Efficient Near Duplicate Document Detection for Specialized Corpora

by

Shreyes Seshasai

B.S., Massachusetts Institute of Technology (2008)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the Massachusetts Institute of Technology

June 2009

Author .....
Department of Electrical Engineering and Computer Science
May 22, 2009

Certified by ..
David Spencer, Senior Software Engineer
VI-A Company Thesis Supervisor

Certified by .....
Regina Barzilay, Associate Professor
MIT Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Efficient Near Duplicate Document Detection for Specialized Corpora

by

Shreyes Seshasai

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2009, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Knowledge of near duplicate documents can be adventagous to search engines, even those that only cover a small enterprise or specialized corpus. In this thesis, we investigate improvements to simhash, a signature-based method which can be used to efficiently detect near duplicate documents. We implement simhash in its original form, and demonstrate its effectiveness on a small corpus of newspaper articles, and improve its accuracy through utilizing external metadata and altering its feature selection approach. We also demonstrate the fragility of simhash towards changes in the weighting of features by applying novel changes to the weights. As motivation for performing this near duplicate detection, we discuss the impact it can have on search engines.

VI-A Company Thesis Supervisor: David Spencer
Title: Senior Software Engineer

MIT Thesis Supervisor: Regina Barzilay
Title: Associate Professor

Acknowledgments

I would like to acknowledge my family, my friends, my advisors, and my colleagues. Many of these people fit into multiple of those categories, and I am thankful to have all of them in my life.

In particular, I would like to thank Dave Spencer for being a great mentor and friend to me at Google, Regina Barzilay for her support and patience at MIT, and Mildred Dresselhaus for five years of continued and caring advice.

I also would like to thank Nick Semenkovich, Michael McGraw-Herdeg, and Arkajit Dey for their help in optimizing my code, trimming processing time from weeks to hours.

Contents

Chapter 1

Introduction

The purpose of this research is to investigate methods to detect near duplicate documents and their impact on the search process in an enterprise setting, which includes crawling a corpus, indexing the documents, and ranking the results.

This thesis will detail what it means to be considered a "near duplicate" document, and describe how detecting them can be beneficial. After describing methods that are currently in use for such detection, we implement simhash [24] to demonstrate its effectiveness on a small, specialized corpus, and offer improvements to its algorithm.

1.1 Motivation

The driving motivation behind these near duplicate algorithms is to quickly and efficiently determine which documents in a large set are similar to each other. Knowing this information can lead to improvements in a variety of ways, depending on the application for which it is being used. Here we consider the case of a search engine working to serve a small corpus of documents.

1.1.1 The Search Pipeline

Being able to detect near duplicate documents can improve the performance of a search engine being used to retrieve documents from the specialized corpus.

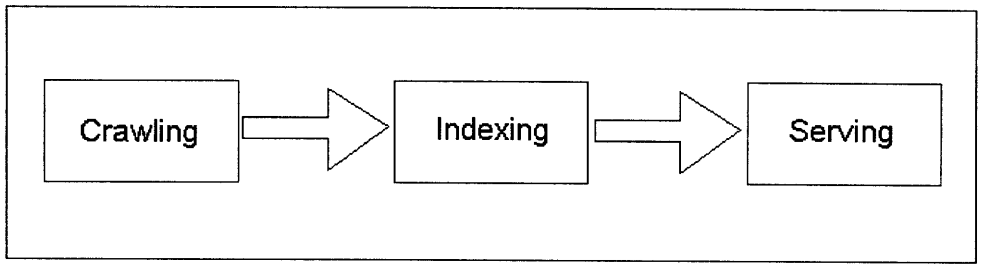

Figure 1-1: A generic search pipeline: crawling includes discovery and downloading, indexing involves parsing the text, and serving retrieves documents as answers to user queries.

The search process can be described as a pipeline that is split into three basic stages: crawling, indexing, and query processing.

Crawling involves discovering new documents and downloading them so their contents are available to process. Updating the contents of previously discovered documents is also important, to maintain the freshness of the corpus. Because there are usually much more documents available to crawl than there is time to process them, deciding which documents to download is important and affects the quality of your search engine. Documents that are left to grow old become less relevant to the user.

When crawling documents, we can adjust the frequency with which it crawls near duplicate documents to optimize for freshness. Suppose each document the crawler knows about is given a priority level, which determines the order that each document is crawled. If a document is known to be a near duplicate of another one, we can decrease its priority, because we know that other documents in the index exist with similar content. This way, documents with unique content are more likely to be downloaded first, so we will cover a larger portion of the total information space of the corpus.

Indexing involves parsing the words from the document and adding them to a reverse look-up table that stores a mapping between word and the document in which the word can be found. Taking a simplistic view on this process, there are two costs: the processor cost in doing the initial text processing, and the memory storage cost of storing the information in the index. Once we download a document, we can detect whether it is a near duplicate of a document already in the index. If it is, we can decide

to discard the document immediately, and not pass it through the rest of the pipeline, which will save on both costs. If there is a concern that discarding the document will result in losing important information, instead of discarding it completely, it can be given a lower priority for indexing, similar to above how documents can be given a lower priority for downloading.

The size of the index will thus be reduced, which is good especially in cases where the size is constrained by the hardware available to the search engine. If space is not an issue, a smaller size still has practical benefits, as searching through a smaller index can be done quicker. Work has been done by Zhang and Suel[31] to create a framework for searching redundant corpora which would help significantly reduce the index size.

Query processing is controlled by the front-end interface that the user sees, and it too can be improved by knowledge about near duplicate documents. Documents which are near duplicates may appear close together in search results, by nature of how the engine retrieves information from the index. However, these duplicates provide little benefit to the user, if the user is not looking for a document in that particular cluster of near duplicates. For example, if a user only sees ten results on the top page, and eight of them are near duplicates of the same document, in effect the user is in effect only seeing three unique documents. The chances the user finds what he is looking for are reduced.

In addition to improving the diversity of results during query processing, the front end interface can also leverage near duplicate information in creating creative display formats. It perhaps can provide the user the ability to narrow in and filter the result set to only those in the same near duplicate cluster. Or perhaps list these near duplicates in a side box sorted by a metric besides relevancy, such as time or length.

The ranking algorithm also may benefit from knowing which other documents are near duplicates to those in a search result. Perhaps if there is near duplicate of much higher quality than the document that would normally be returned, it would return the near duplicate instead. While these different benefits to the front-end are interesting, they will not be discussed further in this thesis, but can be considered as

potential additional benefit should near duplicate information be known.

1.2 Contributions

We make the following four contributions in this thesis:

- We implement simhash, an algorithm for detection near duplicate documents, and demonstrate its effectiveness on a small, specialized corpus, unlike what it has been used on before.
- We improve the performance of simhash through utilizing external metadata in feature selection.
- We demonstrate the fragility of simhash towards changes in the weighting of features, by applying novel changes to the weights.
- We "solve" the problem of inefficient Hamming distance calculation by demonstrating how it may be unnecessary for search applications.

1.3 Outline

Chapter 2 describes the current state in the field of near duplicate detection, and provides some background on the topic. Chapter 3 discusses the original simhash algorithm which we use as a foundation before later improvements. Chapters 4 and 5 describe specific improvements that we make to the algorithm, first through improving feature selection and then by improving the weighting scheme used. Chapter 6 describes the experimental runs done on different corpora, trying each of the potential improvements. Chapter 7 discusses what future work is left to be done, and finally Chapter 8 concludes the thesis.

Chapter 2

Background and Related Work

When comparing two documents, it is easy to detect if they are exactly identical, using checksums or other comparison techniques. Detecting if a document varies slightly, in the case of near duplicates, is more difficult.

There are several approaches to detecting near duplicate documents that have been proposed and tested on the web which can be implemented and explored further. Here we look at how a few of these are accomplished, and discuss their relevance to the enterprise domain.

2.1 Enterprise Corpora

Enterprise corpora differ from regular documents on the web in their size, quality, and (lack of) diversity.

Collections of documents within a company or for a specialized purpose are often

Table 2.1: Relevant properties of most enterprise corpora, or specialized corpora in general.

Properties of Specialized Corpora
Small size (relative to web documents)
Cleaner document content, including less spam
Common page structure in segments
Available metadata or associated content

smaller than the collection of documents on the web. Thus, for processing, one can consider sizes on the order of millions of documents, instead of billions. This allows most of the computation to be done on a single computer, eliminating the need to worry about sharding or splitting resources.

Although it's impossible to measure empirically, documents in specialized corpora are also cleaner than general web documents. There is much less spam, if any at all, in controlled corpora. Instead, it is more likely that documents are simply outdated than intentionally garbage. Since the material may only be available internally, there may be less incentive to keep it up to date, because it is not open to the scrutiny of the web.

Documents within the same corpora can also share similar features, such as page structure. If many documents share the same boiler plate template, this template information can be removed from the page to leave only the most relevant information for similarity detection. Consider the case of a school's website. Each department may have its own template for information, including a person directory that contains structured information in relevant specific parts.

2.2 Definition of Near Duplicate

We now consider exactly what it means for two documents to be near duplicates of each other. There are many different definitions that others have used, each different depending on the particular use case and motivation. In a general sense, "near duplicates" are documents that differ only slightly in content.

Definition 1

In the extreme case, near duplicates are those that appear identical to the user when viewed in a browser, despite the actual code being different. This case arises when there are non-visible changes in a document (such as HTML comments) or a common document that multiple other documents can map to (such as a print style that eliminates most of the formatting).

Definition 2

A slightly relaxed definition from this would be where the relevant content of the page stays identical, but the less important information around it changes. For example, if a page has a visitor counter, a last-visited date, or even the current date, this information is not important for content purposes. If the main body of the document has not changed, then the two should be considered a near duplicate.

This case can also involve page templates, sometimes referred to as a boiler plates. Let us imagine that there are a group of documents with the same header and sidebar. If one of the links on the sidebar changes, that change is not important to the main content of the page. Similarly, there is often dynamic content in the outer regions of the page, which isn't relevant to its main content. For example, on a news site, there may be information about articles that are the most popular, or most commented on, which changes frequently but again is not related to the important content of the page. Dynamic content in the form of advertisements can also create this problem.

Definition 3

Even more relaxed, changes in the main content of a document can also be considered near duplicates. These can be classified into one of the following cases: additions, deletions, transpositions, or combinations of the above.

While these categorizations are mostly self-explanatory, the most common case may be the last, where several changes are made to a document without a concise way to describe them. If you are looking for a specific case such as an addition to the end of the document, the complexity of the parsing is simple. But handling minor edits throughout the body of the text creates more questions. How many edits are required until the revision is considered significant enough that the documents are no longer considered near-duplicates?

We can begin my measuring this in terms of percentage change. Perhaps if less than five percent of the content changes, it should be considered a near-duplicate. But again, this arbitrary measure does not capture well the significance of the change. What if this five percent is actually minor edits to reword a section or fix grammar,

which does not impact the overall content of the page? Compare this to a much smaller edit, to the title of the page, which would be much more significant despite being much smaller in length.

The main conclusion we can draw from this is that the nature of the change is as important as the length of it. If there was a way to quantify how important a particular section is to the entire document, and then quantify the importance of the overall change, then we may be able to better ascertain whether the documents are near documents or not.

Given this range of possible definitions, this thesis will not focus on one specifically, but rather focus on devising a framework that can be adjusted for each of these situations.

2.3 Overview of Methods

As in many information retrieval tasks, proper feature selection is an important component in determining the success of the algorithm. Different systems have didn't sets of feautres that they use when processing a document. Here we go through the most common few.

2.3.1 Shingling

One method that has been explored with similarity detection in web documents is called shingling, developed by Broder et al. [7]. The procedure is as follows:

Each document is tokenized into words, and each sequence of q consecutive words (q-grams) are fingerprinted together using Rabin's 64-bit fingerprint [25]. Each of these fingerprints is called a shingle. We can write the set of all shingles in document A as S(A). To compute the similarity, or resemblence as Broder calls it, between two documents A and B, we compute the Jaccard similarity between their two sets of shingles. $r(A, B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}$.

Research has been done into various methods to filter the shingles in a document to try to make shingling more efficient and accurate. A simple way to filter shingles

would be to reject ones whose remainder mod p does not equal 0, for some number p. The size of p could then determine the factor by which we can compress our space of shingles, depending on how efficient we want to be.

Hoad and Zobel [18] have done work employing many different techniques for filtering which shingles are accepted. For example, accepting all substrings in a document, or just the first r in the document, or the one with the rarest prefix, or the Kth in every sentence, or just the entire Kth sentence, are a few examples.

Each of these strategies had their own benefits and faults. As Hoad and Zobel found, using the full fingerprint had a high precision and recall as expected, since it characterized the full document. Using just the Kth sentence produced the worst results.

Later we discuss improvements made to this shingling algorithm when we discuss similarity metrics.

2.3.2 Using Full Document Vectors

Instead of using n-grams of the document like Broder, some methods construct full document vectors, which store all of the words in a document. Standard information retrieveal techniques while contructing this vector can be used to remove stop words (common English words), perform stemming (e.g. to remove a plural s), and to calculate a weight for each word's level of importance in the document. This weight is normally the tf-idf score of the word, which will be detailed later.

Once we have this document vector, we can take the cosine similarity of the two vectors to determine how similar two documents are. While this may seem like an appropriate approach, and easily computable, Hoad and Zobel note the downside in [18]. The inner product itself will be larger for large documents, so it will outweight the scores for smaller documents. If we normalize this value by the length of the document, then the bias switches to small documents, which will be given a higher weight.

A third option is to calculate the cosine similarity of the vectors, which normalizes the inner product by the magnitude of the vectors. While this results in a value that

is comparable across different document pairs, it removes the measure of relative strength between the documents. If one vector of weights is consistently larger than the other, then this is an important difference for near duplicate detection. This quality is lost though when normalizing by the magnitudes of the vectors.

Full document vectors will be manipulated and discussed in greater detail in the discussion of I-Match and simhash later.

2.3.3 Using Nearby Documents

The features of a document need not rest within the document itself, especially when in the context of a larger corpus. Pages in a web corpus often link to each other, and both the existence of the link, and the text describing the link, are important features for a document.

Say document A has a link to document B. The HTML in document A might look like `<a href="B.html">anchor text</a>`. The words "'anchor text"' here are important features for B, as they usually provide information like a title or topic for B. A search engine can save this information as metadata for B after it parses A.

The power of this information was demonstrated by Dean and Henzinger [13] when they demonstrated a retrieval algorithm that was able to generate closely related web pages simply by analyzing the link graph information and links on a page. They did nothing with the actual content of the page, but only used which pages linked to and from it.

Their algorithm also contains a step where pages were checked to be duplicates by comparing their common links. If two pages shared at least 95% of their links, then the pages were declared duplicates and merged. As noted in their work, not collapsing the two duplicates caused their results to be distorted, which suggests that link information would be a good feature to use for similarity detection.

In sites with a very sparse link graph, such as within enterprises or specialized corpora, information from link connectivity is not as helpful, as it is possible/likely that two pages share none or only a few links in common. In this case, Haveliwala et al. [16] suggest using the anchor text and anchor window as additional features to

help learn about the other document. They suggest that the words around the link are still informative about the link, and can be weighted and used as features. Their results show that this performs better than the pure link option, but understandably performs worse than if they had used information about the full contents of the document.

2.3.4 Using Salient Terms/Phrases

The earlier shingling approach was done on either all or a subset of random substrings in the document. An alternative approach is to pick out specific phrases of the document, and only store and use those phrases when comparing documents among each other.

Cooper et all [12] use a tool called Textract to extract important phrase, or multi-word segments, from the text and store them in a database. They then compared the number of phrases matched between documents to determine whether the documents were similar or not. While the idea may be promising, they only tested it in on an extremely small (less than 50) set of hand-picked documents, so it is unclear if their performance gain over raw shingling is signifgant.

Hammouda and Kamel [14] also focus on phrase information when considering the problem of clustering near duplicate documents, choosing to solve it by creating their own indexing model to retain the phrase structure of the original text. In the standard document vector approach discussed above, the words are split and stored separately, losing information about location and word order in the process. Hammouda's Document Index Graph stores entire phrases with the goal of eventually using the phrases to cluster documents based on similarity. They use the same Jaccard similary value when comparing phrases as was done earlier when comparing entire documents.

Overall, using a variety of clustering techniques including k-nearest neighbors, hierarchical agglomerative clustering, and single-pass clustering, Hommousa and Kamel show improved quality in the clusters they find using their Document Index Graph.

2.4 Calculating Similarity

After collecting features from the document, we apply a similarity metric to be able to measure how similar documents are to each other. For some of the techniques above, this is trivial. For example, with shingling, the similarity is just measured in terms of matching shingles between the documents. However, there are subtlties and technicques that we can apply to these features to achieve better results.

2.4.1 Min-hash

Thinking back to shingling, saving the entire set of shingles for every document is very inefficient. So instead, Broder et al. figured out a nice way to approximate the Jaccard similarity measure [6].

Given m fingerprinting functions, we hash every shingle with each of the m functions, and then choose the shingle that resulted in the minimum value for that particular hash function. That leaves an m-dimensional vector for each document. When determining the similarity of two documents, we can just intersect the two m-dimensional vectors and see how many shingles the two documents have in common.

To be even more efficient, the m-dimensional vector can be reduced to an n-dimensional vector by concatenating non-overlapping shingles from m and taking the fingerprint again. These are called supershingles. To determine the similarity of two documents, we calculate the size of the intersection of the two n-dimensional vectors, with them being considered near duplicates if they share at least two supershingles.

Based on [17], the values of m and n we should choose for this problem are 84 and 6, which means it would require 48 bytes of storage per document.

Haveliwala et al. applied a similar technique like min-hash, called Locality Sensitive Hashing. It too would approximate the Jaccard similarity of the two sets [15]. Instead of only taking the minimum of the m hashes, LSH concatenates k of the m hashes.

This leads to fewer documents being assigned the same hash, meaning fewer false positives. However, the converse is true, as it would increase the number of documents

that we would miss, or false negatives.

2.4.2 Fingerprints on Doc Vectors

A common practice when dealing with a large set of documents is to create a fingerprint for each document. In the most general sense, a fingerprint is just a lower order representation of the document, and hence of the document vector. For example, one can hash the document using the common sha1 or md5, and their result can be an eight byte representation of the document.

Fingerprinting can be used for different purposes, depending on the fingerprinting function. If the function is a cryptographic hash like sha1 or md5, the purpose is to create fingerprints that have high entropy and are not reversible. For example, let's say we have document d_0, and applying md5 to its text results in the sixteen byte value b_0. If were to take the hash b_i of all documents d_i in a set D, then we would expect all b_i to be evenly distributed over the space of possible hashes.

Now, if we were to change document d_0 by a tiny amount, such as changing one character of text, we would expect it's hash b_0 to be much different than its previous hash.

For near duplicate detection, we want to create the opposite output. The goal is that if the document changes by a little bit, the hash as well would only change by a little bit. In a sense, we want the difference in the hash to be a function of the difference in the document.

Then, once we have the fingerprint of all of the documents, in order to determine if two documents are near duplicates of each other, we can simply compare the fingerprints, and not worry about the full original document.

This approach to do similarity detection was explained by Manku, et al. [24]. The algorithm they used for determining the hash is to break up the document into many pieces, hash each of them individually, and then combine those hashes to form the final answer. They called their hash *simhash*, which will be described in more detail in Chapter 3.

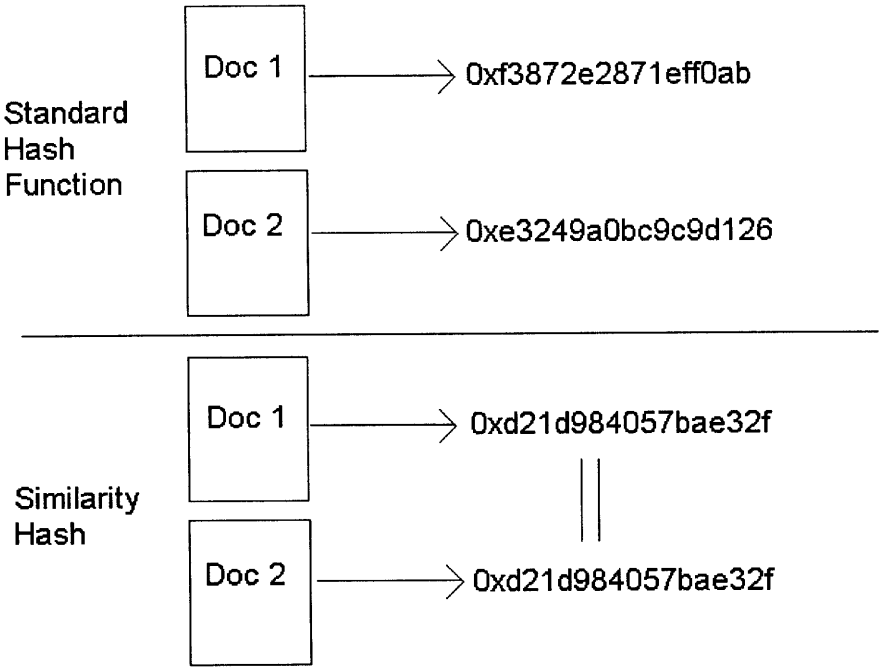

Figure 2-1: Documents 1 and 2 are true near duplicates of each other. Using a standard hash function, their hash values are unrelated. Using simhash, their hash values are identical with Hamming distance 0.

2.4.3 I-Match

I-Match [9] can be viewed as a fingerprinting approach that filters out some words based on their idf value, which is determined by its prevalence in the corpus. An idf value stands for the words inverse document frequency. $idf = log(\frac{N}{n})$, where N is the total number of documents in the corpus, and n is the number of documents in which the word appears.

Words with low idf values are discarded because they are usually common words in the language, so they do little to distinguish the particular article in question. Discarding words with high idf values are also believed to result in a better setup for identifying duplicates [9]. The resulting middle range of words defines the lexicon of acceptable words.

I-Match works by parsing each word in the document, and filtering out words not in this reduced lexicon. The words that remain are sorted in order and added to a SHA1 digest, which results in a single SHA1 digest key per document. If there

is a collision with that key and another document, then those documents are near duplicates of each other.

In the initial experiments performed by Chowdhury et al., I-Match outperformed Broder's shingling (also called DSC), although it was still shown to have problems, especially with word addition or subtraction from the lexicon. A single addition could upset the entire SHA1 digest, so the equality would no longer hold with near duplicate documents.

Kolcz, Chowdhury, and Alspector minimized this problem by introducing a series of randomized lexicons, as opposed to one, which would determine the list of acceptable words [22]. Given K copies of the original lexicon, each one had a random selection of p words removed from it. The I-Match algorithm was then repeated for all of the other K lexicons in addition to the original, resulting in K+1 hashes per document. Two documents are then said to be near duplicates of each other if they share at least one of these K+1 hashes in common.

This improved version of the I-Match algorithm did show improvement over the past version, and did do well in clustering the web and email data presented by Kolcz et al. [22]

2.5 Driving Motivations

While our specified purpose for this enhanced near duplicate document detection is within the context of an enterprise search engine, research has been done in the field for a variety of other purposes. Examples include work to cluster documents in a specific field (which is nearly identical to our target of enterprise corpora, or specialized corpora), detecting mirrors of repositories, clustering results to a query over a corpus, extracting information from pages, and detecting spam in email messages.

2.5.1 Domain Specific Corpora

Work has been done to narrow the scope of near duplicate detection to a specific class of documents, with the goal of revealing characteristics about the files that may be

salient for similarity detection. For example, Conrad and Schriber [11] have detailed a process for created a test collection of news articles relating to law, which rely on taking the result set of actual queries and confirming the near-duplicates through a series of human evaluations. In this particular setting (of law-related news articles), the human experts converged on a definition of nearly identical as documents that overlapped in at least 80% of their content, while not differing in total length by more than $\pm 20\%$.

Given this evaluation, they were able to determine the effect that incorporating new collection statistics have on the determining nearly identical documents [10]. For example, including elements such as a document's published date and length along with the content of the documents improved the accruacy with which they were able to identify near identical documents in the results of test queries. This adds evidence to the claim that the feature set one uses to process documents are important when doing any type of classification, or in this case, clustering.

Another part of Conrad and Schriber's work, but not specific to the legal domain, was the affect of changing the statistics of the corpus while document processing was ongoing. For example, as documents are added to and removed from the corpus, the document frequency, a measure of how many documents a word appears in, will change. This document frequency value is often used in calculating the importance of a word to the document, acting as a factor in its overall weight, as we saw with the idf score. Because the weights are sensetive to changes in the corpus, Conrad suggested freezing the general corpus statistics while processing is going on, as we have done when evaluating the simhash.

The description "domain specific" need not be restricted to topic, but can also include the type of files or repository. Finding duplicates within a database or filesystem has also been an area of study, motivated by similar reasons behind finding similar web pages or documents.

Manber created one such system, *sif*, which found similar documents in a file system [23]. His algorithm worked by extracting substrings from the file of a particular length, either by looking for a particular "anchor" string to begin the substring,

or finding all possible substrings of that length. He then takes the checksum, or fingerprint, of a sample of these substrings, and these act as a representation for the file. If one wants to compare how similar files are to each other, then can then compare how many of these checksums the files have in common. Even if files are similar in as little as 25% of their contents, it is possible that *sif* would report that the files are similar.

2.5.2 Web Mirrors

Detecting web mirrors is a common application for duplicate detection, as they can lead to several problems for search engines. They take up extra room in the index and can cause extra network load on the servers holding the documents. Having some links directed towards the mirror and others directed towards the original also can cause inconsistencies when creating the link graph that is often used to determine page quality.

Bharat et al. studied a variety of ways to detect mirrors in a large collection of web pages [2, 3], in which similarty detection plays an important role. Given a large set of URLs, the algorithm looks for URLs that share common features (such as words in the URL), and adds that as evidence that those hosts are in fact mirrors of each other. It uses a variety of coordinated sampling techniques so that only a few number of urls need to be found to still provide a high likelihood that a pair of hosts are mirrors.

Once it has a pairs of urls that suggest that their hosts may be mirrors, it can compare the content of the pages to determine the level at which the hosts are mirrors. If the documents are exact matches of each other, it adds strong evidence that the hosts are mirrors. If the pages are not identical but similar, by some measure of similarity, it adds smaller evidence that the hosts are mirrors of each other.

The exact similarity metric used in this instance is not as important as the idea that knowing the level of similarity between pages helps make the classification of hosts as mirrors more precise. Deciding if different urls point to the same content is no longer a binary decision. While Barat et al. chose to use the Jaccobian similarity

between the sets of n-grams for each document to measure this similarity between documents, using simhash would have also been a workable solution.

2.5.3 Extracting Data

Near duplicate detection can also "give back" in a sense, as there is valuable information to be gained from knowing which documents are part of the same similarity cluster. Analyzing these documents can reveal information about their page structure or other common elements.

As Arasu and Garcia-Molina demonstrated [1], it is possible to extract template information from a cluster of similar pages by looking for sets of words that the pages have in common. Once we have a model for what the template looks like, we can extract the information that was different at particular locations on the pages, which is the structured information. Examples that this can be used on include sites like Amazon or Ebay, which both follow a strong template.

While the more general near duplicate detection techniques may not be strong enough to cluster pages which share the same template (if the page's other content differs extensively, for example), there are specific techniques developed for detecting pages that share the same structure, or "boiler plate". Joshi et al. created a bag of tree paths model for this purpose [21], which first built up a DOM tree with the pages HTML structure, and then stored each individual path within the tree. After removing inconsequential paths by applying appropriate weights, comparing the paths in two documents will determine the similarity of their structure. According to Joshi's results, this is a sufficient enough measure to cluster together pages with a similar structure.

2.5.4 Spam Detection

Similarity detection is also relevant to spam detection, as often spammers send many copies of identical or similar messages to many people, so properly detecting them can help classify spam vs. ham. Spammers understand that exact matches in emails

are easy to detect, and thus will often change the email slightly, increasing the need for near duplicate based detection.

Kolcz et al. [22] used their improved I-Match algorithm, as described in 2.4.3, on a collection of ham and spam messages, to evaluate the performance of their similarity detection. Querying 10% of their known spam messages against a collection of legitimate and spam messages, their algorithm resulted in no false-positive ham labels, and a recall value that increased from 0.66 to 0.80 as they used more randomized lexicons. This both demonstrated that their method of using randomized lexicons worked, and that spam messages are detectable in the first place, provided you have a good set of examples on which to train (and build your lexicon).

2.5.5 Plagarism

A stricter form of near duplicate detection can also be used for plagarims detection, where someone copies an entire or a portion of a copyrighted work. This is another good example where detecting similar items will be much more beneficial than an exact duplicate, as plagarism is not often an entire body of work. If portions of a copyrighted text were stolen for use elsewhere, we would like the system to be able to detect that only that portion was taken.

Brin in 1995 described a copy protection system called COPS [4], which would parse known copyrighted sentences by chunking them and saving them into a database. Then when evaluating a new document to see if it was copied or not, it would tokenize it by sentence and check each sentence for a match in the database. This would be able to detect if only portion of the document were copied from the source.

Although the field might be moving towards more intrinsic plagarism detection [27], which tries to detect changes in writing style within the document itself as opposed to trying to find a near duplicate document from a repository of known copyrighted works, work still is being done with near duplicates to detect plagarism. For example, Stein and Eissen [26] created a new fuzzy fingerprint for documents which can then be compared for plagarism detection.

2.6 Uses of the Hash

The method that we focus on in this thesis is simhash, the exact algorithm of which will be described in Chapter 3. Once the simhash of all documents in a corpus are available, there are two general ways that can be used. First, to track the progression of a single document over time, and second, to form clusters of similar documents. Section 1.1 above described how these clusters can then be useful.

2.6.1 Comparing the same document over time

Often it is useful to track the evolution of a document over time. Since a change in the simhash of a document is a function of a change in the document contents itself, the larger the simhash changes, the larger the document has changed since its previous version.

Supposed we download a new version of a document. Without using a simhash, we can still find information about how much it has changed since the last time we downloaded it. We can do a plain diff of the entire contents, and know the percentage that has changed. This, however, does not take into account the quality of the change, which was one of the main motivations in using simhash. As mentioned before, if it is a large change percentage wise, but to minor elements of the page, then it's less important than a small change to a major element of the page.

Now using the documents simhash, we can simply compute the Hamming distance between the current and previous simhash, and this will give us an idea of how much the document has changed. If the Hamming distance is 0, then the document is considered not to have changed significantly from the previous version. If you would like to know if there was any change at all, you could always also compute the exact hash of the document (using a plain md5), and compare those values.

The size of the Hamming distance, being proportional to the level of importance of the change of the document, can then be used for a variety of functions. In the example of a search engine, this value can affect the priority with which this document is sent to the indexing stage. Documents which are downloaded that have a higher

change from the current version in the index should be given a higher priority so that the index remains fresher.

Looking at a series of hashes from the past can also affect the long term rate at which the document is checked. For example, if the previous three times a simhash was calculated were all the same, then it should decrease the frequency with which that document is downloaded. It can wait a longer time because it expects the document not to change. However, if the document has changed every time in the last three times it has been hashed, then it should increase the frequency with which it downloads and processes the document.

In terms of storage, using a simhash provides additional benefits. In the specific context of a search engine, it is sometimes expensive to reconstruct the contents of a document once it has been already processed and placed into the index. Time is either lost in reconstructing it, or space is lost in having to store an intact version of the previous copies. Simhashes, however, are only 8 bytes, so storing past versions of the simhash is much more economical.

2.6.2 Clustering documents

Having all of the fingerprints in a corpus can now allow us to cluster these documents by fingerprint. Given the simhash of a document, we can now find all documents whose simhash is within a Hamming distance of 3, and designate a cluster of near duplicates in that way. The difference of 3 was chosen by Manku [24] as an optimal choice for web based documents, so it may vary depending on the domain of documents being compared.

The difficult part here is comparing the fingerprints in an efficient manner. Manku et al. propose a solution to this that tries to optimize for time and space. If the list of fingerprints is sorted, they can first look at the d most significant bits and easily find other fingerprints that match those exactly (since they are adjacent in the sorted list). They then compare the 64-d remaining bits to see if those are off by a Hamming distance ≤ 3. Since they only have to look at the few hashes which match in the first d bits, the amount is small.

In order to cover the entire space, one has to store a second copy of all of the hashes, rotated by d bits and then sorted. Thus, bits [d+1,2d] will match exactly, and the difference of 3 will come from the remaining 64-d bits. In general, this approach will require 64/d sorted lists, so f/d copies of all of the hashes.

Each cluster is keyed by a single document, which is the master document in the cluster. Those with Hamming distance less than 3 from this document are considered part of the cluster. If there was no master document, then there could be inconsistencies on whether to include some documents in the cluster, since the Hamming distance could be 1 with some but 4 with others.

For example, say Document A had a hamming distance of 3 with Document B. Document C had a hamming distance of 1 with Document A, but it was a different bit than the 3 that A and B differed in. This means that the distance between B and C is 4, which means they should not be near duplicates. However, we don't want to reject C from the cluster, because it is very close to A. Also, the order in which documents were added to the cluster would affect it. If A and C were together before B was discovered, then B would be rejected despite being close to A.

By keying a cluster on a url, it eliminates this ambiguity. Deciding which url is the master url in a cluster can be done a number of ways – most recent, highest page quality, shortest page url, etc. The advantages/disadvantages of choosing which method to use will not be discussed here.

Chapter 3

Simhash Algorithm

The simhash algorithm describes a function by which to compute an eight byte hash from any document. It follows the path of Broder's shingling and I-Match as a signature based approach to determining near duplicate documents.

We chose to use simhash here over Broder's algorithm because it was computationally less intensive and required less space, both advantages when working with limited resources of a small search engine. In order to compare two documents, the engine would have to do a set intersection among shingles, which will take much longer than doing a simple comparison of two eight-byte hashes.

Simhash also had demonstrated success on larger web repositories, but moving it to a smaller, specialized corpus has not been done, and could produce interesting results.

First proposed by Charikar [8], the algorithm has several steps.

3.1 Feature Selection

The first step the of the algorithm is to parse the document for features. In a broad sense, feature selection can include all aspects of the document such as raw content (words, bigrams, trigrams), author, title, anchor text, etc. It also can include connectivity information (which pages it links to), or other environment specific traits, such as what domain the document belongs to.

Initially, let's define the features to be take a bag of words approach, in that each feature is simply each individual word in the document. Selecting the appropriate features when processing a document is important, and can often lead to the success or failure of an implementation. The next chapter will cover improvements to this feature selection process.

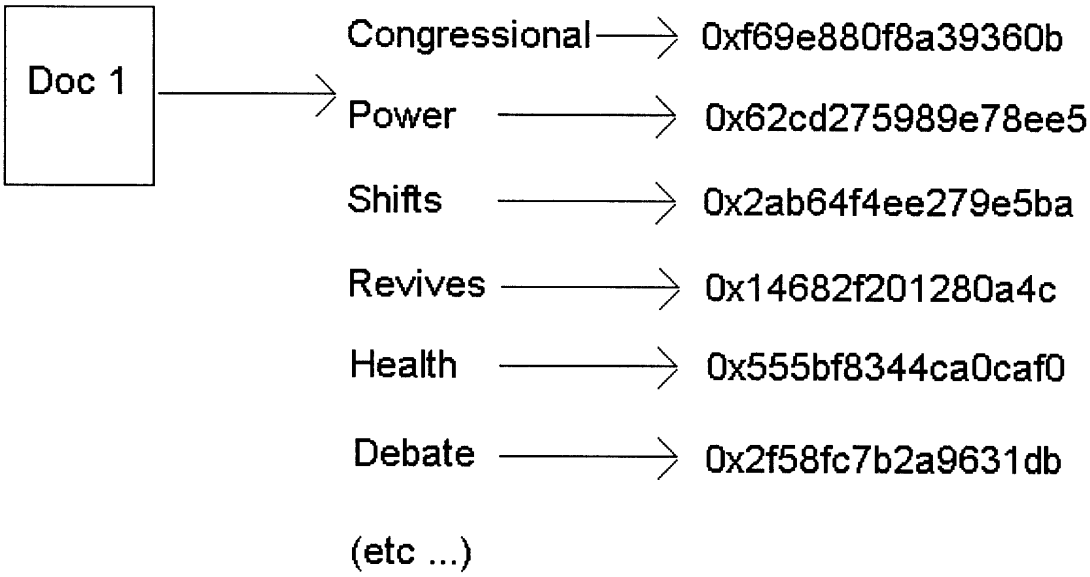

Figure 3-1: A selection of features drawn from an example document in our corpus – a January 2, 2007 New York Times article called "Congressional Power Shift Revives Health Care Debate". Each feature is then hashed individually with a standard hash function.

In our example, we have split the document into a vector F of n features, where n is the number of words in the document.

3.2 Standard Hash Function

Once we have our feature vector F, we apply a standard (possibly cryptographic) hash function to each of the features, and get back an eight-byte hash for each feature. We now have a vector H of n hashes, where H[i] = HashFn(F[i]) for all $1 \leq i \leq n$.

The hash function should be as uniform as possible over all 64 bits, because it ensures that different features will have a legitimate chance in affecting the overall

hash calculated at the end. As we will see in the next step, these hashes are added or subtracted to form the overall final hash, depending on the sign. Let's consider an extreme case where the crypto hash was not high entropy, and only used the lower 8 bits of the 64 we expected. In this case, the final hash will only be meaningful in 8 of its bits. So if two hashes were to differ by 3 bits, that's a much more significant difference than if 3 bits were different when all 64 were used.

The hash function we use here is Bob Jenkins's hash(), which is both fast and provides a close to uniform distribution over out space [20].

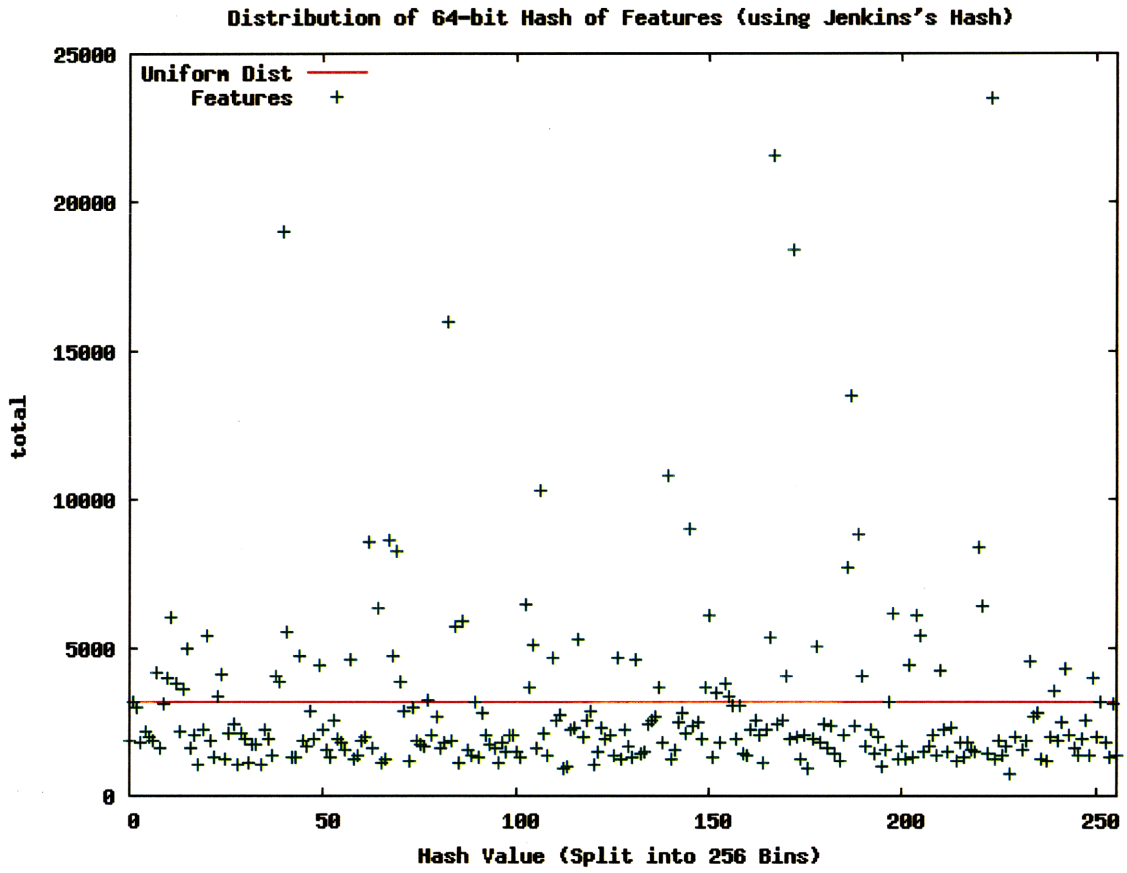

Figure 3-2: Hash value of over 800,00 features, split into 256 bins. The ideal distribution would be one close to a uniform distribution over the entire space, as marked by the red line in the plot.

3.3 Weighting

The next step in the algorithm is to create a weight vector W of size n, where $W[i]$ is the weight of feature $F[i]$. These weights represent the relative importance of the feature to the document, so the higher the weight, the more salient the feature is. For example, features that are common words such as 'this', 'is', 'or', and 'a', are assigned low weights, whereas less common meaningful words such as 'appliance' and 'detection' are given high weights.

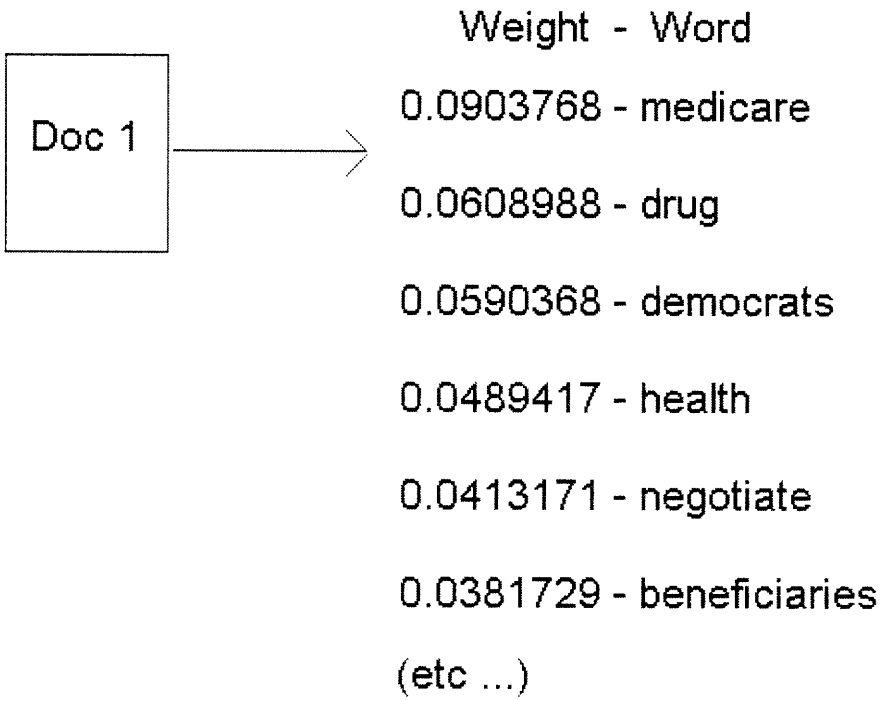

Figure 3-3: A list of the top features in the same example document, sorted by weight. The words medicare, drug, and democrats are calculated to be most important by their tf-idf score.

This weighting can be done for document contents using a standard tf/idf approach. First, the term frequency of the word is determined as the number of times the word is used in the document. This value is then multiplied by the inverse document frequency, which is the inverse of the number of documents in which the word appears. Returning to our example, the word 'medicare' may appear many times in documents that describe healthcare, and it may appear in a few numbers documents

out of the entire repository. Both of these suggest that the weight for appliance will be high. A common word like 'this' will also have a high term frequency in a document, but it also appears in nearly every document in the repository, so the inverse document frequency score will bring the overall score down.

There are several standard modifications to this tf/idf approach that have been tried in other contexts. For example, one could take the logarithm of the inverse document frequency, so as not to scale the weights as dramatically in the original equation. Other possible changes are detailed in Chapter 4.

One thing to note here is that document frequency numbers actually change over time, as it depends on the rest of the corpus. Thus, calculating the simhash of an identical document a second time, shortly after the first time, may in fact lead to a different simhash. This inconsistency can cause problems in that it may cause you to believe that the document has changed when in fact it hasn't. To solve this problems, we freeze the table that stores the document frequency mappings, and only update it once in a long interval.

Initially, we begin with an df table that is derived from a standard corpus of web documents. Over time, after a given number of documents have been processed (say 1 million), we update the document frequency table to be fresh with the current information. At this point, we consider all of our previous hashes as stale, and go back and recalculate them with the new document frequency table.

While the cost of recalculating all of the simhashes may seem large, the benefits do outweigh the cost. The hashes can now be trusted to be consistent, and we also learn more information about the corpus. If the corpus has a particular vocabulary which is unique, it can be expressed in this document frequency table and more accurate weights can be applied to the features. So say for example we have a corpus of documents within Google's private intranet. This corpus is likely to have the word "Google" in many of its documents, so its document frequency will be much larger than from a standard corpus.

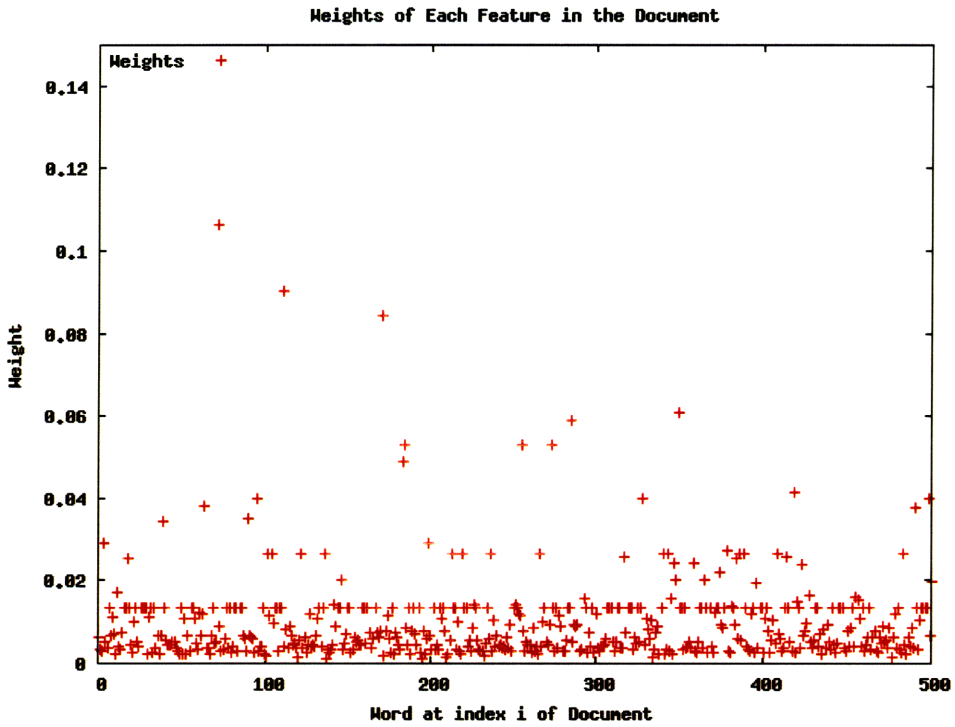

Figure 3-4: The weights of every feature in the same example document, ordered by their location in the document.

3.4 Calculating the Final Hash

Once we have our vector of hashes H and vector of weights W, we can combine them to form a final eight byte hash value. The easiest way to visualize this is to work bit by bit. Let's consider the first bit of our final answer. We look at the first bit of each of the hashes in H. If the first bit of H[i] is 1, we add the value of W[i] to a running sum. If the first bit of H[i] is 0, we subtract the value of W[i] from our sum. After summing all n values, we look at the sign of our final sum. If it's positive, then our final bit will be 1; if it is negative, our final bit will be 0.

This process is then repeated for each of the 64 bits in the final hash. All n weights are added/subtracted based on the bit at the specific index of each hash, and the sign determines the final sign. See Chart [?] for an example of this calculation.

This process of determining the final hash reveals the importance of the weights in this process. In order for the final hash to chance, the sum of all of the weights at that particular bit index has to flip signs. If the weight of a particular feature is very

Table 3.1: An example of how to calculate the final simhash. Note that if a less important word, like plan, is removed, the simhash is unchanged. If a more important workd is removed, the simhash may change values.

Feature	Weight	Hash	⇒	Buckets			
medicare	0.09	'1001'		+0.09	-0.09	-0.09	+0.09
plan	0.01	'1110'		+0.01	+0.01	+0.01	-0.01
democrats	0.06	'0010'		-0.06	-0.06	+0.06	-0.06
health	0.05	'0101'		-0.05	+0.05	-0.05	+0.05
negotiate	0.04	'1101'		+0.04	+0.04	-0.04	+0.04
Sum				+0.03	-0.05	-0.11	0.11
Final Simhash				1	0	0	1

small, then the chance that changing it would change the final sum is very small. If the weight of a feature is relatively big, then the chance that it changes the sign of the final sum is relatively high.

This also once again demonstrates the importance of having a standard hash function for each feature that uses the entire 64-bit space evenly. If the value at a particular bit were not distributed evenly between 0s and 1s, then the sum of the weights for that bit will drift further and further from zero, making a sign change very unlikely.

Chapter 4

Improvements in Feature Selection

We now shift gears to think about improvements we can make to the simhash algorithm, to provide more accurate document clustering. We first look at feature selection, and then consider how to improve the weights applied to each of these features.

4.1 Metadata

As described in section above, enterprise or specialized corpora often contain extra metadata about documents that can be leveraged to provide more information. For example, newspaper articles from a corpus can contain information such as a headline, byline, section, date published, or topic tags. A company's internal intranet can contain titles, authors, departments, and revision information. This extra data is not immediately known from the pure content on the page, so using them as extra features could be beneficial.

For example, consider our document with feature vector F. We can add entries to F such as the article's title, the last modified time, and the domain it belongs to (a prefix of its url that defines the internal department it belongs to). Since we can't easily determine a weight for these features because we can't expect an equivalent tf/idf score for them, we can manually assign a weight for these features as a proportion of the max weight that already exists in W. This way, a change to any of these features

will most likely cause a change in the final simhash value.

Other metadata which is numerical can be incorporated directly into the weights to provide additional features. For example, if you had information about the link graph, you can use information about the number of sites that link to the document as an additional feature. If this is being used within the context of a search engine, it should store some notion of page quality, such as a PageRank, which can directly be added as a feature. We can normalize this PageRank so that pages which are of very high or very low quality will have high weights, and those in the middle will have lower weights.

Other information available to the server conducting the search could also be fed back as features of the system. Data such as the number of times that the page has been viewed, or the number of times that the page has been clicked on from search results, might be effective as a new feature.

4.2 Cleaning Text

As opposed to adding new features to F, we can also consider removing those already present in F. Common stopwords are often removed from the feature list, including words such as a, as, at, in, and many others. These words would have otherwise had a low weight anyway, so their impact on the final hash is probably negligible (except in some cases, which we discuss in section 5.1.

We can also remove from our feature list items such as numbers and dates. Consider, for example, pages that contain tables of numbers as its information. While the table may be continuously updated for information, when determining the real content of the page, the informative information is the text around the table, labeling the columns/rows. If we have 2 documents with the table, with only the numbers having changed, then it is likely that these documents should be considered near duplicates of each other. If the quantity of numbers is large compared to the quantity of text, then if the numbers were not stripped, it would claim the page has changed too much to still be considered near duplicates.

Dates and time can also be removed from the document. This is intended to catch several cases. Sometimes, documents have information about the current date, which clearly is new every time the document is downloaded, but not important for the content of a page. There is also sometimes information on the page that represents the last updated time. While one may argue that it is important for knowing if the page has changed, we want to judge the significance of the change. The main reason to assign weights in the first place is to be able to quantify the level of change, so stripping or giving low weight to the last modified weight and looking at the main content seems appropriate.

An interesting example where dates and times are important to remove is calendar pages. Blank calendar pages (such as pages representing times much into the future) are often blank except for the date that changes with every page. Stripping these allows us to be able to classify all of these pages as near duplicates, and eventually de-prioritize indexing pages of dates far into the future.

Chapter 5

Improvements in the Scoring Function

Once we've selected features to represent the document, assigning them appropriate weights is important in the quality of the hash. The following are multiple ways to modify the way that features are weighted, with the goal of improving the overall ability to hash and cluster near duplicates.

5.1 Low Threshold

The motivation behind applying weights to each feature was so that if an important feature changed, it would have a large affect on the final sums that determined what the final hash would be. However, if there were many changes to features with small weights, it is possible that these would outweigh, or drown out, the changes to an important feature or two. This effect is determined by how big of a range the weights cover.

One way that we have tried to solve this problem is removing stop words in the beginning. Most of the words with low weights are common words, which we can recognize and strip away right away (or assign their weight to be 0).

Another way to solve this problem is to assign a threshold, under which weights are automatically assigned to 0. Say we normalize the weights to float values between

0 and 1. Then it is possible to take any weight that is less than 0.2 (or some constant), and set it to 0. That way, it's unlikely that a large set of lower weight features will drown out the effect of a high weight feature. Figure [?] represents a distribution of what the new features would be valued at, now that ones below the threshold have been removed.

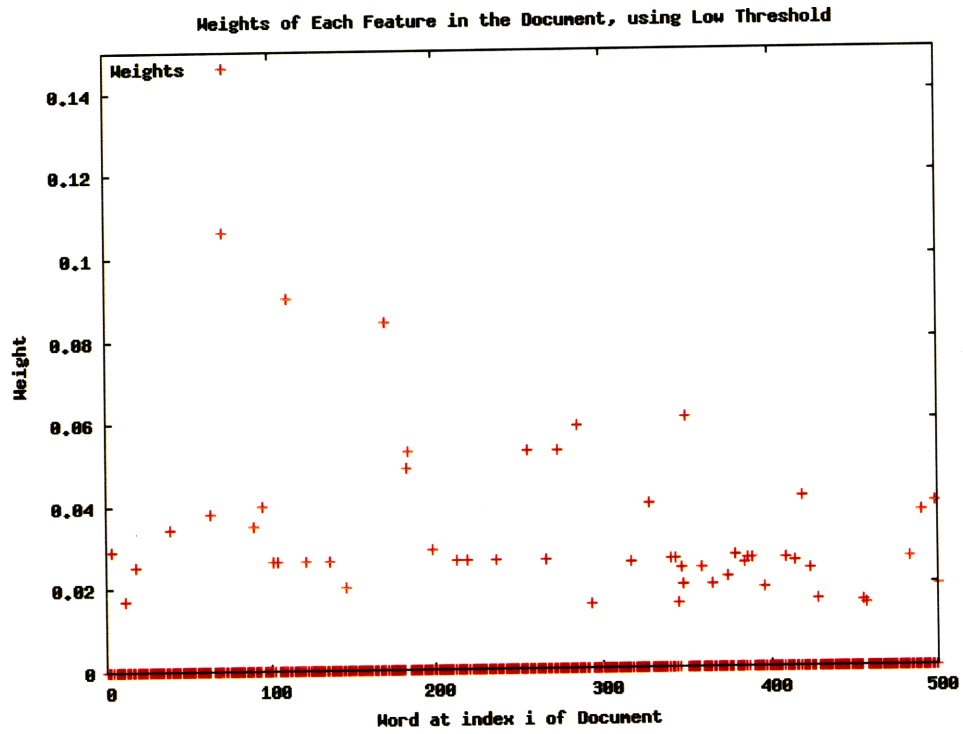

Figure 5-1: The weights of every feature in the same example document, with a low threshold of 0.015. Compare the plot to the full weights in Figure 3-4.

Another way to approach this problem would be to look at the other end of the range, the features with high weights. Instead of assigning a low threshold, we can just as easily assign a high threshold, which only takes into account features with weights higher than a particular value. The alternative is instead to choose a specific number of features, and only keep those number of features. So from our feature vector F, we choose the 20 features with the highest weight, and set all of the other weights to 0. (This is in effect reducing the dimensions of our space, since each feature vector will be a fixed, smaller size).

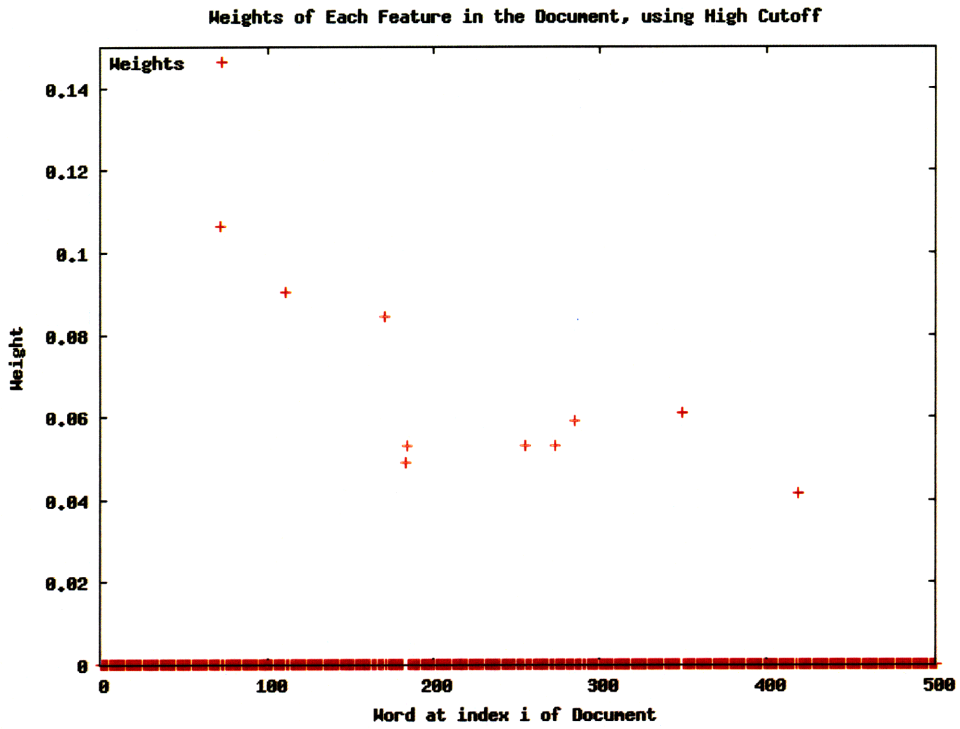

Figure 5-2: The weights of every feature in the same example document, with a high cutoff of 0.04.

5.2 Document Length Normalization

Document length normalization is a common technique used to scale weights so that documents of longer length will have weights on the same scale as those with a shorter length. Without it, documents with a longer length may have much higher term frequency counts, so their features may be given higher weights simply because it was the document was longer, and not because the word is actually more important. Normalizing the term frequency counts by the total length of the document helps to bring all of these weights to the same scale.

However, if we consider how these weights are used in the algorithm, we can see that the standrad DLN may not be as influencial as it is in other IR settings. The weights for each feature in a document are combined only with other weights from features in that same document. This final sum, which becomes the final hash, is what is eventually used to compare documents with each other. So the fact that the weights for a document were on a different scale doesn't matter as much, as long as

all of the weights within the same document were comparable.

For example, a long document will have high term frequency counts, so its weights may range from 0 to 20, to pick an arbitrary range. A smaller document may have weights that range from 0 to 4. If these weights were being compared directly, then one fear would be that a word with weight 15 from the first document would outweigh a word with weight 3 from the second document, even though both words were roughly as important as each other. This will not happen though, as the word with weight 15 was only added with other weights from the same document.

However, we can still use the intuition behind DLN to make improvements on our weights. Consider the content in a long document – as the document gets longer, the importance of the content towards the end diminishes. A change to the beginning of the document may well be more important than changes towards the end of a long document. Thus, if we can scale the weights such that the weights at the end of the document are smaller, then they will be factored less into the final hash.

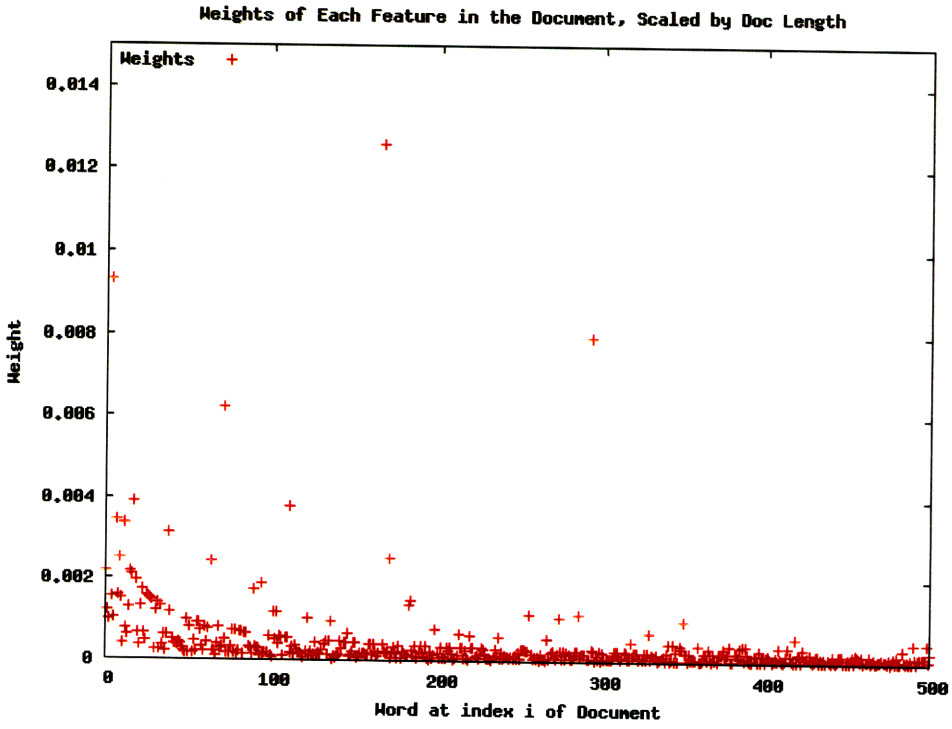

Figure 5-3: The weights of every feature in the document, scaled by log(x)/x so that the beginning part of the document is given more weight than the end.

Scaling functions that could work in this respect are 1/x, or 1/log(x), or log(x)/x.

We choose log(x)/x because it scales much slower than the others. If we multiply every weight W[i] by log(i)/i, then we can achieve this scaling. See Figure[?] for an example this effect being applied to our weights.

5.3 Scaling by an Arbitrary Function

Having read the previous section, one may immediately argue with my statement "as the document gets longer, the importance of the content towards the end diminishes." Rightfully so. The location of a word on a page doesn't always reflect its importance, and it certainly doesn't always apply that words at the end are less important than words at the beginning.

Sometimes, the domain of the document provides insight into regions that are important. For example, it's possible that both the beginning and end of a document are important, while the contents in the middle are less so. If the beginning part, what might describe the objective of the document, and the end part, which may describe its conclusions, change, then that may be considered more significant than if the middle, which contains some minor details, changes.

It's impossible to have a scaling that applies in all cases, so what we can do instead is use a function that one can define differently for different pages. This function will be a mapping between word location in the document to the value you wish to scale it by. So for example, from the previous section, the function would be f(x) = log(x)/x, where x is the position of the word in the document.

In order to not overfit for specific document class, we can use a generic version of a continuous function. So for the example where the beginning and ends of the document are important, we can think of our scaling as a cosine curve. The period of the curve is the length of the document, and it can be shifted up so that all weights will be non-negative. See Figure 5-4. The equation would be: $f(x) = \cos(\frac{2\times\pi\times x}{L}) + 1$.

If a particular document class has a particular structure, then this function can be modified to scale the weights different in that manner.

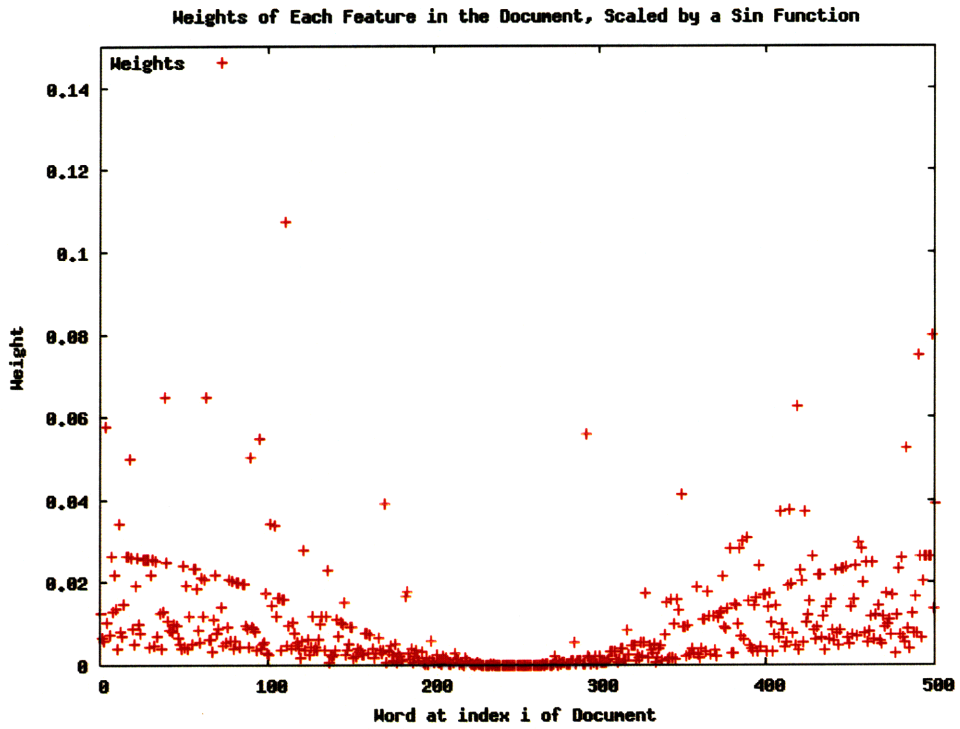

Figure 5-4: The weights of every feature in the document, scaled by a cosine function to give more weight to elements in the front and end of the document.

5.4 Tagged Text

If documents are in the form of html, there are also html tags within the content that can help denote level of relevance or importance. For example, the `<h1>` tag would denote a large heading in a page, which is rather important to the page. If a word in the heading changes, then it should have a greater affect than if a word in the normal body text changes. Similarly, other style tags such as `<b>` and `<i>` denote information that we can take into account in the weights.

We can take this information into account by simple increasing the weights of features that are within these tags by a constant. We could introduce a multiplicative factor instead of an additive one, so that the weights remain on the same scale.

Other content which can be used in this fashion depends on what is available at the time of processing. In a search engine, the anchor text information is generally available. This is is the phrase that is included within `<a>` tags in another page that links to the current page being processed. Often, this is a good short description of

the current page. For example, if page A contains a link to page B, and the text of the link may be useful as a feature when processing B. The weight of this feature can be boosted, just like the features within html tags.

Chapter 6

Experiment Evaluation

To test the effectiveness of modifying simhash by using these other methods, we attempt to detect near-duplicate docuements from a corpus of news articles. For each article, we calculate the simhash using the new method, and then find all other articles whose simhash is within varying Hamming distances of it: 0 (an exact match), 1, 2, and 3 bits.

We measure the precision of these based on the resulting clusters. We do not measure recall, as that would require ennumerating all possible pairs of articles in our corpus by hand to test if a pair of documents are truly near duplicates of each other. Other metrics that have been used in the literature to measure recall include cosine similarity, but again that measure is inefficient over a large corpus.

6.1 Setup

6.1.1 Article Corpora

The specialized corpora that were used to do the evaluation were a collection of news articles from a variety of sources.

- The New York Times annotated corpus – 1.8 million documents, from 1987 to 2007, of which approximately 1.5 million were manually tagged and 275,000 were algorithmically tagged and verified.

- The HARD Corpus – roughly 630,000 articles from 2003 from the Agence France Presse, Associated Press Newswire, Central News Agency Taiwan, Los Angeles Times/Washington Post, New York Times, Salon.com, Ummah Press, and Xinhua News Agency.

- The TIPSTER Complete corpus – over 1 million articles from 1987-1992 from the Associated Press, Wall Street Journal, Dept. of Energy, Federal Register, Ziff/Davis Publishing, and the San Jose Mercury.

These corpora were made available by the Linguistic Data Consortium.

In total there were 3,570,693 articles available to analyze. After an initial pass through all of the data, we decide to narrow the corpus to only include articles from the year 2003. This will enable us to better evaluate the results, in the context of a small set of data, as the results from sampling will be more meaningful.

Thus, the following experiments used a collection of 729,885 documents total from the 2003 portion of the New York Times collection and the entire HARD corpus. Since this collection represents eight different news services covering the same time period, the number of near duplicates we expect from this size corpus is higher than we would expect from a random enterprise corpus. For any given event that took place in the world, there is a good chance that multiple of these services covered it, so the chance of near duplicates is high.

The format of the articles were different among the corpora, creating a challenge in parsing them consistently. The New York Times corpus came in the News Industry Text Format (NITF), a flexible XML specification that allows one to specify metadata associated with the particular article. The HARD and TIPSTER corpora were formatted in Standard Generalized Markup Language (SGML), which presented as a stream of loosely labelled XML, to denote major structures such as headlines and body text.

6.2 Processing the Corpora

The calculating of simhashes and finding all of the documents within a given Hamming distance was done on two Xenon 5400 2.8 GHZ quad-core processors with 6 GB of RAM each running Ubuntu 8.10. Calculating the simhash for all 730k articles took approximately 10 minutes, parallelized over the eight cores. Determining, for each document, the set of all documents within a particular Hamming distance, took approximately 2 hours to run, in parallel over the eight cores.

The processing was done in two stages: simhash calculation, and document clustering. Simhash calculation could be done in parallel trivially by splitting up the document space that we wanted to evaluate. Initially a document frequency table was created covering the entire corpus, and this static table was used throughout for scoring. Note that because we were conducting a snapshot evaluation of these corpora, the documents themselves were not changing. We thus did not have to worry about changes to the document frequency table, which could cause problems in the scoring as described above.

After the simhash was calculated, the value was written out to a file, which afterwards was passed to the second stage for processing. In this stage, all of the simhashes were read into a database in memory, for quick access. We were able to do this because of the relatively small size of the corpus. For much larger corpora that are not able to be stored in memory at once, the keys can be sharded across mutiple disks (see [24]). We then performed all n^2 comparisons to compute the hamming distance between each pair. While this could have been sped up by constant factors through minor tweaking, the sacrifice in using memory was too great that the naive pairwise comparison ended up being the desireable choice.

As an aside, on an online system, determining this set of near-duplicate documents could be an expensive operation if it required iterating through all other n documents in the index. Manku et al.[24] address this problem by using multiple sorted lists of the simhashes, where we can narrow the space down to a small segment and then search over that segment. This is a compromise between the two extreme approaches

of storing all ${}_{64}C_3$ possible matching hashes in a table, or keeping the table small but then doing ${}_{64}C_3$ queries against it. Still, their middle ground solution still assumes access to available memory.

If memory is a concern, sharding the table of simhashes can achieve a constant value increase in processing without using much extra space. If the table of simhashes were sharded by the number of 1's in the 64-bit simhash, then the search space for new documents looking for other simhashes within a certain Hamming distance will be reduced by a constant factor. For example, say we area looking for all documents whose simhash is within a Hamming distance of 3 from our calculated simhash, which we know has i ones in it. If there were a separate table associated with each possible value of number of ones, we would only have to search through seven tables, $i-3$ to $i+3$, instead of all 64.

6.3 Standard Simhash

As a baseline, all articles were run through our implementation of an unaltered, standard simhash algorithm as described above.

A distribution over all of the simhashes created can be seen in Figure 6-1. The distribution has a fairly consistent variation from the mean, althought the periodic nature of the values is interesting. This could be a product of using Jenkins's hash function, or the existence of groups of similar pages each of which consolidate around the same space.

Another interesting view into the simhashes is characteristics of the clusters of documents it created. Figure 6-2 shows the number of clusters of various sizes that were returned after varying the Hamming distance among 0, 1, 2, and 3. Here, a cluster represents the set of all documents that are said to be near-duplciates to an individual document. Thus, if there are 1000 documents who have the identical simhash, we count this as 1000 clusters of size 1000. Each document is the leader of its own cluster, and there is one cluster for every document.

We define clusters in this way to avoid inconsistencies with the requirement for

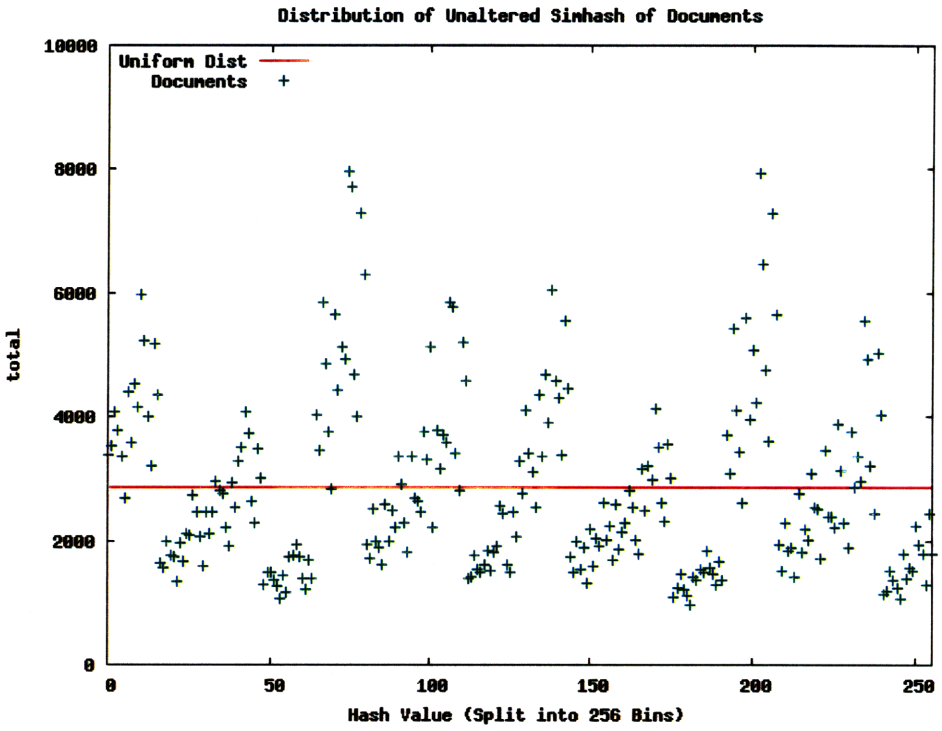

Figure 6-1: The distribution of simhashes over the entire space, binned into 256 bins.

Hamming distance within a cluster. To repeat our example from before, let us consider the case when documents B and C are in A's cluster. Each can have a Hamming distance of 3 with A, but the Hamming distance between B and C can be as large as 6. We allow this to happen, as the goal is to find near duplicates of A itself. We could restrict the definition of a cluster further to only allow documents that are within a Hamming distance of 3 from everyone in the cluster, but this leads to complications in joining/splitting clusters, and the final solution may not be unique.

Returning to the plot, it shows how as the Hamming distance is restricted to smaller values, we expect the size of the clusters to decrease. The clusters appear to move from the right side of the plot with a high hamming distance (as shown in red) to small distances on left side (as shown in purple.).

To determine the precision of the algorithm, for each Hamming distance, 200 documents were selected at random from nearduplicate pairs. Detemermining the percentage of that random sample that were in fact near duplicates was done manually by human evaluation. The pair was declared a near duplicate if the articles were both

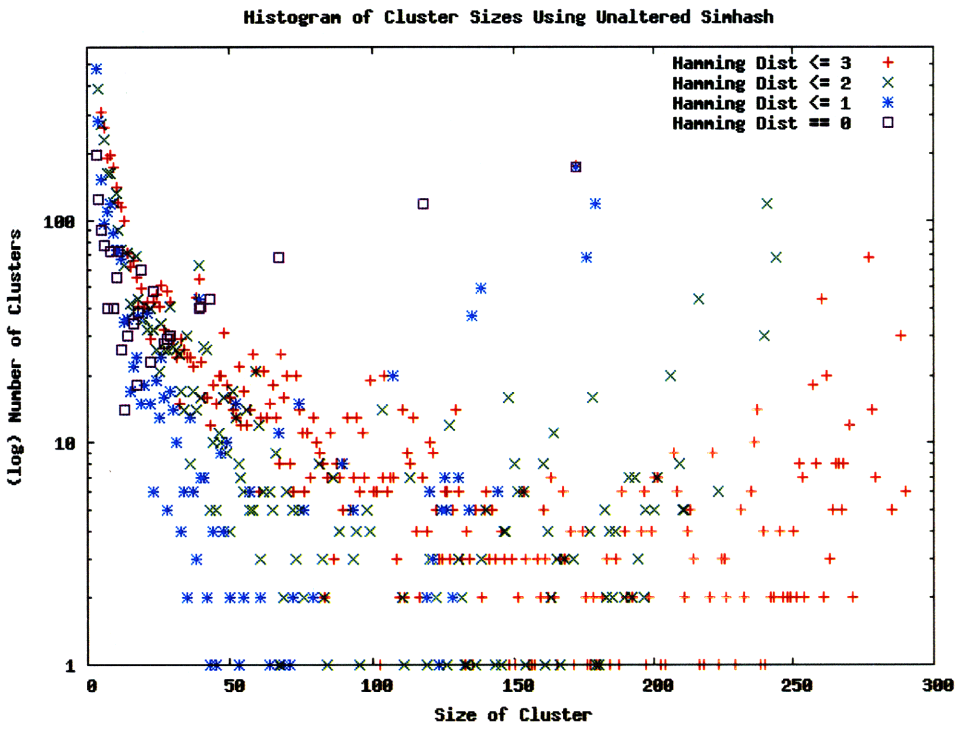

Figure 6-2: The number of clusters of a particular size, shown on a log scale, for clusters of Hamming distance 0, ≤ 1, ≤ 2, and ≤ 3.

on the same specific topic (such as a specific event or person), and shared a significant amount of content. If an article was too close to judge, it was declared "unclear", and not used in the calculation of the precision statistic.

Error is introduced in this calculation because of the size of the sample (we chose random pairs to make it as representative as possible), and because of the human factor in evaluating whether a pair is a true near duplicate or not. Since consistency helps to mitigate this error, we use the same human evaluator to perform all judging.

The results of our testing are shown in Table 6.1. We repeat this process for each of the seven following adjustments to simhash, and note where the results were improved or worsened.

Table 6.1: The precision of the near duplicates returned by varying methods, among pairs of documents with Hamming distances of less than or equal to 0, 1, 2, and 3.

Methods	HD = 0	HD ≤ 1	HD ≤ 2	HD ≤ 3
Original Simhash	0.82	0.71	0.50	0.37
Extra Metadata	**0.89**	0.75	0.66	0.59
Stripping Numbers/Dates	0.61	0.39	0.29	0.18
Low Threshold	**0.84**	0.69	0.50	0.38
High Threshold	0.59	0.50	0.36	0.27
Doc Position Scaling	0.36	0.24	0.22	0.10
Arbitrary Function	0.46	0.34	0.26	0.09
Tagged Text	**0.83**	0.37	0.52	0.35

6.4 Results for the Various Methods

6.4.1 Adding Extra Metadata

The New York Times corpus was annotated with extra metadata which was not included in the calculation of the original simhash. This is material that was added later by professionals manually and through algorithmic means, and were stored in the NITF structure of the document. To someone viewing the page in HTML on the web, this information would be hidden to them. But crawlers or people viewing the full original document can access and use this metadata for processing.

As an example, here is metadata that was associated with two articles that were marked as near duplicates. The first was from June 24, 2003,

```
<title>World Business Briefing | Asia: Japan: Trade Surplus Expands</title>
    <meta content="MB231847" name="slug"/>
    <meta content="24" name="publication_day_of_month"/>
    <meta content="6" name="publication_month"/>
    <meta content="2003" name="publication_year"/>
    <meta content="Tuesday" name="publication_day_of_week"/>
    <meta content="Business/Financial Desk" name="dsk"/>
    <meta content="1" name="print_page_number"/>
    <meta content="W" name="print_section"/>
```

```
    <meta content="1" name="print_column"/>
    <meta content="Business" name="online_sections"/>
...
    <classifier class="indexing_service" type="descriptor">
        International Trade and World Market
    </classifier>
    <classifier class="indexing_service" type="descriptor">
        Economic Conditions and Trends
    </classifier>
    <headline>
       <hl1>World Business Briefing | Asia: Japan: Trade Surplus Expands</hl1>
    </headline>
    <byline class="print_byline">By Ken Belson (NYT)</byline>
    <byline class="normalized_byline">Belson, Ken</byline>
    <abstract>
        <p>Japan's merchandise trade surplus expanded 12.5 percent in
        May from the month a year earlier as growth in exports outpaced
        the increase in imports. Though Japan's surplus increased to
        694 billion yen ($5.8 billion) in the month, exports grew 3.5
        percent, which was the slowest growth in 13 months. The effect
        of severe acute respiratory syndrome, or SARS, in Asia was
        partly to blame.</p>
     </abstract>
```

The second is from January 28, 2003.

```
<title>World Business Briefing | Asia: Japan: Trade Surplus Expands</title>
    <meta content="MB231109" name="slug"/>
    <meta content="28" name="publication_day_of_month"/>
    <meta content="1" name="publication_month"/>
    <meta content="2003" name="publication_year"/>
```

```
    <meta content="Tuesday" name="publication_day_of_week"/>
    <meta content="Business/Financial Desk" name="dsk"/>
    <meta content="1" name="print_page_number"/>
    <meta content="W" name="print_section"/>
    <meta content="5" name="print_column"/>
    <meta content="Business" name="online_sections"/>
...
    <classifier class="indexing_service" type="descriptor">
        International Trade and World Market
    </classifier>
    <classifier class="indexing_service" type="descriptor">
        Economic Conditions and Trends
    </classifier>
    <location class="indexing_service">Japan</location>
    <person class="indexing_service">Belson, Ken</person>
    <headline>
        <hl1>World Business Briefing | Asia: Japan: Trade Surplus Expands</hl1>
    </hedline>
    <byline class="print_byline">By Ken Belson (NYT)</byline>
    <byline class="normalized_byline">Belson, Ken</byline>
    <abstract>
        <p>Japan's trade surplus hit 791 billion yen ($6.7 billion)
        in December, growing 19.9 percent compared with December 2001;
        trade surplus with US grew 18.6 percent, to 624 billion yen.</p>
    </abstract>
```

Note that both articles had different main content in their articles, and were clearly written about two different events. However, they shared a lot of metadata in common, such as the classifier tags that the indexing service applied to it, the headline and author of the article, the section the paper belonged in, etc.

Without including the metadata, these two articles were not selected as near

duplicates, because the content of the articles themselves differed too much. But by including the metadata, we can now discover new similar content (articles written five months apart).

The other corpora used in this evaluation did not include such metadata, so the only articles which we compared against each other were only New York Times articles. Because this makes one of our original goals of discovering near duplicate documents across news services irrelevant (since we only have one), we relaxed the definition of near-duplicate here to include cases such as this where the content of the article was in fact different, but the topic and events discussed were very similar.

This modification to simhash resulted in the highest precision compared to all the others (Table 6.1), suggesting that metadata plays an important role in determining similarity. This is more relevant to our application of enterprise search, as we discussed how metadata information was more readily avialable to enterprise corpora than to the general web.

The impact that Hamming distance played on metadata clusters was also informative. The average size of the cluster returned by the algotihm when comparing the case of documents within Hamming distance 0, 1, 2, and 3 did not change drastically. Consistently staying around four documents suggests that the utility of spending extra processing time calculating Hamming distances may not be worth it in this case. We can much more efficiently calculate an exact match between simhashes than those within a Hamming distance of 3. If doing so does not result in too big of a loss of data, it may be more practical to only look for identical simhashes.

6.4.2 Removing Numbers, Dates

This modification removed from the feature set all features that contained a number, or any feature that represented a date or time. For example, in the sentence from the second article above, applying this change would result in the first sentence of the article becoming:

```
japans trade surplus hit billion yen billion in growing percent
```

```
compared with trade surplus with us grew percent to billion yen
```

(The words 'in', 'with', 'us', and 'to' would also be removed as they are common English stopwords, but were left in here for clarity.)

Depending on the article, this has a significant affect on the size of the feature set. The reduced size had a negative effect on the precision of simhash, as shown in Table 6.1. The average size of clusters also increased.

6.4.3 Low Threshold

Removing features whose weights were below a certain threshold provided a very slight increase in precision for most Hamming distance comparisons. This was largely due to the fact that of all of approximately 730,000 articles, only 1256 of their simhashes changed after the removal of these weights. Thus, most clusters that formed originally were left untouched.

This particular threshold was chosen by seeing what percentage of the max weight would on average remove 50% of the features. A constant value of 0.01 approximately gave this result after testing on the weights of 500 articles, so in this test we removed all weights whose weight was below 0.01 times the max weight of a feature.

Future work could investigate how other constants, or weighting cutoffs, have an affect on the resulting simhash, and thus the resulting nera duplicate clusters.

6.4.4 High Threshold

We also set a high threshold of 0.5 times the maximum weight, and rejected all features below it. This constant was set similarly to how the low threshold was set, in that we wanted to approximately eliminate all but the top 10% of features. The results, however, poorer than the low threshold.

The highest precision obtained through this method was 0.59, and possibly correlated, the average size of a cluster increased slightly. Qualitatively, when doing the human evaluation for this method, the major differences lied in the short articles, which were now clustered together with higher frequency. By removing a large set of

features in a small document, there are only few features remaining to represent the article, so it makes sense that more articles would have closer simhashes.

Note that attacking a top percentage of the features, rather than a fixed number, encapsulates one of the requirements for similarity that Conrad and Schriber [11] found important – document length. If only the top ten features were selected, as opposed to the top ten percent, it would be more difficult to distinguish articles of different sizes. The addition of many more features in the ten percent of large document compared to a small one would have a significant impact on the simhash.

6.4.5 Doc Position Scaling

Scaling the weights by their position in the document resulted in poor precision, a maximum value of 0.36. Each feature's weight was scaled by a factor of $\frac{log(x)}{x}$ (shifted so that the result for small x is positive), with the hope that if more important information were kept at the front of the document, it would get more weight.

In evaluating the results for this method, a common characteristic of pages that were near duplicates shared common information at the start of the page. While for newspaper articles this is promising, as most are written in an inverted-pyramid style to include the most important information at the beginning, the results did not work out. Perhaps scaling the weights by a different, arbitrary function would lead to better results, as we discuss in the next section.

6.4.6 Scaled by Arbitrary Function

Scaling the weights by a cosine function over one period also produced poor results. The highest precision detected, for pages whose simhashes matched exactly, was 0.46.

The fault in this method could be the same underlying problem that the high threshold suffered from – losing too many features. The cosine function dropped the importance of many of the features from the middle of the document, giving very high importance to the ones at the beginning and the end. These features could then dominate the final simhash, and result in large clusters where those few powerful

features would be common among the documents, even if their middle content is much different.

From the results of the past two methods, it appears as if scaling by a general function is not the most productive in determining similarity. The motivation behind these methods was previous knowledge about the structure of the document, thinking that if the most important information were at the front or end of the document, scaling by cosine would add importance to those sections. Instead, it appears as if it has just overly dominated the middle lower features.

Before eliminating this method, however, consider the second use of simhash, which has not been evaluated here. Simhash can be used to determine how a page has changed over time, by comparing its current hash with previous values. Since the corpus we are testing on is not evolving, it is difficult to evaluate this metric precisely. Intuition suggests that using a scaling function will help determine when an important part of a page has changed, so this method may be saved for that purpose, but future work on an evolving corpus will have to demonstrate it.

6.4.7 Tagged Text Boosting

Boosting the weight of specially tagged text provided minimal changes, if any. Each feature that was encalsulated in HTML markup tags, such as `<b>`, `<i>`, or `<h1>`, had their weight boosted by a factor or 1.5.

This resulted in simhash precision nearly identical to that of the original simhash. Based on the human evaluation of this method, this result may be due to a fault in the experimental corpus, rather than the method. Most of the articles in the New York Times corpus and the HARD data streams lacked HTML formatting, besides `<p>` tags to designate paragraph breaks. The most prominent tag was `<h1>` tags around headlines, which appears to have had a small affect, but can be folded into other more promising methods, such as metadata extraction.

Table 6.2: The following numbers are the average number of documents that the algorithm said was a near duplicate of a document, based on having a Hamming distance less than or equal to the specified amount.

Methods	HD = 0	HD ≤ 1	HD ≤ 2	HD ≤ 3
Original Simhash	3.49	6.06	7.15	7.73
Extra Metadata	4.07	4.09	4.16	4.34
Stripping Numbers/Dates	11.25	14.70	16.19	18.24
Low Threshold	3.49	6.06	7.15	7.73
High Threshold	4.34	7.03	8.25	9.30
Doc Position Scaling	44.18	51.99	55.74	58.70
Arbitrary Function	39.98	48.13	56.58	61.32
Tagged Text	3.49	6.06	7.15	7.73

6.4.8 Deciding on a Hamming Distance

A Hamming distance of 3 was the canonical choice for cutoff to determine similarity based on the work of Manku [24]. The higher the cutoff is, the more inefficient the algorithm is in terms of processing time and memory storage requirements. Moving to a Hamming distance of 4 would lead to clusters that are a superset of those with Hamming distance 3, requiring extra memory to store all of the information, and extra documents to process when determining clusters.

Minimizing the Hamming distance necessary to still receive valuable near duplicate information is thus important in a setting limited in resources, such as with an enterprise search engine. One way we can measure this is by looking at the size of the clusters being returned at the various Hamming distances. If the size of the clusters are staying the same as the Hamming distance changes, then it suggests that most of the information is already gained by just using the lower Hamming distance.

For example, using the method with extra metadata, switching from an exact comparison to a Hamming distance of three only results in on average an additional 0.27 documents. This is probably not worth the computing power wasted on computing Hamming distance.

However, looking at the way the original simhash, using a Hamming Distance of 3, would double the number of documents available in the cluster, so depending on the

decrease in precision, using a Hamming distance of 3 may be worthwhile to achieve a higher recall with the larger cluster.

Given that a Hamming distance of 0 is the least resource intensive, and has shown to provide the highest precision, a reasonable iterative plan of action would be to begin with only calculating exact simhash matches, and increase the allowable Hamming distance as resources allow.

Chapter 7

Future Work and Conclusion

There is still much work to be done in the field of near duplicate detection, some of which is a direct follow-on to my findings here. These are a few ideas of what the next steps could be in this direction of exploration.

7.1 Changes to simhash

7.1.1 Feature Selection

The results demonstrated how simhash was sensitive to changes in feature selection, in that the inclusion of metadata had an effect on the final hash. This was not surprising, and builds upon Conrad and Schriber's work which also noted an increase in accuracy when incorporating metadata as features [10].

An open question remains as to how this information can best be encoded within the 64-bit fingerprint. Since these features are not part of the standard text of the document, they can't easily be given a weight that is equivalent to a tf-idf score, which is what is done for most features. In this study, the metadata that was included as features were given a weight equal to a proportion of the maximum weight of a word in the document. This worked in casuing a change, but what is an appropriate amount of change? Instead, should the weight have been twice the maximum weight, or half, or some other arbitrary value?

Another possibility could be to set the weight such that a change in the metadata would guarantee a change in the resulting simhash. If their weights were set to the minimum of the magnitudes of the final running sums which are calculated when the final simhash is calculated, then a change in the feature might guarantee a bit flip in the simhash. For example, if the cumulative sum of weight W[0] for all features in the document were 0.7, the bit in the first position of the final simhash would be 1 (since $0.7 > 0$). If the weight of this additional metadata were now set to 0.7, then a change in the metadata would cause the sum to drop to 0, flipping the bit in the final simhash to 0.

A different idea would be to reserve certain bits of the 64-bits for specific metadata. For example, what if the leading 8 bits were reserved for particular piece of metadata, such as the doc length, the last updated time, or the existence of security restrictions on the page? If the simhashes were then sorted, one can easily restrict their Hamming distance search to the lower 56 bits in only the few simhashes who matched the leading 8 bits exactly.

7.1.2 Adjusting Weights of Features

The results also demonstrated that changing the weights of the features had an effect on the final simhash, in some cases making the clustering more precise and in others making it much worse. Further work could be done on each of the improvements made in this study.

- **Low threshold** - A constant of 0.01 was chosen so that on average the weights of 50% of the features were excluded, because they were below $0.01 \times max(W)$, where W is the weights of all features in the document. Research can be done into how adjusting this parameter affects the resulting simhash, and resulting clusters. Eventually it can go low enough to not make any difference, or high enough to be like a high threshold.

- **High threshold** - Similar to the low threshold, a constant of 0.5 was chosen to only keep on average 10% of the features, as their weights were above $0.5 \times$

$max(W)$. Again, how does changing this value affect the simhash? Does it work better than the approach of saving the top n features, as opposed to the top m%.

- **Middle region** - In actuality, there is no difference between the high and low threshold, as they both specify a minimum cutoff for a weight to factor into the final simhash. Combining these, however, to only allow features with weights from the middle region, could have a positive effect on the resulting cluster. This is similar to what the I-Match [9] algorithm does when it filters out the low and high values in its document frequency table. The resulting set of features will be the more average terms in the language, so it might better represent the page.

- **Normalizing top weights** - From observing the distributions of weights that resulted from parsing documents, it was common for the top one or two features to have weights which were much larger (by a factor of 10) than even the fifth or sixth highest weight. Shifting the weights of outliers down to the same scale as the other features might eliminate the sensativity of that simhash to its top words.

- **Page Structure** - Adjusting the weighting based on known page structure information, such as derived from [1], could improve the quality of features that contribute to the final simhash.

7.1.3 Interaction with Search

The relationship between the near duplicate detection system and a search enginge was the original motivation for this work, particularily a search engine covering an enterprise or specialized space. There are properties of that space that complicate the detection of near duplicates, which could be an interesting area of research.

Security is a big concern within an enterprise, and ensuring that the search engine does not leak information about documents that people should not have access to is

vital. This makes maintaining near duplicate clusters challenging, as it is possible for different documents in the cluster to have different security restrictions. One can solve this problem naively by requiring all documents in a cluster to have the same security classification, but this is inadequate. Sometimes security information is not known until serve time when a query is being processed.

Security metadata also has an effect on the other parts of the search engine. If security prevents some users from seeing one document in a cluster, than the other documents should not be discarded from the index, thinking that the secure document is available for everyone. Thus, the benefits to a near duplicate document detection system for an enterprise search engine (smaller index size, better crawl priority for freshness, supression of duplicates when answering queries) need to be re-evaluated in terms of security.

Note that while security is the example we chose, it could apply to almost any other type of metadata that is stored with documents in a corpus. For example, say that there are two different hosts between which there are many near duplicates. Which ones should the search engine make the default? Whichever it chooses will have an affect on the load to that server, so it seems like we don't want all of the traffic to go to one. But to counter that, we dont want to keep the information still split among both hosts, as that has a negative affect on the link graph.

Chapter 8

Conclusion

We implemented the simhash algorithm to detect near duplicated documents in a relatively small specialized corpus (of size 750k). The corpus we focused on were newspaper articles from the New York Times and other news services from the year 2003. Using the pure simhash function as a baseline, we evaluated many potential changes to the algorithm, broadly categorized as changes to its feature selection and changes to its weight scheme.

These included adding extra metadata available to the documents, stripping all numbers and dates (in addition to stop words), creating a threshold under which weights did not affect the final simhash, and skewing the weights by arbitrary functions, $\frac{log(x)}{x}$ and $cos(x)$, to demontrate if the algorithm could take advantage of knowledge about the structure of the documents.

These changes were in the context of a search engine for an enterprise corpus of documents, which meant the data had certain properties such as metadata associated with it and common page structure. The metadata was directly fed back into the algorithm for calculating the simhash, and led to an improvement in precision for discovering near duplicate content. The common page structure contributed to adjusting the idf table such that common words from a template or boilerplate were given low weights.

We also evaluated the resulting clusters from performing simhash with Hamming distances varying from 0 to 3. The resulting cluster sizes, and particularly the re-

sulting precision, indicates that for systems in which there are limited computing resources, performing the extra check of Hamming distances may not be worthwhile. Simply checking for identical hashes (Hamming distance of 0), by employing a lookup table as is done for I-Match, will still retrieve a smaller but more precise cluster of near duplicates.

Bibliography

[1] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In Proc. ACM SIGMOD 2003, pages 337-348, 2003

[2] K. Bharat and A. Broder. Mirror, mirror on the Web: A study of host pairs with replicated content. In. Proc. 8th International Conference on World Wide Web (WWW 1999), pages 1579-1590, 1999.

[3] K. Bharat, A. Broder, J. Dean, M.R.Henziger. A comparison of techniques to find mirrored hosts on the WWW. J. American Society for Information Science, 51(12):1114-1122, Aug 2000

[4] S. Brin, J. Davis, and H. Garcia-Molina. Copy detection mechanisms for digital documents. In Proc. ACM SIGMOD Annual Conference, pages 398-409, May 1995.

[5] A. Broder. Identifying and Filtering Near-Duplicate Documents. CPM 2000

[6] A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher. Min-wise independant permutatinos. In Proc. 30th Annual Symposiuum on Theory of Computing. 1998.

[7] Broder, A. Z., Glassman, S. C., Manasse, M. S., and Zweig, G. 1997. Syntactic clustering of the Web. Comput. Netw. ISDN Syst. 29, 8-13 (Sep. 1997), 1157-1166.

[8] Charikar, M. S. 2002. Similarity estimation techniques from rounding algorithms. In Proceedings of the Thiry-Fourth Annual ACM Symposium on theory of Computing. STOC '02. ACM, New York, NY, 380-388

[9] A. Chowdhury, O. Frieder, D. Grossman, MC McCabe. Collection statistics for fast duplicate document detection. ACM Transactions on Information Systems. 20(2):171-191, 2002.

[10] J. G. Conrad, X. S. Guo, and C. P. Schriber. Online duplicate document detection: Signature reliability in a dynamic retrieval environment. In Proceedings of CIKM03, pages 443452. ACM Press, Nov. 2003.

[11] J.G. Conrad and C.P. Schriber. Constructing a text corpus for inexact duplicate detection. In SIGIR 2004, pg 582-582., July 2004.

[12] J.W.Cooper, A.R.Coden, and E.W.Brow. Detecting similar documents using salient terms. In Proc 1st International Conf. On Information and Knowledge Management (CIKM 2002), pages 245-251, Nov. 2002

[13] J. Dean and M. Henziger. Finding related pages in the World Wide Web. Computer Networks, 31(11-16):1467-1479, 1999.

[14] K.M. Hammouda and M.S. Kamel. Efficient phrase-based document indexing for web document clustering. IEEE Transactions on Knowledge and Data Engineering, 16(10):1279-1296, 1994.

[15] T.H. Haveliwala, A. Gionis, P. Indyk. Scalable techniques for clustering the web. In Proc. 3rd Intl. Workshop on the Web and Databases (WebDB 2000). pages 129-134, May 2000.

[16] T.H. Haveliwala, A. Gionis, D. Klein, P. Indyk. Evaluating strategies for similarity search on the Web. In Proc 11th International World Wide Web Conference, pages 432-442, May 2002.

[17] M. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In Proceedings of the 29th Annual international ACM SIGIR Conference on Research and Development in information Retrieval (Seattle, Washington, USA, August 06 - 11, 2006). SIGIR '06. ACM, New York, NY, 284-291.

[18] T.C. Hoad and J. Zobel. Methods for identifying versioned and plagarized documents. J of the American Society for Information Science and Technology, 54(3):203-215. Feb 2003.

[19] S. Huffman, A. Lehman, A. Stolboushkin, H. Wong-Toi, Fan Yang, Hein Roehrig. Multiple-signal duplicate detection for search evaluation. SIGIR2007

[20] Bob Jenkins's Hash Function. http://burtleburtle.net/bob/hash/index.html#lookup

[21] S. Joshi, N. Agrawal, R. Krishnapuram, and S. Negi. A bag of paths model for measuring structural similarity in Web Documents. SIFKDD 2003, pages 577-582, 2003

[22] A. Kolcz, A. Chowdhury, J. Alspector. Improved robustness of signature-based near-replica detection via lexicon randomization. KDD2004

[23] U. Manber. Finding similar files in a large file system. In Proc. 1994 USENIX Conference, pages 1-10, Jan 1994

[24] G.S. Manku, A. Jain, and A. Das Sarma. Detecting near-duplicates for web crawling. In Proceedings of the 16th international Conference on World Wide Web (Banff,Alberta, Canada, May 08-12, 2007). WWW '07. ACM, New York, NY, 141-150.

[25] M. O. Rabin. Fingerprinting by ranom polynomials. Technical Report TR-15-81, Center for Research in Computing Technology, Harbard University, 1981.

[26] B. Stein and Sven Eissen. Near Similarity Search and Plagiarism Analysis. From Data and Information Analysis to Knowledge Engineering, pages 430-437, 2006.

[27] B. Stein and Sven Meyer zu Eissen. Intrinsic Plagiarism Analysis with Meta Learning. CEUR Workshop Proceedings, Volume 276, 2007.

[28] M. Theobald, J. Siddharth, A. Paepcke. SpotSigs: Robust and Efficient Near Duplicate Detection in Large Web Collections. SIGIR 2008

[29] Chuan Xiao. Wei Wang. Xuemin Lin. Efficient similarity joins for near duplicate detection. WWW2008

[30] Hui Yang, Jamie Callan, Near-duplicate detection by instance-level constrained clustering. SIGIR 2006

[31] Jiangong Zhang, Torsten Suel. Efficient search in large textual collections with redundancy WWW2007