Similar Article Detection Using Locality Sensitive Hashing Technique on Wikipedia*

Samet Ayhan
University of Maryland
Dept. of Computer Science
sayhan@cs.umd.edu

Joshua Bradley
University of Maryland
Dept. of Computer Science
jgbrad1@cs.umd.edu

Sarah Weissman
University of Maryland
College of Information Studies
sew@umd.edu

ABSTRACT

The growth of Internet has enabled collaboration and cooperation on a large scale, resulting in an abundant number of near-duplicate web documents. The range of near-duplicate occurrences is even more evident within Wikipedia articles, due to its editing largely being open to the online community. Except for particularly sensitive and/or vandalism-prone pages that are "protected" to some degree from editing, the reader of an article can edit the text by copying from another article without prior approval. There is no current automated mechanism for truth validation. Although we are not inspired to measure the quality of Wikipedia articles, we are interested in finding near-duplicate occurrences at various granularity levels.

In this paper, we describe a novel similarity detection algorithm that utilizes a locality sensitive hashing (LSH) technique on the MapReduce framework. The algorithm has been designed and implemented to detect similar articles using large Wikipedia dumps. Experimental results appear to support that our method is able to efficiently and effectively detect similar articles across Wikipedia.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: [Information Search and Retrieval]

General Terms

Algorithms, Design, MapReduce

1. INTRODUCTION

The online encyclopedia Wikipedia is a multilingual, free resource with its 26 million articles. All these articles, with over 4.2 million in English alone, are written collaboratively by volunteer contributors around the world. Most articles can be edited by anyone with internet access, thus there are currently approximately 100,000 active contributors. In light of these facts, it is clear that the unprecedented example of such a large-scale collaboration effort may result in duplicate or near duplicate articles at various granular levels, such as sentence-level or multi-gram granularity [1]

Furthermore, Wikipedia has a history of generating controversy over editorial quality and factual correctness. Although some studies have found Wikipedia's accuracy to rival that of traditional encyclopedias, other studies have found numerous factual errors. Since Wikipedia may be edited anonymously, information is often freely copied between web sources and even between Wikipedia articles. Additionally, many articles in Wikipedia lack citations. There is little mechanism for detecting and correcting such errors. One potential source of error on Wikipedia is factual information that varies over time and is not updated. Wilkinson found that the distribution of article edits has a long tail, meaning that a small number of articles have many edits while most articles have few, and that the number of edits is directly related to article quality. Articles with low edits and low editorial attention, are less likely to be updated when new information is available [10].

As a step toward identifying factual errors, we investigated the problem of finding near duplicate sentences on Wikipedia. Such near duplicate sentences may reflect factual information that has been updated in one article and not another, or merely inconsistencies that should be flagged and addressed.

In order to come up with a novel algorithm that would allow us to detect near-duplicate sentences within Wikipedia articles, we decided that we needed to answer the following questions, first:

- What is the most appropriate LSH technique to apply? Edit distance seems like a reasonable metric for near duplicate sentences, however LSH does not map naturally to the edit distance metric.
- Compare different sentence similarity techniques for effectiveness.
- What is the appropriate granularity for analysis? Is comparing multiple consecutive sentences more effective than comparing single sentences? Similarly, what is the best shingle size?
- How well do techniques for detecting near duplicate documents work when applied at the sentence level?

*This effort is part of the CMSC828G, Data Intensive Computing with MapReduce, class project.

- Do near duplicate sentences correspond to factual errors? Can these errors be automatically detected and/or corrections suggested?

Along with these questions, we also considered the scalability and efficiency issues due to the large size of Wikipedia. To address these issues, we used the MapReduce computing framework and LSH techniques to group sentences with the same signatures and identify near duplicates.

The rest of this paper is organized as follows: In Section 2, we present related work, in Section 3, we explain the algorithm utilizing locality sensitive hashing, in Section 4, we discuss parameter tuning for Minhash. In Section 5, we present the working application within the MapReduce framework. In Section 6, we discuss our experiments with various Wikipedia datasets. The final section contains concluding remarks and future work.

2. RELATED WORK

A number of techniques have been implemented to identify similarity between documents in the literature. These techniques usually differ in terms of corpus of interest, signature representing the document, feature-set determined by the system, and the end goal. In addition to these differences, one can also consider the efficiency and scalability of the system, overall.

In this section, we will go over some of theses techniques offered in the literature. Additionally, we will highlight our novel contribution.

Broder et al. introduced shingling to compute document similarity and containment. They also presented methods for using a subset of shingles while still allowing similarity and containment computations [3].

Seo et al. defined a general framework for text reuse detection. They introduced Discrete Cosine Transform (DCT) fingerprinting for general or local text reuse detection with high accuracy and efficiency [9].

Zhang et al. presented partial duplicate detection algorithm using sequence matching that transforms partial-duplicate detection task into three MapReduce jobs: 1.Indexing, 2.Sequence Duplicate Detection, 3.Sequence Matching [11].

Hajishirzi et al. introduced an adaptive near-duplicate detection (ANDD) method by extending term-weighting framework to learn k-gram vectors. Provided that each document was represented by an informative real k-gram vector, similarity measures were computed to come up with predicted values of near-duplicates [5].

Lin described three MapReduce algorithms for computing pairwise similarity on document collections [6].

1. Based on brute force.
2. Treated as large-scale ad hoc retrieval.
3. Based on Cartesian product of postings lists.

Manku et al. used Charikar's fingerprinting technique for developing near-duplicate detection system. They presented an algorithmic technique for identifying existing f-bit fingerprints that differ from a given fingerprint in at most k bit-positions, for small k. They demonstrated that their system is useful not only for batch but also for online processing [7].

Bendersky et al. examined two information flow representations: the timeline and the link graph. They proposed several simple unsupervised techniques for timeline construction link graph analysis [2].

In order to handle large Wikipedia dumps with the order of tens or hundreds of gigabytes and even a few terabytes, we used MapReduce framework in this work, that was introduced by Dean and Ghemawat [4]. The implementation is able to stream the compressed bz2 dumps and feed into mappers and reducers for further processing, which will be detailed in the following sections.

3. ALGORITHM UTILIZING LOCALITY SENSITIVE HASHING

In order to measure document similarity we use a common LSH technique known as MinHash []. A minhash signature on a text document is calculated using a parametrized family of hash functions F_i, $1 \leq i \leq N$. In our case, we denote "documents" to mean sentences found within the content of Wikipedia articles. Each sentence is broken up into n-gram "shingles". For each shingle set S we calculate the set $\{min_{s \in S}(F_i(s)\}$ of minimum hashes over the hash family. The signature on a document d is represented as a vector of K minhashes, chosen from the set $\{min_{s \in S}(F_i(s)\}$ of minimum hashes. In order to minimize false negatives we apply a technique sometimes known as "banding" [8] where multiple signatures are produced for each input document.

3.1 Design

Many signature techniques such as LSH are embarrassingly parallel. In our project, we designed our algorithm around the MapReduce framework.

DESCRIBE MAPREDUCE.

Because the minhash technique is parametrized by several variables, including shingle length, signature size, and number of signatures to be calculated per document, these values can also be considered as inputs for the MapReduce algorithm.

3.2 Implementation Details

Our implementation is built on top of the Apache Hadoop MapReduce framework, using utilities from the Cloud9 library (`https://github.com/lintool/Cloud9`) for parsing Wikipedia page text from the Wikipedia XML dump format. Sentences are parsed out over each document using a regular expression. Sentences that are too large or too small are discarded. The family of hash functions is implemented using a "Multiply Shift" CORRECT CITATION GOES HERE hashing scheme and is generated from a random seed using the Java Random class. The hash family is also parametrized by hash output key size and this value can be configured to affect the size of the output signatures.

4. PARAMETER TUNING

Algorithm 1 Minhash MapReduce Pseudocode

```
function INITIALIZE
    F ← hash family;
    L ← shingle length;
    K ← vector length;
    N ← number of signatures to emit;
end function
function MAP(docid d, wikipage p)
    sentencect ← 0
    while s ← nextSentence(p) do
        shingles ← shingleSet(s,L);
        minhashes = new List(|F|);
        for i ← 1...|F| do
            minhashes[i] ← ∞;
        end for
        for g ∈ shingles do
            for i ← 1...|F| do
                minhashes[i] ← min(F_i(g),minhashes[i]);
            end for
        end for
        for i ← 1...N do
            sig = select(K, minhahses);
            emit(sig,(docid,sentencect));
        end for
        sentencect++;
    end while
end function
function REDUCE(signature sig, sentenceids S)
    if |S| > 1 then
        emit(sig, S);
    end if
end function
```

As mentioned above the minhash algorithm is parametrized by several values, including shingle length, length of hash vectors, number of hash bands, and hash output size. Each of these parameters has an impact on the error rates in the minhash output.

4.1 Jaccard Similarity and Edit Distance

Jaccard similarity is a measure of set similarity while our stated goal is to find sentences in Wikipedia that have been copied from one article to another and possibly edited. A more natural measure to use for this application is edit distance. There is no known locality sensitive hashing family for edit distance .

In order to estimate Jaccard similarity of sentences X and Y in terms of edit distance we assume that X and Y are both of length N and that their shingle sets are of maximum size (i.e. all shingles are unique). Also assume that X and Y have edit distance e. We limit type of edits to changing characters in place. Additionally, to simplify our calculations, we assume that edits are localized so that no shingle overlaps more than one edit.

(WLOG) Let S be the set of shingles for X and let S_o be the set of shingles of X that overlap edits. The maximum number of unique shingles for a sentence of length N is $N - L + 1 = |S|$. Also, we can see that $e \leq |S_o| \leq e + L - 1$. Since edits at the start or end of the sentence will overlap with fewer shingles than edit mid-sentence.

Now we can define $|X \cup Y|$ and $|X \cap Y|$ in terms S and S_o as follows:

$$\begin{aligned} |X \cap Y| &= |S| - |S_o| \\ |X \cup Y| &= 2 * |S_o| + |X \cap Y| \\ &= |S_o| + |S| \end{aligned}$$

Plugging in the values above for $|S_o|$ and $|S|$ we get:

$$\begin{aligned} E + N - L + 1 \leq \quad |X \cup Y| \quad &\leq E + N \\ N - L + 1 - E \geq \quad |X \cap Y| \quad &\geq N - 2L - E + 2 \end{aligned}$$

$$\frac{N - L - E + 1}{N - L + e + 1} >= \quad J(X,Y) \quad >= \frac{N - 2L - E + 2}{N + E}$$

Note that, at least under this set of assumptions, the Jaccard similarity is always less than the "edit similarity" (1 - normalized edit distance):

$$\begin{aligned} J(X,Y) \quad <= \quad & \frac{N - E}{E + N - L + 1} - \frac{L - 1}{E + N - L + 1} \\ <= \quad & \frac{N - E}{N} \\ = \quad & 1 - \frac{E}{N} \end{aligned}$$

Finally we can divide through the equations for J(X,Y) by

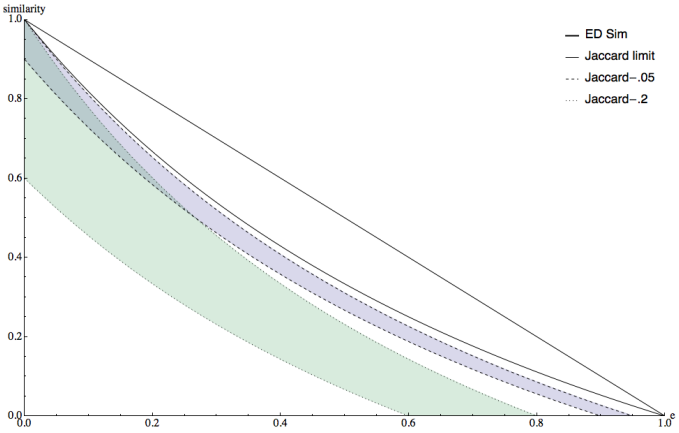

Figure 1: Jaccard Similarity vs. normalized edit distance.

N, which gives us:

$$\frac{1-l-e}{1-l+e} >= J(X,Y) >= \frac{1-2l-e}{1+e}$$

where l is relative shingle size and e is normalized edit distance.

Note that as relative shingle size goes to 0 there is a nice convergence for Jaccard similarity:

$$\lim_{l->0} J(X,Y) = \frac{1-e}{1+e}$$

As we would expect, Jaccard similarity decreases as edit distance increases, but it also decreases as shingle length increases. Figure 1 shows Jaccard Similarity plotted against edit distance for $l = .05$ and $l = .2$. At $l = .05$ and $e = .2$ our estimate for Jaccard similarity has already fallen to the range [0.58,0.65]. Clearly setting l too high can increase the false positive rate, but setting l too low will increase the false positive rate, since it will increase the size of the intersection of the shingle sets.

4.2 False positives

In order to estimate the affect of shingle length on false positive rate we ran a number of experiments for a 1 G collection of wikipedia documents. For fixed $N = 10$ we ran experiments for three values of $K = \{8, 9, 10\}$ and $5 \leq L \leq 15$. Setting a normalized edit distance threshold of $e = .25$, we calculated the total number of "good" ($e <= .25$) and "bad" ($e > .25$) (unique) sentence pairs as well as the false positive rate (# bad pairs divided by the total number of sentences). Results are presented in Figure 2.

We can see from the graphs in Figure 2 that all three statistics hit a low at shingle length 12 or 13. This means that in order to achieve the lowest false positive rate we should use parameter settings around $N = 10, K = 10$ and $L = 12$ or $L = 13$. However the number of *good* pairs also hits a minimum at $L = 12$, which means that while overall the output is more precise, we are also missing output pairs.

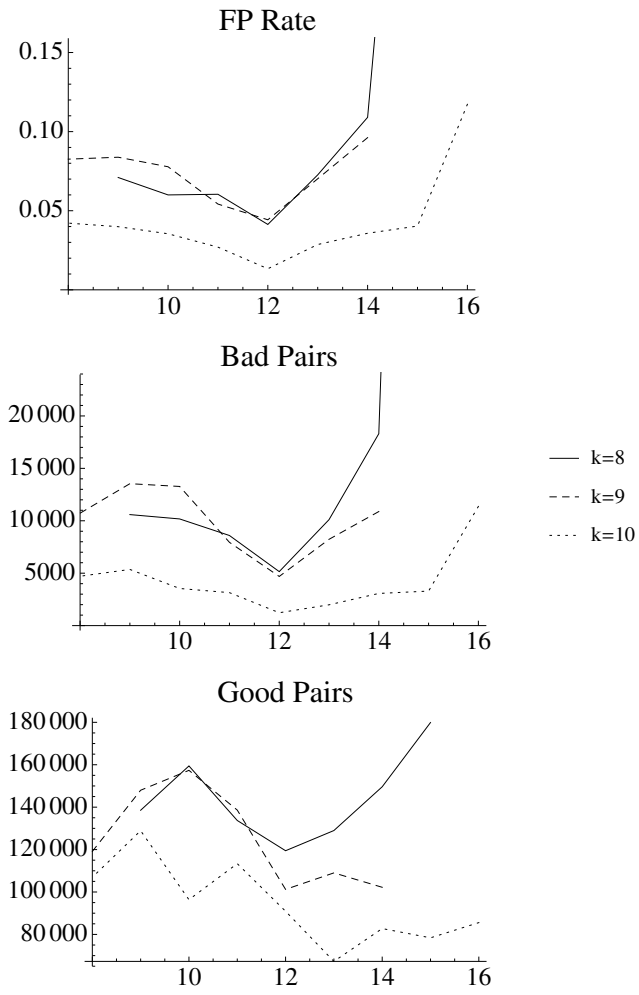

Figure 2: False Positive Rate, Good Count, Bad Count for $N = 10, K \in \{8, 9, 10\}, 5 \leq L \leq 15$

Another feature of the graphs in Figure 2 that stands out is that while the false positive rate and total number of bad pairs bottoms out at around $L = 12$, for $L > 12$ the output quality actually decreases. This is counter to our intuition that output quality increases as the length of the shingle increases. One would expect larger shingles to correspond to closer edit distance scores, since larger shingles correspond to longer consecutive sequences of identical text. One explanation for these counter-intuitive results could be limitations of the underlying hash family. If longer shingle lengths lead to greater hash collisions this could explain the increased numbers of bad pairs. Improving the hash function to reduce collisions could improve results.

Since minimizing the false positive rate also means minimizing reducing good pairs, another option is to let parameters to maximize good output and use edit distance as a secondary filter. With this approach our experimental results suggest that the best parameter combination would be $N = 10, K = 8, L = 10$.

5. WORKING APPLICATION

While the Minhash signature code provides the raw similarity scores, it is one part of a larger framework composed of several pieces, described below. The input to the process is an XML dump format. (See `http://en.wikipedia.org/wiki/Wikipedia:Database_download` for details.) The output of the process is a list of sentence clusters containing the original article title and sentence text. Decomposing the sentence clustering into several pieces allows us to minimize the amount of raw text that gets passed between mappers and reducers, thus improving scalability.

The process can be summarized as follows:

1. **Preprocess Wikipedia Pages** - Using a utility from the Cloud9 library, preprocess the raw Wikipedia dump into a SequenceFile format. Working with SequenceFiles gives the option to use block or record compressed input compatible with the MapReduce framework.
2. **Minhash Wikipedia Sentences** - Reads processed Wikipedia input, and uses a regular expression to match sentences in the article text. Outputs a file of signature "buckets" organized by signature. Each output sentence is represented by a (docid,sentenceid) pair.
3. **Dedup Sentence Buckets** - Because multiple sentences may occur in each bucket, a second MapReduce job is used to map from buckets to raw sentence pairs, removing duplicates.
4. **Compute Sentence Clusters** - Performs a local (i.e., non-MapReduce) connected component analysis on the dedup-ed pairs to form sentence clusters.
5. **Get Sentence Text** - Maps clustered sentences (represented by (docid,sentenceid) pairs) back to sentence text by streaming over the Wikipedia input a second time and recovering the corresponding sentences by ID.
6. **Edit Distance Filter** - *(optional)* For each output cluster and filter all pairs within the cluster by desired edit distance threshold.

5.1 Limitations to Our Approach

In addition to the aforementioned hash issue there are other limitations in our implementation and improvements that could be made.

Data cleanliness - The Cloud9 utilities that process Wikipedia code do some preprocessing of non-text elements from Wikipedia pages. However there is still improvements that could be made. Section headings can confuse our regular expression mapping, causing pieces of extraneous text to be appended to sentences. Parsing page text with a better understanding of how Wikipedia articles are structured could improve our input data. Additionally, keeping track of the order in which sentences occur on a page could improve analysis. For example, references and external links are typically found at the bottom of a Wikipeda page. More carefully filtering text based on sentence depth could improve results.

Sentence matching - Since Wikipedia data is not chunked by sentence, we rely on regular expression matching to break up input text into sentences. Since sentences have recursive structure, our regular expression can lead to stack overflow exceptions from the underlying recursive pattern match search. Setting bounds on the sentence structure nesting could provide more reliable results. The encyclopedic style of Wikipedia may lend itself to simpler sentence structures that don't require a general purpose sentence matching algorithm.

Large clusters - Since we rely on a connected component analysis to compute the final sentence clusters, our large clusters can be a bottleneck. Unfortunately, the Wikipedia data does contain large clusters of similar sentences (see the discussion of templatification below). Since our initial experimental results suggest that more interesting clusters (for example, sentence clusters representing copied data with factual errors) are much smaller (on the order of 2 or 3 sentences per cluster), we set a limit on maximum bucket size and only consider the first n sentences in any bucket up to that limit. Setting a limit on cluster size helps ensure more predictable runtime and discards large portions of relatively uninteresting clusters, making output data more manageable.

Non-determinism - Our current implementation of Minhash suffers from a yet unidentified source of non-determinism. This affects the number of output records produced from the Minhashing stage of our implementation from one run to the next and has unknown effects on output quality.

6. EXPERIMENTS AND DISCUSSION

We performed several analyses to provide more insight into the occurence of duplicate sentences. In particular, we were interested in detecting similar sentences that arose from copying text from one article to another since such instances would represent content that is either redundant, or content that has diverged in the sense that either the original or the copy was edited for clarification of factual correctness and the other was not. Such instances give insight into the Wikipedia editing process, and also represent an opportunity to improve article quality on Wikipedia by identifying places where further editing or article merging may be needed.

All of our experiments were run against an XML dump of current English-language Wikipedia (enwiki) articles (as of 4/2013) obtained from dumps.wikimedia.org. The enwiki dump is approximately 40G in size and contains 4,113,785 articles (after excluding certain non-article page types). Running the first four steps of our application (up to but not including the final edit distance filtering pass) against the enwiki dump on a shared cluster of 16 nodes with (15 GB RAM and 4 cores each) took ≈ 2 hours.

After minhashing and deduplification, we found a total of 9,465,881 similar article/sentence pairs. After clustering these pairs, the total number of similar sentence clusters was 415,936, including 1,792,748 unique article/sentence pairs over 759,297 article titles. The total number of unique sentences over all cluster was 1,305,359.

Cluster sizes ranged from 2 to 28,008. (See Figure 3 for a breakdown of counts by cluster size.) Most sentence clusters are small, but the distribution of cluster sizes exhibits a long tail. As a result, while 99% of our clusters have size below or equal to 30, about 1/3 of the output sentence/article pairs fall into clusters with size greater than 30. One explanation

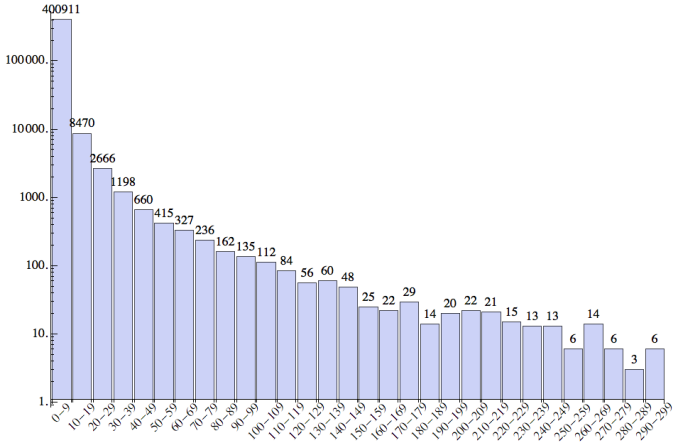

Figure 3: Histogram of Cluster Sizes (for cluster size < 300, log scale).

for this distribution of sentences is the phenomenon we call "templatification," which we describe in more detail below.

6.1 Similar Sentence Types

Manual inspection of cluster output reveals several different similar sentence types. These types can be summarized as follows:

- Templatification
- References
- Direct Copying
- Direct Copying with Copy Editing
- Direct Copying with Factual Drift

Templatification occurs when one sentence is an exact duplicate of another sentence, but the main topic has been changed by only a few pieces of key information having been modified. For example, it could be said that the following two sentences follow a particular *template*.

- Bush had an approval rating of 22% by the end of his term in 2008.
- Obama had an approval rating of 56% by the end of his term in 2012.

From the perspective of editing/writing Wikipedia articles, this is a strong indicator that the author copied the original sentence and modified the relevant information to pertain to a different topic. What remains shared between both sentences though is the inherent structure.

Direct Copying occurs when the same sentence has been copied into another (or possibly the same) article. This can happen for many reasons, including articles that deal with similar subjects, or articles that are subtopics of other topics. (SOME EXAMPLES?). Similarly, *Direct Copying with Copy Editing* occurs when the same sentence has been copied into another article and then one or both sentences has been edited for style or clarity. For example:

Table 1: Sentence Breakdown Counts

Not Similar	3
Factual Drift	65
Template	52
Reference	23
Copy Edit	214
Other	12
Identical	455

- Bush had an approval rating of 22% by the end of his term in 2008.
- Bush, a US President, had an approval rating of 22% by the end of his term in 2008.

Factual drift most notably occurs when the same sentence has been copied and a piece of factual information about the *same* topic has changed. For example, factual drift can be seen in the following two sentences.

- Obama had an approval rating of 56% by the end of his term in 2012.
- Obama had an approval rating of 46% by the end of his term in 2012.

Although factual drift is similar to copy editing and may occur along with copy editing, we distinguish the two cases since factual errors are more severe than style issues.

References refers to citations, typically occurring at the end of a Wikipedia article. Since Wikipedia does not adhere to one single citation style, the same work may be referenced on multiple pages in different styles. Similarly different portions of the same work may be referenced in different articles. Although citation errors might be harder to distinguish from factual errors, errors in citation are arguably no less important. (It would also be interesting to compare whether articles that share similar sentences also share similar citations.)

- Neotropical Ichthyology 11 (1): 73-80.
- Neotropical Ichthyology 10 (2): 245-253.

6.1.1 *Break down of sentence types*

By manually counting a small sample of 824 output clusters, we came up with a breakdown of sentence types in Table 1.

6.2 Sentence Similarity as a Measure of Article Similarity

Based on our preliminary analysis, templatification tends to occur most often between articles that maintain large lists of information related to the same general sub-theme of different topics. For example, of the top 10 pairs of articles with the highest similarity count, one article pair was the articles titled "List of birds of Zambia" and "List of birds of Tanzania". With both countries sharing the same geographic region, it is not unexpected that their respective wikipedia articles on the countrys' bird species would be highly comparable. For clarity purposes, we define *similarity count* as simply the total number of sentences between two articles that were found to share the same hash signature. Using the output of the minhashing routine, we performed

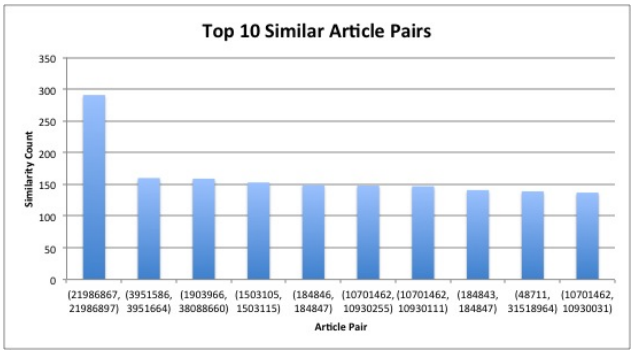

Figure 4: Top 10 Similarity Counts

Table 2: Top 10 Article Pair Titles

Article ID	Title
21986867	2000 New Year Honours
21986897	1999 New Year Honours
3951586	Instruments of the United Kingdom, 1994
3951664	Instruments of the United Kingdom, 1995
1903966	Management of baldness
38088660	Management of androgenic alopecia
1503105	Jayne Torvill
1503115	Christopher Dean
184843	Henry VI, Part 1
184846	Henry VI, Part 2
184847	Henry VI, Part 3
10701462	List of birds of Zambia
10930255	List of birds of Zimbabwe
10930111	List of birds of Mozambique
48711	Oregon Trail
31518964	History of the Oregon Trail

a pairwise similarity count across all articles that shared similar sentences. Overall, we found ≈ 6500 wikipedia article pairs that had a similarity count greater than 50.

Looking at the distribution of cluster sizes in Figure 3, we can see the expected behavior of decreasing number of clusters of size k as k grows large. What is unusual though is the number of relatively large clusters in the size range of $[25, 65]$. Based on the fact that we found ≈ 6500 article pairs with a similarity count greater than 50, this would indicate that there are collections of articles that have a large number of near-duplicate sentences in common. As can be seen in the Top 10 similarity counts, there can even be chains of articles that share large portions of their content with each other (a chain of 3 is demonstrated in Figure 4).

7. CONCLUSION AND FUTURE WORK

This paper presents our work on near-duplicate detection of Wikipedia articles at the sentence-granularity level using MinHash, a LSH technique. Our novel MapReduce algorithm tackles the problem by transforming it into more manageable pieces where each piece is handled in parallel. Our unique contribution includes empirical analysis of signatures belonging to Wikipedia articles at the sentence level. Experimental results verify that the proposed method is able to effectively and efficiently detect similar articles.

In the future, we would like to investigate revision histories of these articles, correlate them with their timestamps, and better relate similarities based on temporal dimensions.

8. ACKNOWLEDGMENTS

We would especially like to thank Dr. Jimmy Lin for his advise and directions.

9. REFERENCES

[1] Wikipedia. `https://en.wikipedia.org/wiki/Wikipedia`, 2013.

[2] M. Bendersky and W. B. Croft. Finding text reuse on the web. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 262–271, 2009.

[3] A. Z. Broder. On the resemblance and containment of documents. In *In Compression and Complexity of Sequences (SEQUENCESâĂŹ97)*, pages 21–29, 1997.

[4] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[5] H. Hajishirzi, W. tau Yih, and A. Kolcz. Adaptive near-duplicate detection via similarity learning. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 419–426, 2010.

[6] J. Lin. Brute force and indexed approaches to pairwise document similarity comparisons with mapreduce. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 155–162, 2009.

[7] G. S. Manku, A. Jain, and A. D. Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th international conference on World Wide Web*, pages 141–150, 2007.

[8] A. Rajaraman, J. Leskovec, and J. D. Ullman. *Mining of Massive Datasets*. January 2011.

[9] J. Seo and B. W. Croft. Local text reuse detection. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 571–578, 2008.

[10] D. M. Wilkinson and B. A. Huberman. Assessing the value of cooperation in wikipedia. *First Monday*, 12(4), April 2007.

[11] Q. Zhang, Y. Zhang, H. Yu, and X. Huang. Efficient partial-duplicate detection based on sequence matching. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 675–682, 2010.