SpotSigs: Near-Duplicate Detection in Web Page Collections

Masters Thesis Report
Siddharth Jonathan
Computer Science, Stanford University

ABSTRACT
Motivated by our work with political scientists we present an algorithm that detects near-duplicate Web pages. These scientists analyze Web archives of news sites. The archives were collected with crawlers and contain a large number of pages that look very different because the frame around their core content differs. However, the news stories in the pages are nearly identical. The close proximity of unrelated items on the pages makes the detection of content overlap difficult. Our SpotSigs algorithm generates signatures that are spread across each document. Places for these signatures are determined by the placement of common words, like ‘is’ and ‘the’ in the documents. We can vary our method of computing the signatures. Using hash collisions the algorithm detects overlap among the signatures of matching contents. We study how the different SpotSigs parameters impact precision and recall performance. We propose and evaluate variants of SpotSigs on a test bed of 2168 Web Pages and study the tradeoffs involved. One of our motivations was also to keep pre-processing requirements low for the detection of near duplicates and to this end we do not remove ads, client side scripts and other HTML formatting elements from the documents. On this data set SpotSigs obtains a precision of over 93% and a recall of over 85% for near duplicate detection.

Categories and Subject Descriptors
H.3.7 [Digital Libraries]: Collection, Systems Issues

General Terms
Algorithms, Management, Theory.

Keywords
Near-duplicate detection, Web archives.

1. INTRODUCTION
The detection of near-duplicate documents and records in data sets is a long-standing problem. Near-duplicates are pairs of items that are very similar along some dimension, but different enough that simple byte-by-byte comparisons fail. In databases this difficulty has given rise to a collection of algorithms known as entity resolution approaches. These algorithms attempt to analyze record contents, like postal addresses, and try to infer identity between records. Examples are differences merely in the use of abbreviations, as for ‘avenue,’ or name spelling variations.

On the World-Wide Web, document duplication is problematic as well. Most notably, search engines attempt to eliminate highly similar documents from their search result sets.

We have encountered this problem of near duplicate content in the very different context of our Web Sociologists Workbench. This project constructs tools for political scientists, historians, and sociologists to help them analyze Web archives. Our users’ first project with us was an analysis of news coverage around a California State election. We had crawled numerous online news sites, and had filtered obviously uninteresting pages automatically. The remaining pages were to be tagged manually to...
mark content semantics, such as unions, feminism, celebrity, and local-issue.

These semantic differences were subtle enough and the tagging guidelines were ambiguous enough that machine learning algorithms of sufficient accuracy could not readily be constructed for the task of tagging or classifying these documents. We instead implemented an interface that displayed one Web page at a time, framing it with checkboxes for the addition of tags. A number of related features, such as random archive sampling, were included as well.

The human coders went to work on this collection, enthusiastically welcoming our application. Unfortunately, they failed to complete the task. Excessive numbers of near-duplicate documents overwhelmed their efforts.

There are in fact two types of near-duplicates for this application. Figure 1 shows a pair of same-core Web pages. The pair differs only in the framing. The core article about Israeli Prime Minister Olmert calling for that country’s President to resign is identical, albeit formatted differently.

![Figure 1: Same-core Web pages.](image1)

Figure 2: Duplicate Web pages; identical framing, but different core content.

Figure 2 is an example of the opposite case. The core content of interest is different between the two pages, but the framing around this content is identical. We call this case a same-frame near-duplicate.

Typically, our sociologists would only consider Figure 1’s same-core pair a near-duplicate. These would not be discarded, but would need to be collected into a set that is coded or tagged in batch.

The frequent presence of many diverse semantic units in individual Web pages makes near-duplicate detection particularly difficult. Framing elements for branding, and advertisements are often freely interspersed with other content elements. The branding elements tend to be deliberately replicated across the pages of individual sites, creating a ‘noise level’ of similarity among the pages.

Another difficulty that many Web pages present to near-duplicate detection algorithms is that coherent units of text are often small and lexically fragmented. This fragmentation occurs because HTML tables are used for layout control. The insertion of images, ads or entirely unrelated material that is arranged to either side of a visually uninterrupted text segment may thereby partition the text lexically: a linear scan through the page source finds the text interrupted by HTML tags and the neighboring material. Algorithms for detection must therefore be particularly robust.

In past work [5], a cosine similarity of 0.9 is generally accepted to be a reasonable indicator of near duplication. Unfortunately, same-core duplicates of the type we describe in Figure 1 have an average cosine similarity of only about 0.6. Our application of social scientists analyzing Web archives therefore requires a looser definition of ‘near,’ for near-duplicate detection algorithms to be effective.

We performed most of our experiments with a data set of 2168 Web pages. This manually assembled set included 67 clusters of same-core near duplicates. The average cosine similarity by cluster is shown in Figure 3.

![Figure 3: Average Cosine Similarity among all pairs of Near Duplicate documents in each Cluster](image2)

Even if care is taken to avoid true duplicates during the collection of Web archives, near-duplicates frequently slip into the corpus. For example, while long-term archives for news sites are usually closed to crawlers, many news sites contain small archives of material published during recent days. A crawler that harvests such a site daily will keep stumbling into those areas of the site. However, rather than re-collecting identical pages, the crawler is served with different framing ads upon each access, which introduces near-duplicates into the crawl. Such archive regions of...
sites are not easy to identify and avoid in a large scale collection operation, because each site develops its own site naming conventions. A duplicate detection algorithm is considered brittle if it is too sensitive to exact duplicates and cannot detect near duplicates with minor variations. One of the key contributions of this paper is to propose a near duplicate detection algorithm that does not require any pre-processing with respect to stripping out HTML, ads, scripts etc, to work well. Although near duplicate detection in the presence of ad/HTML noise is an interesting theoretical exercise in itself, we believe that there are practical benefits to such an approach. Firstly we are less prone to brittleness of the signatures. Secondly, we save on computational cycles and time spent in extracting and separating the content of the page from the formatting by methods like DOM or SAX parsing. Several approaches have been developed for the detection of near-duplicates. These methods generally fall into three categories: shingling, clustering, and signature based. We sketch these approaches in our related work section. Our algorithm is most closely related to signature based work. We describe our algorithm and identify the research questions that our approach induces in Section 3. In Section 4 we discuss performance of our approach in terms of running time. In Section 5 we analyze the impact of various SpotSigs parameter settings on precision and recall, and explain the tradeoffs involved. We end with a discussion of future work.

2. RELATED WORK

The problem of near duplicate detection of documents in general, and Web pages in particular, has been well studied, and a variety of approaches have been proposed. These approaches can broadly be classified into the following categories.

- Shingling based methods
- Clustering based methods
- Fingerprinting/Signature based methods

Broder et al. [2] proposed shingling as a method to detect near-duplicates by computing a sketch of the document. A subset of shingles, or n-grams from each document is chosen as its sketch, and similarity between two documents is computed based on the Jaccard overlap measure (size of intersection/size of union) between document sketches. To reduce the complexity of shingling for processing large collections, ‘super shingling’ was proposed by Broder [3], which makes use of meta-sketches, or sketches of sketches. Here, documents with matching super shingles have a sequence of sketches in common. Recently, Henzinger [8] combined two algorithms for detecting near-duplicate Web pages, namely Broder et al.’s shingling, and Charikar’s [4] random projection algorithm. Henzinger improved on precision compared to using the constituent algorithms individually.

Clustering based schemes have also been proposed, although these are computationally expensive. Generally, some form of semi-supervised clustering is performed, which allows for incorporating additional knowledge [10, 15]. Such approaches typically focus on ways to introduce conditions that constrain the clustering process on the fly, such as by initializing clusters using constraints in the k-means algorithm, or controlling cluster assignments based on constraints [15]. Incorporating information about the document attributes and content structure into the clustering process to form near duplicate clusters [17] has also been suggested.

Another scheme for detecting similar documents is fingerprinting based on work by Manber [12], and subsequent work by Brin, Davis and Garcia-Molina [1], and Garcia-Molina and Shivakumar [13,14]. A document fingerprint, or signature, is typically a collection of integers that represent some key content in the document. In most signature schemes, words or sentences from documents are hashed to generate the signature. However, signatures generated from the whole document would make the system very brittle with respect to minor variations in content. The schemes differ in how they select the words for generating signatures, and in how they then compute the signatures.

Some of the methods for string selection are position-based, hash-value based, anchor-based and frequency-based. Position-based schemes [1] select strings based on their offset in a document. Hash-value based schemes like [2] pick strings whose hash values are multiples of an integer. An anchor-based approach [12] makes use of an anchor which is a string of characters. A set of anchors is selected beforehand. From the documents, a stream of characters of fixed length is extracted from places where these anchors occur and a checksum is computed over these. This approach is closest in spirit to our work. Fixed strings are used for document ‘synchronization’ in both approaches. In the main body of [12], however, all possible sets of n-length substrings are extracted, which makes [12] significantly more complex than SpotSigs. No timing and precision/recall analysis is offered in [12] for the simple anchor+one-word method that SpotSigs uses. A frequency based approach would select strings based on their frequency of occurrence in the collection. Conrad et al. [6] and Chowdhury et al. [5] choose word strings with high idf. I-Match [11] made use of external collection statistics and improved on recall by introducing multiple fingerprints.

SpotSigs, as will be explained in the next section is also a signature scheme which varies in the nature of generation of the fingerprints or signatures.

3. THE SPOTSIGS ALGORITHM

As indicated in Section 2, SpotSigs is based on signatures. In contrast to other approaches, SpotSigs creates for each document a set of localized document spot signatures that are distributed across the document. Document similarity is then determined by the degree of overlap among the spot signatures. This degree of overlap is judged by the familiar Jaccard measure, which will be explained in more detail in the following section.

3.1 Concepts and Notation

The points in the text at which spot signatures are generated are all the locations in the document where one of a previously chosen set of words occurs. We call the words in this set antecedents. These are chosen to be frequent within the corpus. Typical choices are stop words, like is, or the, which are likely to occur in every document and whose occurrences are distributed widely within each document.

The spot signature of a location in a document consists very simply of the word that follows the antecedent at a fixed spot distance d. Each antecedent is associated with its own spot distance. We use the notation a(d) to denote a spot signature that
is computed by finding the \( d^{th} \) word after the occurrence of the antecedent \( a \). For example, \( \text{that}(4) \) denotes a spot signature that is computed wherever the antecedent \( \text{that} \) occurs in a document. The signature consists of the fourth word after the occurrence of this antecedent.

We call the quantity \( \{a(d)\}(docID) \) a spot set. This is the set of all spot signatures with antecedent \( a \) and spot distance \( d \) in the document with \( id \ docID \). We may apply multiple types of spot signatures to a single document. For example, \( \text{is}(3), \text{that}(5), \text{is}(2) \) are all valid spot signature specifications that can be applied to a single document. Figure 4 shows a simplified example of three documents, \( A, B, \) and \( C \).

![Figure 4: Three documents (A, B, C) with spot signatures](image)

Document \( A \) and \( B \) share the same layout, while \( C \) has no sidebar and different placement of a text box. All three documents feature advertising at the top. Below the figure we list the spot sets for each document. Note an important condition in \( \{\text{the}(4)\}(A) \). This spot set contains the word \( \text{already} \), which is the fourth word after \( \text{the} \) in the text box above the Raiders sentence. The spot distance was large enough, and the antecedent close enough to the bottom of the self-contained text element that the signature computation ‘spilled over’ from one layout element to the next. Such spill-over is often undesirable, because layout elements often correspond to coherent semantic units.

When evaluating whether document \( B \) is a near-duplicate of document \( A \), the Jaccard overlap in the spot signatures serves as a confidence measure.

\[
c = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}
\]

The vertical bars denote set cardinality. The SpotSigs algorithm considers two documents to be near-duplicates if their confidence measure is larger than a chosen threshold \( t \).

The parameters of the SpotSigs algorithm imply a number of research questions.

**Research questions:**
- Best number of antecedents to use?
- Best spot distances for each antecedent?
- Best confidence threshold?
- Best choice of antecedents?

We did not have prior hypotheses for the best number of antecedents to use. For the confidence thresholds we expected that very high thresholds would negatively impact recall, but would maximize precision. Our hypotheses for the other parameters were as follows. We expected that we would benefit from using very common words for the antecedents to ensure that all documents would have non-empty signatures, and that the spot signatures would be distributed somewhat evenly across each document. Using commonly occurring words like stop words for the antecedents also ensures that we need not do anything specific to our data to select the antecedents.

On the other hand, a large spot distance harbors the danger of frequently spanning across Semantic units in the text, as in the example of Figure 4 above. We therefore expected that a small spot distance of 2 or 3 would be optimal.

We also experimented with using more than one level of chaining for the signature generation. We call this the Multi-Chain variant. A two chain instance of this Multi-Chain approach would involve using spot signatures of the form \( a(m,n) \). Here the spot signature is a word pair consisting of the \( m^{th} \) and the \( (m+n)^{th} \) word after the occurrence of the antecedent \( a \). The number of levels of chaining corresponds to the number of spot distances used.

For example, an antecedent for the two chain version would be \( \text{is}(3,4) \). Here the signature would involve the word following \( \text{is} \) three words later and the word following \( \text{is} \) seven words later \((3+4)\).

This approach could be extended to more levels of chaining but then we would run into similar problems of spanning semantic units in text as before.

### 3.2 Experimental Setup

We created a corpus of same-core duplicate news pages from the World-Wide Web. Our procedure for collecting these pages operated as follows. We selected an article on the Google News search site. From this article we selected a small paragraph and entered that snippet as a new query into the site. This process often generated near-duplicates, some of which were hidden by Google for the convenience of normal users of the site. Those near duplicates are available by following a respective link. However, not all these near duplicates are same-core near duplicates. Google considered some of them near duplicates based on their overlap in vocabulary with other results already shown. The near duplicates were then manually inspected and downloaded. Manual inspection was necessary to include just the same-core near duplicates for our experiment. Even with such manual inspection we had occasional cases of mistakes were documents that were not same core duplicates were accidentally included.
The resulting corpus consisted of 2168 documents spread across 67 sets of same-core near-duplicates. Each such cluster or equivalence set contains a varying number of documents. For some of our experiments we worked with a subset of this data set with 1397 documents comprising 43 clusters.

This well-understood gold standard collection enables us to compute both, precision and recall. We define precision in the usual way as follows,

\[
\text{Precision} = \frac{\# \text{of true duplicates detected}}{\# \text{of duplicates detected}}
\]

We have a stricter semantics for recall than the usual definition. We define it as follows,

\[
\text{Recall} = \frac{\# \text{of duplicates detected correctly}}{\# \text{of duplicates in the collection}}
\]

A duplicate document \(d\) is said to be detected correctly when its identified duplicate document \(d'\), belongs to the same cluster as \(d\) in our gold data set. This stricter measure is necessary for the following reason. Our gold data set contains at least one duplicate \(d'\) for every document \(d\). The algorithm might incorrectly identify \(d\), that is, a document not in a different equivalence set than \(d\), as a duplicate. Thus, even though the algorithm would in this case properly claim the presence of a duplicate, this conclusion is correct only by accident. We would not want recall to be credited in this case.

### 3.3 Matching Procedures

We have described the basics of the SpotSigs algorithm. Since SpotSigs is a signature based scheme that uses hashing, two documents that are duplicates of each other are said to collide. But we have not specified how all the documents in a Web collection should be compared to each other. Multiple options are available for how to proceed. The approaches we discuss differ along two dimensions. On the one hand, the approaches vary based on how a document’s duplicate can be identified from the potential set of colliding documents. Here we study First Guess and Best Guess approaches. On the other hand, approaches vary based on the number of passes that need to be made through the data. Based on this criterion we study three methods namely Full Matching, Incremental Matching and Seeded Matching.

#### 3.3.1 First Guess vs. Best Guess Approaches

There are two simple approaches for identifying a document’s duplicate from a set of documents that the document collided with.

In the First Guess approach, as soon as a document collides with another document that we have seen in the past, if its Jaccard overlap in spot signatures is greater than the confidence threshold, we declare it a duplicate. In the Best Guess approach we compute the Jaccard overlap score for all documents that we collide with, and which also cross our confidence threshold, but report only the document with which the highest Jaccard similarity is obtained as the duplicate. As can be expected, this approach boosts the precision, particularly at lower confidence levels. This is because for lower confidence levels with the First Guess approach, there is a higher chance of picking a false positive even when a true duplicate exists among the colliding documents. Recall is also higher for the Best Guess approach for the same reasons.

The precision and recall numbers for both approaches are shown in Figures 5, 6, 9 and 10.

### 3.3.2 Full Matching

The following sections discuss options for how many documents are compared to one another throughout the near-duplicate detection process.

The Full Matching method is to first scan all documents and generate a data structure that contains all document signatures.

We call this data structure the index. We then make a second pass through the documents for the matching phase. During that phase, for each document \(d\), a set of documents are identified that have at least one spot signature in common with \(d\). If even just one document in this set has a Jaccard overlap higher than the current threshold setting we declare \(d\) to be a duplicate. Full Matching is thus a two pass algorithm.

This approach is quite slow because of the potentially large number of documents we end up comparing each document with.

We discuss these speed performance issues later. The space requirement also increases under the Full Matching regime, since we have all documents in our index.

![Figure 5: Variation of Precision with Confidence for Full Match version of SpotSigs](image)
3.3.3 Incremental Matching

A likely lower performance, but less expensive approach is the following. We build an index one document at a time. The index maps each antecedent to a list of docID/spot signature pairs. That is, given an antecedent we can determine all spot signatures and the documents in which they occurred. Figure 7 shows this simple structure for our running example. Initially, one document is inserted into this index. Subsequently other documents are added to the index, unless they are believed to be duplicates of documents that are already indexed.

that)

is(3) \rightarrow ((\text{Pants}, A), (\text{year}, B), (\text{this}, B), (\text{Pants}, C))

is(2) \rightarrow ((\text{here}, A), (\text{this}, B), (\text{expected}, B), (\text{here}, C))

the(4) \rightarrow ((\text{already}, A), (\text{knowing}, A), (\text{to}, A), (\text{passes}, C), (\text{knowing}, C))

Figure 7: Example spot signature index for matching

That is, before adding a document to the index we check whether any of its spot signatures is already in the index for the respective antecedent. If we find a match we compute the confidence score between all spot signatures in all antecedents of the new document against the spot signatures of the colliding document that is already in the index (using for efficiency a different lookup index that we do not elaborate on here). If the confidence computation (Jaccard similarity) exceeds our threshold, we do not index the new document but set it aside as a duplicate of the already indexed document. We check the new document this way against all collisions with other already indexed documents. Incremental Matching is thus a one pass algorithm. Figure 8 shows pseudo code that sketches this process.

// Index data structure:
New Hashtable index(antecedent->list([spotSig, docID]));
Foreach document In randomized(documents) {
Foreach antecedent In antecedents
    spotSigs[antecedent] := computeSpotSigs(document);
Foreach spotSig In spotSigs {
    // Does any previously indexed document share this spotSig?
    Foreach indexedDoc In (collisions(index(spotSig))) {
        Compare all remaining spot sigs of document with
        spot sigs in indexedDoc and compute confidence;
        If (confidence > threshold) {
            declare document to be duplicate of indexedDoc;
        } // Do not add document to index
    } // Next indexedDoc
} // Next spotSig
// No collisions for any spot signature in document:
Foreach antecedent In antecedents
    index.add(antecedent, spotSigs[antecedent])
} // Next document

Figure 8: Pseudo code for incremental match algorithm

Note that the index can be implemented as a hashtable so that spot signature replication is easily detected via collisions.

The advantage of this approach is its decreased running time. The index stays smaller and documents are not compared against all documents they can potentially collide with. The algorithm’s drawback, however, is that it is more sensitive to the confidence threshold in that it does not afford an opportunity to find the best match among all document pairs. If a document qualifies as a duplicate early, before the best matched document has been inserted into the index, then an inferior match will be registered.

Figures 9 and 10 show the precision and recall curves with varying confidence threshold settings for this approach. The behavior of the curves is similar to the earlier Full Matching approach.

// Index data structure:
New Hashtable index(antecedent->list([spotSig, docID]));
Foreach document In randomized(documents) {
Foreach antecedent In antecedents
    spotSigs[antecedent] := computeSpotSigs(document);
Foreach spotSig In spotSigs {
    // Does any previously indexed document share this spotSig?
    Foreach indexedDoc In (collisions(index(spotSig))) {
        Compare all remaining spot sigs of document with
        spot sigs in indexedDoc and compute confidence;
        If (confidence > threshold) {
            declare document to be duplicate of indexedDoc;
        } // Do not add document to index
    } // Next indexedDoc
} // Next spotSig
// No collisions for any spot signature in document:
Foreach antecedent In antecedents
    index.add(antecedent, spotSigs[antecedent])
} // Next document

Figure 9: Variation of Precision with Confidence for Incremental Match version of SpotSigs

Figure 8: Pseudo code for incremental match algorithm

Note that the index can be implemented as a hashtable so that spot signature replication is easily detected via collisions.

The advantage of this approach is its decreased running time. The index stays smaller and documents are not compared against all documents they can potentially collide with. The algorithm’s drawback, however, is that it is more sensitive to the confidence threshold in that it does not afford an opportunity to find the best match among all document pairs. If a document qualifies as a duplicate early, before the best matched document has been inserted into the index, then an inferior match will be registered.

Figures 9 and 10 show the precision and recall curves with varying confidence threshold settings for this approach. The behavior of the curves is similar to the earlier Full Matching approach.

// Index data structure:
New Hashtable index(antecedent->list([spotSig, docID]));
Foreach document In randomized(documents) {
Foreach antecedent In antecedents
    spotSigs[antecedent] := computeSpotSigs(document);
Foreach spotSig In spotSigs {
    // Does any previously indexed document share this spotSig?
    Foreach indexedDoc In (collisions(index(spotSig))) {
        Compare all remaining spot sigs of document with
        spot sigs in indexedDoc and compute confidence;
        If (confidence > threshold) {
            declare document to be duplicate of indexedDoc;
        } // Do not add document to index
    } // Next indexedDoc
} // Next spotSig
// No collisions for any spot signature in document:
Foreach antecedent In antecedents
    index.add(antecedent, spotSigs[antecedent])
} // Next document

Figure 9: Variation of Precision with Confidence for Incremental Match version of SpotSigs

Figure 8: Pseudo code for incremental match algorithm

Note that the index can be implemented as a hashtable so that spot signature replication is easily detected via collisions.

The advantage of this approach is its decreased running time. The index stays smaller and documents are not compared against all documents they can potentially collide with. The algorithm’s drawback, however, is that it is more sensitive to the confidence threshold in that it does not afford an opportunity to find the best match among all document pairs. If a document qualifies as a duplicate early, before the best matched document has been inserted into the index, then an inferior match will be registered.

Figures 9 and 10 show the precision and recall curves with varying confidence threshold settings for this approach. The behavior of the curves is similar to the earlier Full Matching approach.
A side by side comparison of the best precision and recall numbers achieved by these variants is shown in Figures 11 and 12. We are presently investigating the reason for the poorer performance of the Full Matching version which appears to be a counter intuitive result.

3.3.4 Seeded Matching
A third method of matching, seeded matching, is a compromise between full matching and the incremental approach. Seeded matching can be used when examples of Web pages in each set of duplicates are available a priori. This might be the case when a collection contains specifications for a known set of products, or when one example among a limited number of news stories has been retrieved before matching begins.

Seeded matching inserts these examples into the index at the outset. After this initialization the matching process proceeds as in incremental matching.

These matching methods induce two research questions beyond the ones listed earlier:

Additional research questions:
- What is the performance difference between incremental and seeded matching?
- How are recall/precision performance affected by index size?

Our experiments revealed that seeded matching did not offer gains over incremental matching in precision or recall.

3.3.5 Adding Duplicate Documents to Index
In the previous incremental indexing approach we retained just unique documents during our pass through the data. One hypothesis is that in terms of near duplicate detection it might be good to have information about the duplicates as well in the index. This would ensure that we can catch a wider range of near duplicates. Our experiments however revealed no noticeable improvement in precision or recall under this regime, while running time increased.

3.3.6 Multi-Chain Performance
We found that having more than one level of chaining increased precision at lower levels of confidence.

Figure 13 shows the difference in precision between single-chain and multi-chain SpotSigs variants. At higher levels of the confidence threshold (not shown) both precision and recall were comparable. The increase in precision can be explained by the
stricter conditions for collisions. Recall also improved for the lower confidence settings as shown in Figure 14. The multi-chain approach could of course be extended to more than two levels but it would be a case of trading precision for recall.

Figure 14: Comparison of Recall between Single Chain and Multi-Chain SpotSigs variants

4. PERFORMANCE CONSIDERATIONS

In this section, we look at the running times of the SpotSigs variants to better understand the associated tradeoffs.

Running time is dominated by the number of collisions encountered during the matching process. This interdependence is particularly evident for high values of the confidence threshold. For those values, many documents are judged to be unique, and are added to the index. This bloating of the index in turn increases the potential number of colliding documents and thus the time taken per document.

Figure 15 shows the correlation between the number of colliding documents and the time taken to index a document for the incremental match version.

Speed optimizations can therefore aim to reduce the potential number of documents that can cause collisions and hence the need to compute a similarity score. Towards one such optimization we store the number of spot signatures in each document as an additional piece of information in the hashtable.

This simple optimization is best explained with an example. Say we have a confidence threshold setting C. As per the SpotSigs algorithm we are only interested in documents whose Jaccard similarity cross this threshold. Say we are currently processing document X and we have collided with document Y. Let the number of spot signatures that exist in X be \( N_X \) and the number of spot signatures that exist in Y be \( N_Y \).

We know that the best possible value for the Jaccard similarity between the two documents is the following,

\[
BestSim(X, Y) = \frac{\min(N_X \cdot N_Y)}{\max(N_X \cdot N_Y)}
\]

If we find that the best case similarity score between X and Y is smaller than our confidence threshold C, we can drop Y from the set of colliding documents since there is no way X will be a duplicate of Y with the current confidence threshold setting. As can be expected, the savings are particularly significant for higher values of confidence since these typically have a lot of colliding documents and the best case similarity score for many of them fail to cross the confidence threshold.

Figures 16 and 17 show the improvement in running time that this optimization provides in the case of Incremental Matching and Full Matching.

Table 1 shows the percentage gain in running time that the optimizations provide for both incremental matching and full matching. In general, the percentage improvements were higher for the Full Matching version. The gain also increases with the confidence setting as explained before.

Figure 18 shows the running times when optimizing the different versions of SpotSigs that were discussed in section 3. Not surprisingly, the Full Matching versions continue to be more expensive than the Incremental Matching versions and Best Guess is more expensive than First Guess. One of our biggest gains with the optimization is that now, efficiency when tuning SpotSigs towards higher precision by operating at high confidence levels is improved. This bias towards precision matches the application goals in our work with the sociologists.
The described optimization interacts with chaining as follows. Given the strong correlation between running time and the number of collisions, we show in Figure 19 the difference in the number of colliding documents between the single-chain and multi-chain approaches of Incremental Matching.

Figure 20 then shows the improvements in running time that the multi-chain version allows. Both variants shown in Figure 20 are the optimized versions. These improvements are particularly significant for the mid range of confidence (~0.4-0.7). In general, as explained earlier, with increase in the confidence threshold, the number of unique documents added to the index increases. This increases the potential number of documents that cause collisions. For this reason the left half of Figure 20 shows low running times that increase with the confidence threshold.

Beyond the mid-point the optimization begins to impart a tempering effect, counteracting the increase in collisions. In the middle band of confidence settings, it is only the multi-chain setting that keeps the running time low (without compromising precision and recall).

Table 1: Percentage improvement in running time of optimized versions of Incremental Matching and Full Matching

<table>
<thead>
<tr>
<th>Confidence</th>
<th>% Improvement in running time of Optimized Incremental matching</th>
<th>% Improvement in running time of Optimized Full matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>1.238096</td>
<td>23.68074</td>
</tr>
<tr>
<td>0.2</td>
<td>4.709382</td>
<td>12.41286</td>
</tr>
<tr>
<td>0.3</td>
<td>17.75743</td>
<td>-7.46832</td>
</tr>
<tr>
<td>0.4</td>
<td>18.59757</td>
<td>24.99871</td>
</tr>
<tr>
<td>0.5</td>
<td>23.3458</td>
<td>32.17989</td>
</tr>
<tr>
<td>0.6</td>
<td>32.14612</td>
<td>48.9759</td>
</tr>
<tr>
<td>0.7</td>
<td>40.99486</td>
<td>65.79285</td>
</tr>
<tr>
<td>0.8</td>
<td>48.14954</td>
<td>78.4167</td>
</tr>
<tr>
<td>0.9</td>
<td>56.61014</td>
<td>89.0032</td>
</tr>
<tr>
<td>1</td>
<td>50.01067</td>
<td>98.07083</td>
</tr>
</tbody>
</table>

Figure 19: Comparison of colliding documents between Single Chain and Multi Chain variants using Incremental Matching

Figure 18: Comparison of Running Times for different optimized variants of SpotSigs
5. Basic SPOTSIGS PARAMETER SETTINGs

In this section we present a more detailed analysis of the various tradeoffs that go with the different parameter settings for SpotSigs. For these experiments we used a subset of our earlier data set involving 1397 documents with 43 clusters.

We begin with an exploration of how the number of antecedents impacts recall and precision.

5.1 Number of Antecedents

Figure 21 shows the result of incremental match runs with a fixed spot distance of 3 and confidence threshold 0.7. Precision and recall are shown with the number of antecedents varying between one and seven.

Note the precision increase between two and three antecedents when said is added to is and the. Note as well the almost constant precision as additional antecedents are added beyond this point.

We spot checked sensitivity of the system to the choice of antecedents. For this experiment we moved the addition of said from its current location to the far right of the graph. That is, we withheld this antecedent until the last run. The result was a drop of precision for runs three through six, down to the level of run two. That is, the precision increase was specifically due to the choice of said as antecedent.

Figure 21: Precision and recall for different numbers of antecedents

Once this antecedent was present, the other antecedents we added barely impacted performance. All the antecedents that we chose to use were ones we believed were reasonably frequent in the corpus (since this was a news corpus ‘said’ was thought to be reasonable). In this study we explicitly did not want to undertake more corpus specific tuning by collecting detailed term distribution statistics, because such tuning would increase complexity and impact scalability. Nevertheless, investigation into such tuning would likely improve precision and recall and would be a candidate for a separate study.

5.2 Spot Distances

We next present results from our spot distance investigation in Figure 22. We experimented with several spot distances, running incremental matching for each setting. In order to avoid bias in the order of documents that we matched, we picked each new document randomly from among our 1397 item corpus. This procedure implies that documents entered the index in unpredictable order. For any given run different documents will therefore be compared against each other. This scheme matches actual applications in that any given archive that social science analysis tools will be used to manipulate, will generally be organized in unknown or even changing order.

The randomized approach, however, also introduces uncertainty into our measurements. We therefore repeated most measurements three times and present the mean of the three runs.

Figure 22 shows the averages of twelve spot distance values below 100, and of distances from 100 to 400. As spot distances grow, a clear downward trend of both precision and recall are evident.
5.3 Index Size

We examined two aspects of index size. The first is how the confidence setting threshold affects the size of the index. The second is how the index size affects precision.

5.3.1 Index Size and Confidence Threshold

As we increase the confidence threshold, more documents are rejected and taken as unique. Recall that with Incremental Matching only documents that are deemed unique are added to the index. Figure 23 illustrates how this behaviour plays out as all of the 1397 documents are in turn compared to the existing index content, and are added or marked as duplicates as appropriate. As mentioned above, we ran these measurements three times and show the mean in Figure 23. We measured the index size in units of documents every 100 document-matching steps. Once all 1397 documents were matched, the high-confidence index contained over 400 documents, while for the low-confidence case index size was below 200.

5.3.2 Index Size and Precision

We measured how precision is impacted by the growing size of the index. In Figure 24 we see how the number of false positives in each batch of 200 documents decreases as more documents are seen in Incremental Matching. Figure 25 is a closely related graph which shows all mistakes made in each batch of 100 documents that are indexed. Here, mistakes refer to both false positives as well as false negatives. We believe that the reduction in mistakes could be because towards the end most of the difficult to detect near duplicates are in the index and hence we have better coverage with respect to detecting near duplicate variants.
Disappointing recall in our context means that the human content inspectors are burdened with more pages to judge than if the requirements. For our application—content coding tools for sociologists—precision is more important than recall. Given the choice, precision is therefore to be favored. Our optimized SpotSigs version is particularly helpful in manual social science analyses performed on the archives. We introduced the SpotSigs algorithm that attempts to identify the duplicates with manageable computation.

We considered the impact of several algorithm variants and parameter settings on recall, precision, running time, and space requirements. For our application—content coding tools for sociologists—precision is more important than recall. Disappearing recall in our context means that the human content inspectors are burdened with more pages to judge than if the system was functioning well. Lacking precision, in contrast, leads to incorrect coding. Given the choice, precision is therefore to be favored. Our optimized SpotSigs version is particularly helpful in this regard since it speeds up the running time of the higher confidence threshold setting. In cases where we want speed (reduced running time) at a minimal cost of precision, the multi-chain version at a lower confidence setting proves useful. Here is a brief summary of our results.

**Number and choice of antecedents:** The number of antecedents that appears to work well for high values of precision (>90%) and recall (>85%) appears to be 2 or 3. This low number is good because a smaller number implies manageable index size and overall complexity. However, as mentioned, few antecedents are only sufficient if they are well chosen for the collection. A well chosen set of antecedents is one that covers most areas of all documents in the corpus.

Although we show that corpus agnostic choice of antecedents work well, we believe that simple exploratory word distribution statistics over a corpus are likely to prove helpful for selecting antecedents. In the absence of such preparatory work the inclusion of, say two or three stop words as antecedents is advisable. For our dataset and choice of antecedents larger sets of antecedents did not increase precision very much after a point. Additional studies with tens of antecedents and maybe with other data sets would be helpful to increase confidence that no significant performance gains are induced by even larger antecedent sets than we tried in our experiments.

**Spot distance:** Our spot distance exploration confirmed our hypothesis that overly large distances are detrimental. We believe that the clear downward trend with increasing spot distance is due to signatures spanning beyond semantic units in the documents.

**Index size and its impact on performance:** We found that index size is highly sensitive to the confidence threshold. This behavior matches our intuition. As the threshold is increased, fewer pages are set aside as duplicates and are instead entered into the index.

**Impact of multiple levels of chaining on performance:** Using multiple levels of chaining offers improved precision and recall at lower values of the confidence threshold. More importantly it significantly speeds up the running time, the gains being particularly noticeable for confidence threshold settings in the middle of the range (0.4-0.7).

**Matching Algorithm findings:** We found that in general, Incremental Matching works very well and on our data set there did not seem to be any noticeable gains with precision or recall with using the Full Matching scheme. The Full Matching scheme is also much slower in terms of running time. In terms of identifying duplicates from the set of colliding documents, the Best Guess approach offers superior precision and recall to the First Guess approach. The gains are particularly pronounced at lower levels of the confidence threshold. The Best Guess approach is also much slower than the First Guess Approach. The optimization that we introduced to reduce the number of colliding documents allowed us a significant speedup for the higher confidence threshold setting.

A number of open questions remain. We plan to compare SpotSigs with other near-duplicate detection methods. Some of the references in the related work section above offer precision and recall measures, which are lower than ours. However, the measurements were taken over different data sets, preventing a direct comparison.

Beyond these tasks more radical explorations offer opportunities. For example, we could attempt to partition Web pages in an attempt to identify structure, much as OCR programs parse page layout. Different duplicate detection algorithms could then be applied to the fragments, depending on their length and content. We also plan to introduce methods for distinguishing same-core from same-frame near-duplicates.

We plan to investigate approaches that start with some set of antecedents, and grow that set of antecedents at runtime during the first pass through the data. The hope is to do well on pages which currently contain very few antecedents.

As a variant of collection statistics based choices of antecedents we are exploring differential weighting of spot signatures based on the IDF score of the consequent terms. Some site specific machine learning might be of use in adjusting spot signature settings differentially in the sections of each document: certain sections of pages might be recognizable as ads or other page fragment genres for which specific settings are optimal.

Near-duplicate Web pages are difficult to detect. But successful analysis of Web archives by the social sciences will require human inspection of at least some portions of the archives. Near-duplicate content makes such investigations extremely tedious and expensive. As history increasingly manifests on the World-Wide Web, the problem of studying its archives looms. Near-duplicate
detection, while prosaic, is proving to be an important roadblock to progress in this area. SpotSigs looks to be an effective, simple to implement mechanism for grouping near-duplicates so that investigators can inspect and code them as sets.

7. ACKNOWLEDGEMENTS
I would like to thank Dr. Andreas Paepcke and Prof. Hector Garcia-Molina of the Stanford InfoLab for their constant support and invaluable guidance.

8. REFERENCES


