance the speed of the proposed B$^{ed}$-tree solution, which did not identify an algorithm for computing edit distance or normalized edit distance.

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.

   This paper, referenced in [5], is the current best exact value algorithm for computing string edit distances. As mentioned above, it runs in $O(n^2/\log^2 n)$ time, though it requires a constant-size alphabet. Fortunately, our database relies on transliterated titles in the English alphabet, and so the alphabet used is of constant size. The algorithm

computes the matrix of edit operations by dividing this into all possible $(m+1) \times (m+1)$ submatrices of a chosen parameter $m$, then combines these submatrices to form the full matrix and compute the edit distance. Their proposed algorithms can also be computed to calculate the length of the longest common substring in time complexity $O(|A| \cdot B|/log|A|)$, where $|A|$ and $|B|$ are the lengths of strings $A$ and $B$, respectively; this length can then be used in a slightly modified algorithm from their original design to calculate the edit operations and thus the subsequence.

One important issue with the algorithms presented in this paper is that the algorithms require that the domain for the cost function is discrete. However, even if this condition is violated completely, this algorithm still runs in $O(n^2)$ time, which is of equal performance to the generally-taught dynamic programming techniques.

7. T. Kahveci and A. K. Singh. Efficient index structures 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.

   This paper proposes a methodology, referred to as MOBIUS, that identifies users based on the behaviours exhibited in username selection in online communities, such as blogs or social media platforms like Google+. It uses a supervised learning framework, in combination with a calculated 414 feature set per username, to group usernames (ie, strings) according to features such as vocabulary and alphabet requirements, uniqueness, edit distance, longest common subsequence, distance between positions on a QWERTY keyboard, and so on. Although it is an application of data mining techniques, this paper identifies and ranks the top 10 features in identifying string similarity. These ranked features successfully identified groups of strings with a 92.72

10. M. Rafsanjani, Z. Varzaneh, N. Chukanlo. A survey of hierarchical

clustering algorithms. TJMCS 5(3):229-240, 2012.

This survey conducts a comparative experiment to analyze the performance of various well-known clustering algorithms. It identifies two leading methods, BIRCH AND CURE, that use an agglomerative hierarchical approach to identify clusters within the search space. Both of these methods minimize I/O cost, reducing their execution times; however, the BIRCH method is a dynamic model and more effective than the static CURE method at handling data set modification. Additionally, the $O(n)$ complexity of BIRCH defeats the near-quadratic runtime of CURE. This paper was used to identify the best clustering algorithm for our filtering technique.

11. T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH : an efficient data clustering method for very large databases. SIGMOD, 103-114, 1996.

    The algorithm proposed within this paper is still widely used today as one of the most efficient methods for data clustering large data sets. BIRCH, or Balanced Iterative Reducing and Clustering using Hierarchies, utilises a proposed **CF** tree to manage the clusters of data. This **CF** tree is closely similar to a $B^+$-tree; however, the **CF** tree is a much more compact representation of the data, where each leaf node is a subcluster with a specified radius rather than an individual data point. The BIRCH algorithm sees significant progress with just one pass of the data, which constructs the initial **CF** tree with a defined memory limit. Further iterations are used to refine the tree. In the original performance, outline in 1996, BIRCH took under 50 seconds to cluster nearly 100,000 data points. Given its long standing position as leading clustering algorithm, it was awared the SIGMOD 10-year test of time award.