SpotSigs: Robust and Efficient Near Duplicate Detection in Large Web Collections

Martin Theobald Jonathan Siddharth Andreas Paepcke
Stanford University
Department of Computer Science
353 Serra Mall, Stanford CA-94305
{theobald,siddharth.jonathan,paepcke}@cs.stanford.edu

ABSTRACT

Motivated by our work with political scientists who need to manually analyze large Web archives of news sites, we present *SpotSigs*, a new algorithm for extracting and matching signatures for near duplicate detection in large Web crawls. Our spot signatures are designed to favor natural-language portions of Web pages over advertisements and navigational bars.

The contributions of SpotSigs are twofold: 1) by combining stopword antecedents with short chains of adjacent content terms, we create *robust* document signatures with a natural ability to filter out noisy components of Web pages that would otherwise distract pure n-gram-based approaches such as Shingling; 2) we provide an *exact* and *efficient*, *self-tuning* matching algorithm that exploits a novel combination of collection partitioning and inverted index pruning for high-dimensional similarity search. Experiments confirm an increase in combined precision and recall of more than 24 percent over state-of-the-art approaches such as Shingling or I-Match and up to a factor of 3 faster execution times than Locality Sensitive Hashing (LSH), over a demonstrative "Gold Set" of manually assessed near-duplicate news articles as well as the TREC WT10g Web collection.

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

General Terms: Algorithms, Experimentation, Performance, Reliability.

Keywords: Stopword Signatures, High-dimensional Similarity Search, Optimal Partitioning, Inverted Index Pruning.

1. INTRODUCTION

Detecting near-duplicate documents and records in large data sets is a long-standing problem. Syntactically, near duplicates are pairs of items that are very similar along some dimensions, but different enough that simple byte-by-byte comparisons fail. On the World Wide Web, document duplication is particularly problematic, and in fact any of the major search engines attempts to eliminate highly similar documents from its search results.

Moreover, we encountered the problem in the related context of our Web Sociologists Workbench [26]. This project constructs tools for political scientists, historians, and sociologists to help them analyze large Web archives. Our users' first project was to analyze 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 by content semantics, such as unions, feminism, celebrity, or local issue. These semantic differences were subtle enough that machine learning algorithms of sufficient accuracy could not readily be constructed. Human coders went to work on this collection, using a specially constructed visual tagging tool. Unfortunately, they failed to complete the task. Excessive amounts of near duplicates overwhelmed their efforts.

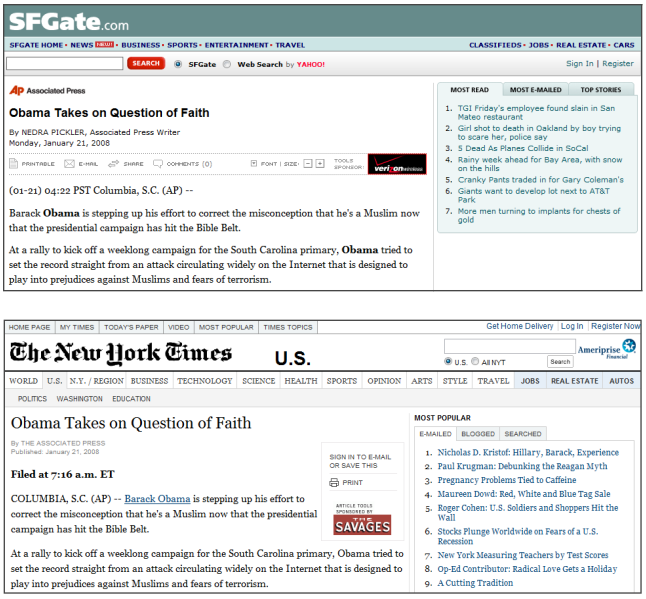

Figure 1: Near-duplicate Web pages 1): identical core content with different framing and banners (additional ads removed).

There are in fact two types of near duplicates for this application. Figure 1 shows a pair of same-core Web pages that only differs in the framing, advertisements, and navigational banners added each by the San Francisco Chronicle and New York Times. Both articles exhibit almost identical core contents, reporting on presidential candidate Obama's take on faith during the 2008 U.S. pre-election phase, because both were delivered by Associated Press. Figure 2 is an example of the opposite case from Yahoo! Finance, showing two daily summaries of the NASDAQ and Dow Jones indexes. In particular for domains like stock markets, news sites often use very uniform layouts and the actual contents-of-interest only constitute a fraction of the page. Hence, though visually even more similar than the pair in Figure 1, the pair in Fig-

SIGIR'08, July 20–24, 2008, Singapore.

ure 2 should not be identified as near duplicates. Typically, our sociologists would only consider Figure 1's same-core pair to be near duplicates, since the core articles are their focus. Near duplicates would not be discarded but could be collected into a common set which is then tagged in batch.

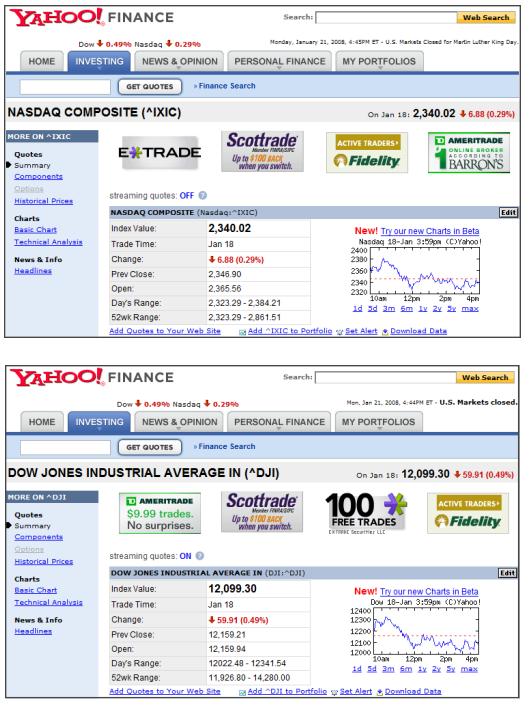

Figure 2: Near-duplicate Web pages 2): identical framing but slightly different core content.

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 pages of the same site. 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 all kinds of neighboring layout material. Algorithms for duplicate detection must therefore be particularly robust.

Even if care is taken to avoid exact 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, 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. However, rather than re-collecting identical pages, the crawler is served with different framing ads upon each access, which introduces near duplicates into the archive.

1.1 Contributions

- SpotSigs provides a robust scheme for extracting characteristic signatures from Web documents, thus aiming to filter natural-language text passages out of noisy Web page components. While our approach is similar to Shingling [7], it provides a more semantic pre-selection of shingles. The SpotSigs parser only needs a single pass over an incoming token stream, which is much more efficient, easier to implement, and less error-prone than sophisticated and inherently expensive tools for layout analysis, and it remains largely independent of the input format.
- We propose an exact and self-tuning, highly parallelizable algorithm for similarity search in high-dimensional feature spaces that yields no false negatives nor positives due to the matching procedure itself. Yet we are able to show very competitive runtimes to the fastest known but more error-prone hashing schemes like I-Match [12, 22] and Locality Sensitive Hashing (LSH) [20, 15]—if the similarity threshold is high. Moreover, SpotSigs creates clustered output in the form of all linked pairs of near duplicates.
- For low similarity thresholds, using our signature scheme still significantly improves recall of potentially faster but more "brittle" hashing approaches such as I-Match. It also allows for a more effective and robust parameterization of LSH which may decrease the amount of tuning required for these methods [1, 23].
- We provide a thorough experimental evaluation over a very challenging, manually curated "Gold Set" of near-duplicate news articles and over a large-scale Web corpus, comparing SpotSigs with state-of-the-art approaches such as Shingling and Jaccard/Cosine similarity for measuring document resemblance, as well as I-Match and LSH for efficient matching.

2. RELATED WORK

Broder et al. [7] proposed a Shingling algorithm, coined DSC, 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 common Jaccard overlap measure between these document sketches. To reduce the complexity of Shingling for processing large collections, the use of "super shingles" (DSC-SS) was later proposed by Broder in [5]. DSC-SS makes use of meta-sketches, i.e., sketches of sketches, with only a minor decrease in result precision. Hod and Zobel [17] investigate a variety of approaches for filtering good shingles, while Büttcher and Clarke [9] focus on information-theoretic measures such as Kullback-Leibler divergence in the more general context of search. Recently, Henzinger [16] combined two algorithms for detecting near-duplicate Web pages, namely Broder's DSC and Charikar's [10] random projection algorithm. Henzinger improved on precision compared to using the constituent algorithms individually. Moreover, sophisticated clustering schemes allow for incorporating additional knowledge in the form of explicit constraints to the clustering process [21, 29]. Incorporating information about document attributes or the content structure for near-duplicate clustering [30] has also been suggested.

Another scheme for detecting similar documents is fingerprinting based on work by Manber [24], and subsequent work by Brin, Davis and Garcia-Molina [27], Garcia-Molina and Shivakumar [28], and more recently by Manku et al. [25]. A document fingerprint is typically a collection of integers that represent some key content in the document, wherein hashes of words or entire sentences are concatenated to some bit string to generate a characteristic fingerprint of the document. These fingerprinting schemes vary in the specific hash function used, as well as in the choice of strings used for hashing. Hash-value-based schemes like [7] pick strings whose hash values are multiples of an integer. Position-based schemes [4], on the other hand, select strings based on their offset in a document. Conrad et al. [14] and Chowdhury et al. [12] choose word strings with high inverse document frequency (IDF) [2] using collection statistics. Their I-Match algorithm [12, 22] makes use of external collection statistics

and improved on recall by introducing multiple fingerprints (based on different lexica) per document.

Locality Sensitive Hashing (LSH), proposed by Indyk and Motwani [15, 20], is an approximate similarity search technique that scales to both large and high-dimensional data sets. LSH can be tuned by concatenating k signatures from each data object into a single hash value for high precision, and by combining matches over l such hashing steps—using independent hash functions—for good recall. Min-hashing investigated by [6, 13, 18] has the interesting property that the probability of a match (i.e., a hash collision) between two data objects exactly confirms to the Jaccard similarity of their feature sets, which allows LSH in combination with Min-hashing to converge quickly to the full recall over near-duplicate documents with high Jaccard similarity. More self-tuning, iterative LSH approaches like LSH-Tree [3] or Hamming-LSH [13], however, typically increase the number of signature extraction and hashing steps involved. While behaving significantly more robust than a single LSH step, the increased amount of hashing may become the actual delimiting factor for runtime performance.

3. SPOT SIGNATURE EXTRACTION

The points (or *"spots"*) in the page at which spot signatures are generated are all the locations where one out of a previously chosen set of anchor words occurs. We call the anchor words *antecedents*, which are typically chosen to be frequent within the corpus. The most obvious, largely domain-independent choices for natural-language text are *stopwords*, like *is*, *the*, *do*, *have* etc., which are likely to occur in every document and whose occurrences are distributed widely and uniformly within any snippet of natural-language text. Hence spot signatures are expected to achieve more semantic-driven document surrogates than other signature schemes in that they tend to occur mostly in the natural-language passages of Web documents and skip over advertisements, banners, and the navigational components.

3.1 Concepts and Notation

A *spot signature* s_j of a location in a document simply consists of a chain of words that follow an *antecedent* word a_j at a fixed *spot distance* d_j. We use the notation $a_j(d_j, c_j)$ to denote a spot signature that is computed by finding a contiguous *spot chain* of c_j words, each of which is not itself a stopword. For example, *the(2,3)* denotes a spot signature that is computed wherever the antecedent *the* occurs in a document. For a spot distance $d = 2$ and chain length $c = 3$, the signature would then consist of the second, forth and sixth word after the occurrence of this antecedent. If one of these words after the antecedent is itself a stopword, we move on to the next non-stopword and then continue building the chain. We also allow the chain to be cut off if the whole signature would exceed the boundary of the document and at least one non-stopword is found after the antecedent. Chains of spot signatures may also overlap if further antecedents are contained within the range of the previous chain.

We hence call the quantity $A = \{a_j(d_j, c_j)\}$ a *spot set*. We may apply multiple types of spot signatures to a single document. For example, *is(3,1)*, *that(5,2)*, and *is(2,1)* are all proper spot signature specifications that can be applied to each document. When evaluating whether document B is a near duplicate of document A, their spot set resemblance is computed using the common Jaccard similarity measure.

3.2 Signature Extraction Example

Consider the last sentence in each of the two pages of Figure 1: *"At a rally to kick off a weeklong campaign for the South Carolina primary, Obama tried to set the record straight from an attack circulating widely on the Internet that is designed to play into prejudices against Muslims and fears of terrorism."*

Choosing the articles *a*, *an*, *the* and the verb *is* as antecedents with a uniform spot distance of 1 and chain length of 2, we obtain the set of spot signatures $S = \{$*a:rally:kick*, *a:weeklong:campain*, *the:south:carolina*, *the:record:straight*, *an:attack:circulating*, *the:internet:designed*, *is:designed:play*$\}$, which already characterizes the core content of the page very well. Note that the banners in Figures 1 and 2 hardly contain any of these otherwise frequent antecedents. Moreover, for the stock market example in Figure 2, no signatures at all would be extracted, such that the algorithm would conservatively skip over considering this pair as duplicates.

4. SPOT SIGNATURE MATCHING

Our key observation is that for any given similarity threshold τ, we can provide tight upper bounds for the Jaccard similarity of two documents just by looking at a single document property each, e.g., by comparing the cardinality of their signature sets or the length of their signature vectors. This lets us drastically prune the search space between a) potentially matching documents and b) the vector dimensions at which we need to compare potential matches—if the similarity threshold is high. To address a), we show how to derive an *optimal partitioning* of the collection into buckets of potentially matching documents, which minimizes the size of the partitions while guaranteeing not to miss any potential match (thus ruling out false negatives and positives due to the matching procedure). As for b), we introduce an efficient approach for a two-dimensional *inverted index pruning* with early termination of the index traversals. Both steps are developed on the basis of the very same bounding approach for (multi-)set Jaccard similarity.

Even though the worst-case complexity of our algorithm remains quadratic, we empirically show that SpotSigs can outperform linear but more error-prone approaches such as LSH for sufficiently large ranges of similarity thresholds τ. In contrast to LSH or I-Match, our approach is exact and self-tuning as it automatically adapts to the best-possible partitioning of the spot signature sets for any given threshold τ, without the need for any further algorithm-specific tuning parameters.

4.1 Upper Bounds for Jaccard Similarity

4.1.1 Jaccard Similarity for Sets

Let $sim(A, B) = |A \cap B|/|A \cup B|$ be the default Jaccard similarity as defined over two sets A and B, each consisting of distinct spot signatures s_j in our case. A simple, yet tight upper bound for the Jaccard similarity is

$$sim(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad \leq \quad \frac{\min(|A|, |B|)}{\max(|A|, |B|)} \qquad (1)$$

since $|A \cap B| \leq \min(|A|, |B|)$ and $|A \cup B| \geq \max(|A|, |B|)$. Without loss of generality, for $|A| \leq |B|$, we thus find:

$$sim(A, B) \quad \leq \quad \frac{|A|}{|B|} \qquad (2)$$

We now switch to a vector representation of spot signatures for documents, where the vectors d_1 and d_2 correspond to the sets A and B, respectively. Two document vectors d_1, d_2 have similar length if $|d_1|/|d_2| \geq \tau$; hence only similar-length pairs can be near duplicates with respect to the predefined threshold τ. That is, the bound in Inequality (2) tells us that if we would like to consider only document pairs $\langle d_1, d_2 \rangle$ with similarity $sim(d_1, d_2) \geq \tau$, we can safely disregard any such pair where $sim(d_1, d_2) \leq |d_1|/|d_2| < \tau$. This

is equivalent to omitting those pairs $\langle d_1, d_2\rangle$ from the pairwise vector comparisons, where

$$|d_2| - |d_1| \quad > \quad (1-\tau)|d_2| \quad \text{for} \quad |d_1| \leq |d_2| \tag{3}$$

without ever inspecting their spot signature sets or performing any vector operations for comparison. This observation will be key for partitioning the collection, as it allows us to efficiently map documents into buckets of similar signature vector lengths in a single, linear pass through the collection.

4.1.2 Generalization for Multi-Sets

The default Jaccard similarity is defined over binary variables only, i.e., elements being present in a set or not. We are now interested in generalizing Jaccard to also take weighted variables, hence multi-sets of spot signatures, into account, continuing to maintain tight upper bounds for the similarity of two documents just given the lengths of their corresponding signature vectors. The rationale behind this is that multi-sets of signatures may characterize a document better, since they allow for capturing the frequency of individual signatures and hence the length of the document more accurately. For two multi-sets A and B, we define the weighted *multi-set generalization* of Jaccard to be

$$\widetilde{sim}(A,B) \quad = \quad \frac{\sum_{s_j \in A \cap B} \min\left(freq_A(s_j), freq_B(s_j)\right)}{\sum_{s_j \in A \cup B} \max\left(freq_A(s_j), freq_B(s_j)\right)} \tag{4}$$

where $freq_A(s_j)$ denotes the frequency of SpotSig s_j in the spot signature multi-set A. Then $\widetilde{sim}(A,B)$ can similarly to $sim(A,B)$ be upper-bounded by:

$$\widetilde{sim}(A,B) \leq \frac{\min\left(\sum_{s_j \in A} freq_A(s_j), \sum_{s_j \in B} freq_B(s_j)\right)}{\max\left(\sum_{s_j \in A} freq_A(s_j), \sum_{s_j \in B} freq_B(s_j)\right)} \tag{5}$$

Again without loss of generality, for $|d_1| = \sum_{s_j \in A} freq_A(s_j)$ and $|d_2| = \sum_{s_j \in B} freq_B(s_j)$, we get:

$$\widetilde{sim}(d_1,d_2) \quad \leq \quad \frac{|d_1|}{|d_2|} \quad \text{for} \quad |d_1| \leq |d_2| \tag{6}$$

Here, d_1 and d_2 are frequency-weighted vectors representing multi-sets of spot signatures A and B, which allows for a compact (and sparse) representation of these multi-sets. Note that for the special case when A and B are sets, the two Inequalities (2) and (6) provide exactly the same bound for sim and $\widetilde{sim}$, respectively; and in fact, both measures yield exactly the same similarity value in this case.

4.2 Optimal Partitioning

According to the pruning condition in Inequality (3), we may safely ignore all pairs of documents during the matching process whose SpotSig vectors exceed a certain difference in length. Consequently, before comparing any signature sets, we partition the documents into buckets of potentially matching pairs. Then only spot sets within the same or at most two subsequent buckets are near-duplicate candidates and need to be compared. We hence aim at finding a contiguous partitioning $[p_k, p_{k+1})$ over the discrete distribution of vector lengths $|d_i| \in [1, \rho)$, such that all pairs $\langle d_i, d_{i'}\rangle$ with similar vector lengths are "close" to each other. Note that it is not possible to eliminate all borderline cases for any arbitrary distribution of $|d_i|$ by a single contiguous partitioning $[p_k, p_{k+1})$—there could always be a borderline pair that is highly similar but is spread across two different partitions—but we can in fact find a partitioning such that any two similar documents can at most be located in two contiguous partitions. We thus define the following three conditions for an *optimal partitioning* of documents in a collection.

For a given range of vector lengths $|d_i| \in [1, \rho]$, find a partitioning $[p_k, p_{k+1})$ with $1 \leq p_k \leq \rho$ such that:

(A) For all pairs $\langle d_i, d_{i'}\rangle$ with $|d_i| \leq |d_{i'}|$, $|d_i|/|d_{i'}| \geq \tau$ and $|d_i| \in [p_k, p_{k+1})$, d_i and $d_{i'}$ belong to either the same partition or two subsequent partitions, i.e., $|d_{i'}| \in [p_{k-1}, p_{k+2})$ (no false negatives).

(B) There is no pair $\langle d_i, d_{i'}\rangle$ with $|d_i| \leq |d_{i'}|$, $|d_i|/|d_{i'}| < \tau$ and $|d_i| \in [p_k, p_{k+1})$, that is mapped into the same partition, i.e., $|d_{i'}| \notin [p_k, p_{k+1})$ (no false positives).

(C) The width of all partitions $p_{k+1} - p_k$ is minimized (i.e., the amount of partitions is maximized) for $1 \leq p_k \leq \rho$ with respect to conditions (A) and (B) (minimality).

While (A) and (B) would suggest finding the partitions by looking at all combinations of documents, which would be expensive and could yield multiple valid partitions, there is only one optimal partitioning that minimizes the partitions' widths as demanded by (C). This allows for a simple constructive (but approximate) solution, using the pruning condition as given by (3): *Starting with $p_0 = 1$, for any given p_k, choose p_{k+1} as the smallest integer $p_{k+1} > p_k$ such that $p_{k+1} - p_k > (1-\tau)p_{k+1}$, while $p_{k+1} \leq \rho$.*

Although the above series is not in closed-form, the p_k values can easily be found numerically by iterating over the integer range $[1, \rho)$. Then, according to (3), any SpotSig vector with length $|d_i| = p_k$ can at most have a distance to a similar and larger vector $d_{i'}$ of $|d_{i'}| - |d_i| < (1-\tau)|d_{i'}|$, which is the case when $|d_{i'}| < p_{k+1}$ and thus $|d_{i'}| \in [p_k,\ p_{k+1})$. Hence d_i and $d_{i'}$ will be located in the same partition in this case. More generally, for $|d_i| \in [p_k,\ p_{k+1})$, any larger SpotSig vector $d_{i'}$ with length difference of at most $|d_{i'}| - |d_i| < (1-\tau)|d_{i'}|$ will have at most a length of $|d_{i'}| < p_{k+2}$. Conversely, any shorter SpotSig vector $d_{i'}$ with length difference of at most $|d_i| - |d_{i'}| < (1-\tau)|d_i|$ will have at least a length of $|d_{i'}| \geq p_{k-1}$. Hence *any* possible pair of similar documents must either be within the same partition or within two immediate neighbor partitions as demanded by (A). Conversely, no pair of non-similar documents can be within the same partition as demanded by (B).

By minimizing the partition widths according to (C), we also minimize the number of documents being mapped into the same partition (and thus the number of potentially matching pairs). That is, for both variations of Jaccard similarity defined in Subsection 4.1, the partition boundaries $p_k \in [1, \rho)$ are merely a function of the threshold τ, regardless of the data. Note that this remains an approximate solution as there could still be a smaller partitioning that does not violate condition (B). However, with the above strategy, conditions (A) and (B) will be satisfied for *any* distribution of vector lengths $|d_i|$ as a consequence of the bound provided in (3), and the partitions can be created a priori without knowing the actual data. Furthermore, this approximation converges to—and finally reaches—the optimal solution as the distribution gets more dense, i.e., it is equal to the optimal solution when all distinct vector lengths $|d_i|$ within the given range $[1, \rho)$ occur in the collection.

Further note that all documents d_i, which exceed the expected range of spot signature lengths ρ, can be mapped into an additional partition $[\rho, \infty)$ for $|d_i| \geq \rho$. Then ρ is a fairly robust tuning parameter and can deliberately be chosen large (or with ρ being just above the maximum spot signature length of all documents in a given collection—if known a priori).

4.3 Inverted Index Pruning

In addition to the above partitioning approach, which breaks the overall quadratic runtime into much smaller sets of candidates for pairwise comparisons, we can further accelerate the deduplication step *within* the partitions through the use of auxiliary inverted indexes. Using inverted indexes for the actual deduplication instead of the vectors themselves immediately reduces the amount of pairwise vector compar-

isons to only those pairs within a partition that share at least one common spot signature. Furthermore, it allows for a novel and elegant form of index pruning, again exploiting the very basic similarity threshold as given by pruning condition (3).

That is, for each partition k and spot signature s_{ij} in document d_i, we store a list of pointers to all document vectors $d_{i'}$ containing s_{ij} (including d_i itself) sorted in descending order of vector lengths $|d_{i'}|$. Then, starting with the least frequent signature s_{ij} in d_i, we start processing these inverted lists until no yet unseen document $d_{i'}$ can still match d_i with high similarity, which lets us reduce the amount of necessary vector operations using several early breaking conditions. Altogether, the expected performance can be optimized by traversing the inverted lists for s_{ij} in ascending order of their length within each partition (hence by the document frequency of s_{ij} in the given partition k) and by processing multiple documents from different partitions in parallel, e.g., in random order of vector lengths. The exact behavior is captured by Algorithm 1. Note that in doing so, SpotSigs already generates clustered output in the form of a complete near-duplicate graph, with any document being connected to *all* of its near duplicates, which might otherwise require an expensive post-processing step.

4.3.1 SpotSigs Deduplication Algorithm

Algorithm 1 defines the exact behavior of the SpotSigs deduplication step, which is in principle a highly optimized nested loop with parallel processing of disjoint partitions and various break conditions.

Algorithm 1 SpotSigs Deduplication

```
1: Input: document vectors d_i with weighted spot signatures s_ij;
   partitions P with boundaries [p_k, p_{k+1}) and inverted lists list_kj
2: pairs ← ∅
3: for all d_i in random order of |d_i| using t threads in parallel do
4:    partition_k ← P.get(|d_i|)
5:    sort all s_ij ∈ d_i by asc. document frequency in partition_k
6:    δ_1 ← 0
7:    checked_i ← ∅
8:    for all s_ij ∈ d_i do
9:       list_kj ← partition_k.get(s_ij)
10:      δ_2 ← 0
11:      for all d_i' ∈ list_kj sorted by descending |d_i'| do
12:         δ_2 ← |d_i| − |d_i'|
13:         if d_i = d_i' or d_i' ∈ checked_i then
14:            continue
15:         else if δ_2 < 0 and δ_1 − δ_2 > (1 − τ)|d_i'| then
16:            continue
17:         else if δ_2 ≥ 0 and δ_1 + δ_2 > (1 − τ)|d_i| then
18:            break
19:         else if sim(d_i, d_i') ≥ τ then
20:            pairs ← pairs ∪ {⟨d_i, d_i'⟩}
21:            checked_i ← checked_i ∪ {d_i'}
22:         end if
23:      end for
24:      δ_1 ← δ_1 + freq_{d_i}(s_ij)
25:      if δ_1 ≥ (1 − τ)max{|d_i'|}_{d_i' ∈ partition_k − checked_i} then
26:         break
27:      end if
28:   end for
29:   if p_{k+1} − |d_i| ≤ (1 − τ)p_{k+1} then
30:      partition_k ← P.get(p_{k+1})
31:      goto 6
32:   end if
33: end for
34: return pairs
```

Here, the equality check $d_i = d_{i'}$ in line 13 may refer to both, equal documents identified by some id or memory location, or *exact duplicates* identified by a single checksum or hash code comparison. Furthermore, remembering the set of already checked documents $checked_i$ for each d_i in lines 13 and 21 avoids potentially redundant Jaccard similarity comparisons. These could otherwise arise due to multiple spot signatures pointing to the same document from different inverted lists.

Crucial for good runtime performance are the *continue* and *break* conditions in lines 16, 18, and 26. That is, we can apply a two-dimensional pruning of the inverted index traversal using two additional bounds δ_1 and δ_2 for the distance $||d_i| - |d_{i'}||$ of d_i to any yet unseen document $d_{i'}$ located in the same partition and/or inverted list. Thus, for each d_i, δ_1 is incremented by the frequency $freq(s_{ij})$ of signature s_{ij} in d_i whenever processing an index list $list_{kj}$ has been finished; and δ_2 is updated to $|d_i| - |d_{i'}|$ at each access to the next $d_{i'}$ in $list_{kj}$.

δ_1 then is the minimum-possible distance of d_i's length to any other, yet unseen document in the same partition; and δ_2 is the distance to any other, yet unseen document in the same inverted list. Both in turn provide lower-bounds for the actual distances, which gives rise to multiple early breaking conditions:

- $\delta_1 \leq |d_i| - |d_{i'}|$ for $d_{i'} \in partition_k$ and $d_{i'} \notin checked_i$ and $|d_{i'}| \leq |d_i|$ (line 26)
- $\delta_1 + \delta_2 \leq |d_i| - |d_{i'}|$ for $d_{i'} \in partition_k$ and $d_{i'} \in list_{kj}$ and $|d_{i'}| \leq |d_i|$ (line 18)
- $\delta_1 - \delta_2 \leq |d_{i'}| - |d_i|$ for $d_{i'} \in partition_k$ and $d_{i'} \in list_{kj}$ and $|d_{i'}| > |d_i|$ (line 16)

The iteration into the next neighbor partition through line 31 will occur at most once per document and hardly affects runtime in practice. Note that the similarity check for any $\langle d_i, d_{i'} \rangle$ pair is symmetrical, hence it suffices to check each document only against its right neighbor partition.

Further note that we sort all $s_{ij} \in d_i$ at most once by their local document frequency in partition k. Then the obtained ordering of signatures can be reused as approximation of the document frequencies in the next neighbor partition $k+1$ for the potential second iteration as well, which prevents us from the overhead of a second sorting. In practice, this strategy proved to outperform a sorting by global document frequencies, which would in turn be an approximation of the document frequencies in both the actual partition k as well as the neighbor partition $k+1$.

4.3.2 Deduplication Example

Suppose we have three documents $d_1 = \{s_1{:}5, s_2{:}4, s_3{:}4\}$, $d_2 = \{s_1{:}8, s_2{:}4\}$, and $d_3 = \{s_1{:}4, s_2{:}5, s_3{:}5\}$ with $|d_1| = 13$, $|d_2| = 12$, and $|d_3| = 14$, a threshold of $\tau = 0.8$, and all three documents are mapped into the same partition $[p_k, p_{k+1})$. We can now deduplicate, for example, d_1 using only a single pairwise Jaccard computation on top of the index structures as shown in Figure 3. Since s_3 is the least frequent signature in d_1 with respect to this partition, we first navigate into the inverted list for s_3 to find d_3 as a potential match. As this is the first list considered, we get $\delta_1 = 0$, and in comparison to d_3 we get $\delta_2 = |d_1| - |d_3| = -1$, such that none of the continue/break conditions hold yet. We thus compute $sim(d_1, d_3) = (4{+}4{+}4)/(5{+}5{+}5) = 0.8$ and indeed identify $\langle d_1, d_3 \rangle$ as a near-duplicate pair. Now, as we finish processing this list, we increment δ_1 by $freq_{d_1}(s_3) = 4$, such that the break condition in line 26 already holds, which terminates the index processing for d_1. Note that a subsequent processing of d_3 will not be dispensable, since d_3 might in turn be similar to documents that do not necessarily have to be similar to d_1.

5. EXPERIMENTS

5.1 Implementation and Setup

SpotSigs is implemented as a compact Java prototype in only about 500 lines of code. All experiments were performed on a dual Xeon-3000 quad-core CPU with 32 GB RAM. All processing steps for the SpotSigs, LSH, and I-

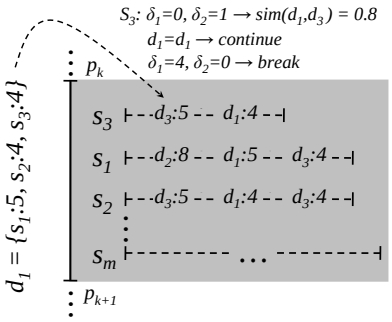

Figure 3: Threshold-based inverted index pruning.

Match implementations, like parsing, sorting, and clustering are multi-threaded, using 8 threads to fully utilize the machine's 8 physical cores.

As for effectiveness, we report micro-averages for precision, recall, and/or F1-measure [2] as quality measures. We distinguish *absolute* (i.e., user-perceived) precision and recall values obtained from manual judgments for the Gold Set, thus showing the advantage of our signature extraction, and *relative* recall evaluated against a synthetic recall base for the much larger TREC collection, thus primarily aiming to show the advantage of the efficient SpotSigs matcher.

For all runs, we consider normalized IDF values computed separately for each signature scheme and corpus as $idf_j = log(N/df_j)/log(N)$, where N is the corpus size, and df_j is the document frequency of signature s_j. The partitioning range ρ is fixed to 1,000 for all runs as well, creating a static partitioning of 501 partitions for $\tau = 1.0$, still 41 partitions for $\tau = 0.9$, and only 3 partitions for $\tau = 0.1$. As a simple form of noise reduction, all HTML markup is removed from both collections prior to signature extraction.

5.2 Competitors

- *SpotSigs* – Spot signatures in combination with the efficient matching algorithm for multi-set Jaccard as described in Section 4. SpotSigs by default performs a clustering step within each partition (on top of the inverted index structures) to identify all near-duplicate pairs.
- *LSH-S* – Locality sensitive hashing (LSH) [20] using spot signatures (S) which in turn serve as input for computing the Min-hash [18] signatures. Since high-dimensional Min-hash permutations are impractical (and in fact intractable) for sparse data [18, 19], we use m random linear functions of the form $\pi_i(x) = (\alpha_i \cdot d + \beta_i) \bmod P$, where α_i and β_i are random integers drawn from the interval $[0, D-1]$, and P is the first prime larger than the dimensionality D of the signature vectors, similarly to the approach investigated by [6]. Candidates within each LSH bucket are post-processed by an additional pairwise clustering step (by calculating exact Jaccard similarities) to filter out false positives and provide pairwise output similar to SpotSigs.
- *I-Match* – The original I-Match algorithm [22] using a single SHA1 hash over a concatenation of tokens filtered by stopwords, special-character tokens, and IDF. Because of the single SHA1 hash used, I-Match does not need to filter out false positives in an additional clustering step but merely returns all pairs of documents that get hashed to the same SHA1 bucket to create comparable output to SpotSigs and LSH.
- *I-Match-S* – An improved I-Match algorithm using sequences of spot signatures (S) instead of simple token sequences.

The runtime of the hashing-based LSH and I-Match approaches is linear in the number of documents n and dimensions m, with $O(k\,l\,n\,m)$ for LSH and $O(n\,m)$ for I-Match, versus $O(n^2\,m)$ in the worst case for SpotSigs. Despite our unfavorable asymptotical behavior, we show that we are empirically able to outperform the competitors in both runtime and recall for high similarity thresholds τ.

5.3 Gold Set of Near Duplicate News Articles

Our first collection consists of 2,160 manually selected, near-duplicate news articles distilled from a large Stanford Web Base [11] crawl in 2006, with articles from `SFGate.com`, `Herald.com`, `Chron.com`, `MCClatchy.com`, etc. The articles have been clustered into 68 directories with an overall size of 102 MB. This "Gold Set" serves as our most valuable reference collection, since these near-duplicates have been manually judged by human assessors which provides an ideal recall base from an IR point-of-view. The huge variations in the layouts used by different news sites makes this an extremely difficult setting for any deduplication algorithm. As shown in Figure 4, the macro-average Cosine similarity of documents (using single tokens with TF·IDF weights) across these directories is only 0.64. This demonstrates the difficulty in processing this manually selected collection of near-duplicate news articles, as even documents within the same directory vary substantially in their overall content.

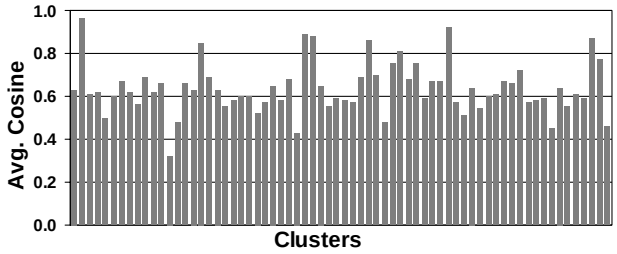

Figure 4: Average Cosine similarity within near-duplicate clusters in the Gold Set.

The processing time for parsing and extracting spot signatures from these 2,160 news articles is I/O bound and constantly takes 17 seconds for all algorithms. Runtime for the actual deduplication step is then almost negligible and takes 1.7 seconds at $\tau = 0.44$ for SpotSigs and up to 9.4 seconds at $\tau = 0.4$ for Shingling, each at their best F1 spots. That is, simple Shingling creates higher-dimensional signature vectors than SpotSigs (denoted as #Spots in Table 1).

5.3.1 SpotSigs vs. Shingling

Figure 5 compares F1 for SpotSigs (using our best signature scheme) against 3-Shingling and single-token signatures, each for Jaccard similarity and Cosine measure (with the latter using TF·IDF weights). Signatures exceeding an IDF range of $[0.2, 0.85]$ were filtered out for all competitors as an effective means of noise- and dimensionality reduction.

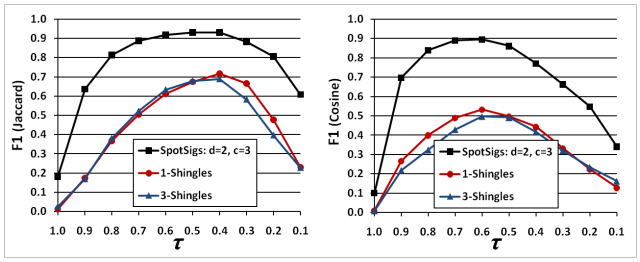

Figure 5: SpotSigs vs. Shingling, each for Jaccard (left) and Cosine (right), as functions of τ.

The plots show that SpotSigs with multi-set Jaccard consistently performs best overall. Using the spot signatures yields an impressive boost from 0.71 to 0.94 in F1 compared to 3-Singling for Jaccard, and an increase from 0.53 to 0.89 for Cosine. For Jaccard this means a 24 percent relative increase in F1, and even 40 percent for Cosine, respectively.

3-Shingling itself does not yield a conclusive improvement in combined recall and precision over single tokens for this hard setting. Note that using IDF statistics already provides a more reliable filtering than the mod-p shingling proposed by [7] that would simply keep only every p^{th} shingle.

5.3.2 Choice of Spot Signatures

We now consider variations in the choice of SpotSigs antecedents, thus aiming to find a good compromise between extracting characteristic signatures while avoiding an overfitting of these signatures to particular articles or sites. Figure 6 shows that we obtain the best F1 result from a combination of articles and flexions of the verbs *be*, *can*, *will*, *have*, *do**, mostly occurring in text contents and less likely to occur in ads or navigational banners. Using a full stopword list (e.g., the official 571 list used in SMART [8]) already tends to yield overly generic signatures but still performs significantly better than IDF-filtered Shingling. Similarly, we find a chain length of 3 and a uniform spot distance of 2 to slightly increase F1 by about 1 absolute percent each (figures omitted). Hence, although the spot signature extraction is highly tunable, it proves to be fairly robust for a variety of different stopword antecedents, spot distances, and chain lengths.

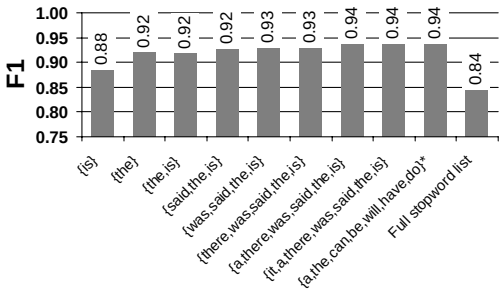

Figure 6: SpotSigs with different antecedents.

5.3.3 SpotSigs vs. Hashing

Table 1 summarizes our results for SpotSigs versus the hashing-based LSH-S and I-Match(-S) variants. Recall that our entire matching approach requires only the configurable parameter τ and an optional IDF range, whereas LSH-S additionally introduces k and l, and I-Match does not even allow for setting a similarity threshold.

For LSH-S, we keep the IDF range fixed to $[0.2, 0.85]$, thus using exactly the same spot signatures as input for both SpotSigs and LSH-S. With k fixed to 6, LSH-S needs about $l = 64$ different hash tables to converge to the same recall as SpotSigs (figures omitted). We choose $l = 32$ to achieve about 90 percent absolute recall on the Gold Set, which conforms to about 98 percent of relative recall compared to the SpotSigs baseline, and we use this setting also for our TREC experiments. Larger values of k, i.e., more specific Min-hash signatures, would in turn require larger values of l, i.e., more hash functions, for good recall.

For I-Match, we vary the bounds of the IDF interval, thus significantly reducing the size of the feature spaces. We achieve the best result for $[0.4, 0.75]$ (essentially confirming results from [12]), yielding 9,473 *distinct* spot signatures at a perfect precision of 1.0, but a recall of only 0.03, and F1 of 0.05. Particularly remarkable is that recall for I-Match increases by an order of magnitude when switching from term-based feature vectors to the spot signatures used for I-Match-S, with only a minor decrease in precision from 1.0 to 0.96. Overall, I-Match and I-Match-S achieve the highest absolute (i.e., user perceived) precision but remain inherently lossy in recall due to the single SHA1 hash function used (coined "brittleness" in [12]). As discussed in [22], I-Match recall may be improved by 40–60 percent by employing several underlying lexica, which is in principle a similar effect as the one LSH achieves by using multiple hash functions.

5.4 TREC WT10g

There is no manually curated gold set of duplicates for larger benchmark collections such as WT10g, consisting of 1.6 Mio documents in 10 GB text data. Thus, based on our Gold Set results, we employ I-Match-S as a highly precise but relative recall base of near duplicates to evaluate SpotSigs versus LSH with a focus on runtime. Furthermore, we limit this setting to documents with at least 5 spot signatures, i.e., documents with significant amounts of natural-language text. This reduces the amount of documents considered from 1,691,565 to 1,171,960, as there are very many short "junk" documents in the collection, and deduplicating those is not in the scope of using spot signatures. We again observe constant parsing and signature extraction times of about 157 seconds for the WT10g collection.

5.4.1 SpotSigs vs. Hashing

Figure 7 shows that SpotSigs starts off by a factor of about 3 better runtimes than LSH-S at $\tau = 1.0$, and it is still 2.6 faster than LSH-S at $\tau = 0.9$ (see Table 2 for details), until the plots cross at about $\tau = 0.78$. This confirms to a deduplication runtime of only 14.2 seconds at $\tau = 1.0$ and 17.1 seconds at $\tau = 0.9$ for SpotSigs. We also see that the general runtime behavior of SpotSigs remains quadratic after all, as the number of partitions and threshold-based pruning effectiveness decreases with decreasing τ.

Table 2 also depicts two more baseline runs for SpotSigs at $\tau = 0.9$. Using only a single partition (NoPart) already accounts with an overall runtime of 196.7 seconds, whereas a run over this single partition without the threshold-based pruning (NoPrune) even takes 2.8 hours. Note that full pairwise vector comparisons without any index structure are clearly intractable for a collection this size, and a respective test run had to be stopped after several days.

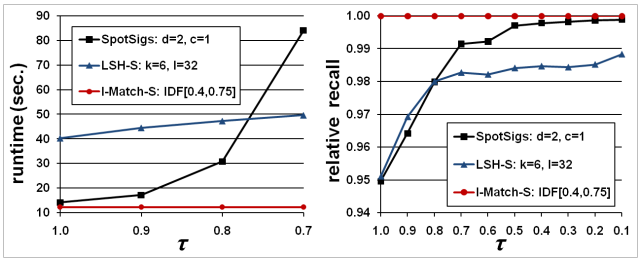

Figure 7: Runtime (left) and relative recall (right) for SpotSigs and LSH-S as functions of τ, using I-Match-S as recall base.

For LSH-S, computing the $k \times l$ Min-hash signatures for each document consumes a major amount of time, even more than sorting the inverted lists in SpotSigs. Thus, both methods generally yield comparable runtimes when LSH-S is tuned to provide sufficient recall, e.g., at $k = 6$ and $l = 32$, and the confidence threshold τ for SpotSigs is reasonably high. Memory consumption (MB) is computed as a lower bound for the data structures used, counting 4 bytes for an integer and 2 bytes for a short integer (used for capturing the frequency values for weighted Jaccard). For example, for LSH-S with 25,033,143 overall spot signatures contained in all vectors, we thus get 25,033,143×(4+2) bytes for the SpotSig vectors, plus 32×1,171,960×4 bytes for the l Min-hashes, and 1,171,960×4 bytes for the document ids. SpotSigs, on the other hand, needs to keep the auxiliary inverted indexes in memory instead of the Min-hash signatures, which account with one additional 4-byte pointer for each spot signature. Tables 1 and 2 show that all setups considered here easily fit into the main memory of current machines.

	IDF	τ	k	l	#Spots	**MB**	Sort/Hash(msec)	Clust.(msec)	**Total(msec)**	Prec.	Rec.	**F1**
SpotSigs	[0.2,0.85]	0.44	n/a	n/a	6,848	**2.5**	814	934	**1,748**	0.96	0.92	**0.94**
1-Shingles	[0.2,0.85]	0.4	n/a	n/a	16,097	**24.8**	1,167	8,284	**9,451**	0.81	0.64	**0.71**
3-Shingles	[0.2,0.85]	0.4	n/a	n/a	65,453	**18.6**	1,620	7,582	**9,202**	0.73	0.65	**0.69**
LSH-S	[0.2,0.85]	0.44	6	32	6,848	**2.0**	345	365	**710**	0.96	0.90	**0.93**
I-Match	[0.4,0.75]	n/a	n/a	n/a	9,473	**0.1**	575	6	**581**	1.00	0.03	**0.05**
I-Match-S	[0.4,0.75]	n/a	n/a	n/a	4,575	**0.1**	274	10	**284**	0.96	0.23	**0.37**

Table 1: Summary of SpotSigs vs. Shingling (using Jaccard), LSH-S, and I-Match(-S) for the Gold Set.

	IDF	τ	k	l	#Spots	**MB**	Sort/Hash(msec)	Clust.(msec)	**Total(msec)**	**Rel.Rec.**
I-Match-S	[0.4,0.75]	n/a	n/a	n/a	86,579	**49**	10,741	1,554	**12,295**	**1.00**
SpotSigs	[0.4,0.75]	1.0	n/a	n/a	86,579	**339**	8,392	5,765	**14,157**	**0.95**
SpotSigs	[0.4,0.75]	0.9	n/a	n/a	86,579	**339**	7,051	10,085	**17,136**	**0.96**
LSH-S	[0.4,0.75]	1.0	6	32	86,579	**180**	20,078	20,148	**40,226**	**0.95**
LSH-S	[0.4,0.75]	0.9	6	32	86,579	**180**	20,213	24,301	**44,514**	**0.97**
NoPart	[0.4,0.75]	0.9	n/a	n/a	86,579	**339**	21,302	175,447	**196,749**	**0.96**
NoPrune	[0.4,0.75]	0.9	n/a	n/a	86,579	**339**	21,425	10,068,588	**10,090,013**	**0.96**

Table 2: Summary of SpotSigs and LSH-S for TREC WT10g, using I-Match-S as recall base.

6. CONCLUSIONS

SpotSigs proved to provide both increased robustness of signatures as well as highly efficient deduplication compared to various state-of-the-art approaches. We demonstrated that for a reasonable range of similarity thresholds, simple vector-length comparisons may already yield a very good partitioning condition to circumvent the otherwise quadratic runtime behavior for this family of clustering algorithms. Moreover, unlike other approaches based on hashing, the SpotSigs deduplication algorithm runs "right out of the box" without the need for further tuning, while remaining exact and efficient. For low similarity thresholds or very skewed distributions of document lengths, however, LSH remains the method-of-choice as it provides the most versatile and tunable toolkit for high-dimensional similarity search.

The SpotSigs matcher can easily be generalized toward more generic similarity search in metric spaces, whenever there is an effective means of bounding the similarity of two documents by a single property such as document or signature length. Future work will focus on efficient access to disk-based index structures, as well as generalizing the bounding approach toward other metrics such as Cosine.

7. REFERENCES

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, p. 459–468, 2006.

[2] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[3] M. Bawa, T. Condie, and P. Ganesan. LSH forest: self-tuning indexes for similarity search. In *WWW*, p. 651–660, 2005.

[4] S. Brin, J. Davis, and H. García-Molina. Copy detection mechanisms for digital documents. In *SIGMOD*, p. 398–409, 1995.

[5] A. Z. Broder. Identifying and filtering near-duplicate documents. In *COM*, p. 1–10, 2000.

[6] A. Z. Broder, M. Charikar, and M. Mitzenmacher. A derandomization using min-wise independent permutations. *J. Discrete Algorithms*, 1(1):11–20, 2003.

[7] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the Web. *Computer Networks*, 29(8-13):1157–1166, 1997.

[8] C. Buckley, G. Salton, and J. Allan. Automatic retrieval with locality information using SMART. In *TREC*, p. 59–72, 1992.

[9] S. Büttcher and C. L. A. Clarke. A document-centric approach to static index pruning in text retrieval systems. In *CIKM*, p. 182–189, 2006.

[10] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, p. 380–388, 2002.

[11] J. Cho, H. Garcia-Molina, T. Haveliwala, W. Lam, A. Paepcke, S. Raghavan, and G. Wesley. Stanford WebBase components and applications. *ACM Trans. Inter. Tech.*, 6(2):153–186, 2006.

[12] A. Chowdhury, O. Frieder, D. Grossman, and M. C. McCabe. Collection statistics for fast duplicate document detection. *ACM Trans. Inf. Syst.*, 20(2):171–191, 2002.

[13] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang. Finding interesting associations without support pruning. *Knowledge and Data Engineering*, 13(1):64–78, 2001.

[14] J. G. Conrad, X. S. Guo, and C. P. Schriber. Online duplicate document detection: signature reliability in a dynamic retrieval environment. In *CIKM*, p. 443–452, 2003.

[15] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, p. 518–529, 1999.

[16] M. Henzinger. Finding near-duplicate Web pages: a large-scale evaluation of algorithms. In *SIGIR*, p. 284–291, 2006.

[17] T. C. Hoad and J. Zobel. Methods for identifying versioned and plagiarized documents. *JASIST*, 54(3):203–215, 2003.

[18] P. Indyk. A small approximately min-wise independent family of hash functions. *J. Algorithms*, 38(1):84–90, 2001.

[19] P. Indyk. Nearest neighbors in high-dimensional spaces. In *Handbook of Discrete and Computational Geometry*. CRC Press, 2004.

[20] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, p. 604–613, 1998.

[21] D. Klein, S. D. Kamvar, and C. D. Manning. From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. In *ICML*, p. 307–314, 2002.

[22] A. Kolcz, A. Chowdhury, and J. Alspector. Improved robustness of signature-based near-replica detection via lexicon randomization. In *KDD*, p. 605–610, 2004.

[23] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe LSH: Efficient indexing for high-dimensional similarity search. In *VLDB*, p. 950–961, 2007.

[24] U. Manber. Finding similar files in a large file system. In *WTEC*, p. 2, 1994.

[25] G. S. Manku, A. Jain, and A. D. Sarma. Detecting near-duplicates for Web crawling. In *WWW*, p. 141–150, 2007.

[26] Web Sociologist's Workbench: `http://dbpubs.stanford.edu/~testbed/doc2/WebBase/SGERHighlight.pdf`

[27] N. Shivakumar and H. García-Molina. SCAM: A copy detection mechanism for digital documents. In *DL*, 1995.

[28] N. Shivakumar and H. Garcia-Molina. Finding near-replicas of documents and servers on the Web. In *WebDB*, p. 204–212, 1998.

[29] K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In *AAAI/IAAI*, p. 1097, 2000.

[30] H. Yang and J. P. Callan. Near-duplicate detection by instance-level constrained clustering. In *SIGIR*, p. 421–428, 2006.