Detecting Near-Duplicates for Web Crawling

Gurmeet Singh Manku
Google Inc.
manku@google.com

Arvind Jain
Google Inc.
arvind@google.com

Anish Das Sarma*
Stanford University
anishds@stanford.edu

ABSTRACT

Near-duplicate web documents are abundant. Two such documents differ from each other in a very small portion that displays advertisements, for example. Such differences are irrelevant for web search. So the quality of a web crawler increases if it can assess whether a newly crawled web page is a near-duplicate of a previously crawled web page or not.

In the course of developing a near-duplicate detection system for a multi-billion page repository, we make two research contributions. First, we demonstrate that Charikar's fingerprinting technique is appropriate for this goal. Second, we present an algorithmic technique for identifying existing f-bit fingerprints that differ from a given fingerprint in at most k bit-positions, for small k. Our technique is useful for both online queries (single fingerprints) and batch queries (multiple fingerprints). Experimental evaluation over real data confirms the practicality of our design.

Categories and Subject Descriptors

E.1 [**Data Structures**]: Distributed data structures; G.2.0 [**Discrete Mathematics**]: General; H.3.3 [**Information Search and Retrieval**]: Search process

General Terms

Algorithms

Keywords

Hamming distance, near-duplicate, similarity, search, sketch, fingerprint, web crawl, web document

1. INTRODUCTION

Web crawling is an integral piece of infrastructure for search engines. *Generic* crawlers [1, 9] crawl documents and links belonging to a variety of topics, whereas *focused* crawlers [27, 43, 46] use some specialized knowledge to limit the crawl to pages pertaining to specific topics. For web crawling, issues like freshness and efficient resource usage have previously been addressed [15, 16, 19]. However, the problem of elimination of near-duplicate web documents in a generic crawl has not received attention.

*Anish worked on this problem at Google in Dec 2005.

Copyright is held by the International World Wide Web Conference Committee (IW3C2). Distribution of these papers is limited to classroom use, and personal use by others.
WWW 2007, May 8–12, 2007, Banff, Alberta, Canada.
ACM 978-1-59593-654-7/07/0005.

Documents that are exact duplicates of each other (due to mirroring and plagiarism) are easy to identify by standard checksumming techniques. A more difficult problem is the identification of *near-duplicate* documents. Two such documents are identical in terms of *content* but differ in a small portion of the document such as advertisements, counters and timestamps. These differences are irrelevant for web search. So if a newly-crawled page $P_{duplicate}$ is deemed a near-duplicate of an already-crawled page P, the crawl engine should ignore $P_{duplicate}$ and all its out-going links (intuition suggests that these are probably near-duplicates of pages reachable from P). Elimination of near-duplicates[1] saves network bandwidth, reduces storage costs and improves the quality of search indexes. It also reduces the load on the remote host that is serving such web pages.

A system for detection of near-duplicate pages faces a number of challenges. First and foremost is the issue of scale: search engines index billions of web-pages; this amounts to a multi-terabyte database. Second, the crawl engine should be able to crawl billions of web-pages per day. So the decision to mark a newly-crawled page as a near-duplicate of an existing page should be made quickly. Finally, the system should use as few machines as possible.

Our contributions:

A. We show that Charikar's simhash [17] is practically useful for identifying near-duplicates in web documents belonging to a multi-billion page repository. simhash is a fingerprinting technique that enjoys the property that fingerprints of near-duplicates differ in a small number of bit positions. We experimentally validate that for a repository of 8B webpages, 64-bit simhash fingerprints and $k = 3$ are reasonable.

B. We develop a technique for solving the Hamming Distance Problem: *In a collection of f-bit fingerprints, quickly find all fingerprints that differ from a given fingerprint in at most k bit positions, where k is a small integer.* Our technique is useful for both online queries (single fingerprints) and batch queries (multiple fingerprints).

C. We present a survey of algorithms and techniques for duplicate detection.

Road-map: In §2, we discuss simhash. In §3, we present a technique for tackling the Hamming Distance Problem. In §4, we present experimental results. In §5, we present a survey of duplicate-detection techniques.

[1] *In practice, presence/absence of near-duplicates may not translate into a binary yes/no decision for eliminating pages from the crawl; instead, it may be used as one of a small number of scoring components that set the priority of a URL for crawling purposes.*

2. FINGERPRINTING WITH SIMHASH

Charikar's simhash [17] is a dimensionality reduction technique. It maps high-dimensional vectors to small-sized fingerprints. It is applied to web-pages as follows: we first convert a web-page into a set of features, each feature tagged with its weight. Features are computed using standard IR techniques like tokenization, case folding, stop-word removal, stemming and phrase detection. A set of weighted features constitutes a high-dimensional vector, with one dimension per unique feature in all documents taken together. With simhash, we can transform such a high-dimensional vector into an f-bit fingerprint where f is small, say 64.

Computation: Given a set of features extracted from a document and their corresponding weights, we use simhash to generate an f-bit fingerprint as follows. We maintain an f-dimensional vector V, each of whose dimensions is initialized to zero. A feature is hashed into an f-bit hash value. These f bits (unique to the feature) increment/decrement the f components of the vector by the weight of that feature as follows: if the i-th bit of the hash value is 1, the i-th component of V is incremented by the weight of that feature; if the i-th bit of the hash value is 0, the i-th component of V is decremented by the weight of that feature. When all features have been processed, some components of V are positive while others are negative. The signs of components determine the corresponding bits of the final fingerprint.

Empirical results: For our system, we used the original C++ implementation of simhash, done by Moses Charikar himself. Concomitant with the development of our system in 2004—2005, Monika Henzinger conducted a study that compared simhash with Broder's shingle-based fingerprints [14]. An excellent comparison of these two approaches appears in Henzinger [35]. A great advantage of using simhash over shingles is that it requires relatively small-sized fingerprints. For example, Broder's shingle-based fingerprints [14] require 24 bytes per fingerprint (it boils down to checking whether two or more Rabin fingerprints out of six are identical). With simhash, for 8B web pages, 64-bit fingerprints suffice; we experimentally demonstrate this in §4.

Properties of simhash: Note that simhash possesses two conflicting properties: (A) The fingerprint of a document is a "hash" of its features, and (B) Similar documents have similar hash values. The latter property is atypical of hash-functions. For illustration, consider two documents that differ in a single byte. Then cryptographic hash functions like SHA-1 or MD5 will hash these two documents (treated as strings) into two completely different hash-values (the Hamming distance between the hash values would be large). However, simhash will hash them into similar hash-values (the Hamming distance would be small).

In designing a near-duplicate detection system based on simhash, one has to deal with the quaintness of simhash described above. The strategy we employed is as follows: we design our algorithms assuming that Property A holds, i.e., the fingerprints are distributed uniformly at random, and we experimentally measure the impact of non-uniformity introduced by Property B on real datasets.

After converting documents into simhash fingerprints, we face the following design problem: Given a 64-bit fingerprint of a recently-crawled web page, how do we quickly discover other fingerprints that differ in at most 3 bit-positions? We address this problem in the next Section.

3. THE HAMMING DISTANCE PROBLEM

Definition: *Given a collection of f-bit fingerprints and a query fingerprint $\mathcal{F}$, identify whether an existing fingerprint differs from $\mathcal{F}$ in at most k bits. (In the batch-mode version of the above problem, we have a set of query fingerprints instead of a single query fingerprint).*

As a concrete instance of the above problem[2], consider a collection of 8B 64-bit fingerprints, occupying 64GB. In the online version of the problem, for a query fingerprint $\mathcal{F}$, we have to ascertain within a few milliseconds whether any of the existing 8B 64-bit fingerprints differs from $\mathcal{F}$ in at most $k = 3$ bit-positions. In the batch version of the problem, we have a *set* of, say, 1M query fingerprints (instead of a solitary query fingerprint $\mathcal{F}$) and we have to solve the same problem for all 1M query fingerprints in roughly 100 seconds. This would amount to a throughput of 1B queries per day.

Let us explore the design space by considering two simple-minded but impractical approaches. One approach is to build a sorted table of all existing fingerprints. Given $\mathcal{F}$, we probe such a table with each $\mathcal{F}'$ whose Hamming distance from $\mathcal{F}$ is at most k. The total number of probes is prohibitively large: for 64-bit fingerprints and $k = 3$, we need $\binom{64}{3} = 41664$ probes. An alternative is to pre-compute all $\mathcal{F}'$ such that some existing fingerprint is at most Hamming distance k away from $\mathcal{F}'$. In this approach, the total number of pre-computed fingerprints is prohibitively large: it could be as many as 41664 times the number of fingerprints.

We now develop a practical algorithm that lies in between the two approaches outlined above: it is possible to solve the problem with a small number of probes and by duplicating the table of fingerprints by a small factor.

Intuition: Consider a sorted table of 2^d f-bit truly random fingerprints. Focus on just the most significant d bits in the table. A listing of these d-bit numbers amounts to "almost a counter" in the sense that (a) quite a few 2^d bit-combinations exist, and (b) very few d-bit combinations are duplicated. On the other hand, the least significant $f - d$ bits are "almost random".

Now choose d' such that $|d' - d|$ is a small integer. Since the table is sorted, a single probe suffices to identify all fingerprints which match $\mathcal{F}$ in d' most significant bit-positions. Since $|d' - d|$ is small, the number of such matches is also expected to be small. For each matching fingerprint, we can easily figure out if it differs from $\mathcal{F}$ in at most k bit-positions or not (these differences would naturally be restricted to the $f - d'$ least-significant bit-positions).

The procedure described above helps us locate an existing fingerprint that differs from $\mathcal{F}$ in k bit-positions, *all of which are restricted to be among the least significant $f - d'$ bits of $\mathcal{F}$*. This takes care of a fair number of cases. To cover all the cases, it suffices to build a small number of additional sorted tables, as formally outlined in the next Section.

3.1 Algorithm for Online Queries

We build t tables: $T_1, T_2, \ldots, T_t$. Associated with table T_i are two quantities: an integer p_i and a permutation π_i over the f bit-positions. Table T_i is constructed by applying permutation π_i to each existing fingerprint; the resulting set of permuted f-bit fingerprints are sorted. Further, each table is compressed (see §3.2) and stored in main-memory

[2]*Please note that the numerical values chosen for the online and the batch versions are for illustrative purposes only.*

of a set of machines. Given fingerprint $\mathcal{F}$ and an integer k, we probe these tables in parallel:

Step 1: Identify all permuted fingerprints in T_i whose top p_i bit-positions match the top p_i bit-positions of $\pi_i(\mathcal{F})$.

Step 2: For each of the permuted fingerprints identified in Step 1, check if it differs from $\pi_i(\mathcal{F})$ in at most k bit-positions.

In Step 1, identification of the first fingerprint in table T_i whose top p_i bit-positions match the top p_i bit-positions of $\pi_i(\mathcal{F})$ can be done in $O(p_i)$ steps by binary search. If we assume that each fingerprint were truly a random bit sequence, *interpolation search* shrinks the run time to $O(\log p_i)$ steps in expectation [52].

3.1.1 Exploration of Design Parameters

Let us see how a reasonable combination of t and p_i can be fixed. We have two design goals: (1) a small set of permutations to avoid blowup in space requirements; and (2) large values for various p_i to avoid checking too many fingerprints in Step 2. Recall that if we seek all (permuted) fingerprints which match the top p_i bits of a given (permuted) fingerprint, we expect 2^{d-p_i} fingerprints as matches. Armed with this insight, we present some examples for $f = 64$ and $k = 3$. We present an analytic solution in §3.1.2.

EXAMPLE 3.1. *Consider $f = 64$ (64-bit fingerprints), and $k = 3$ so near-duplicates' fingerprints differ in at most 3 bit-positions. Assume we have $8B = 2^{34}$ existing fingerprints, i.e. $d = 34$. Here are four different designs, each design has a different set of permutations and p_i values.*

20 tables: *Split 64 bits into 6 blocks having 11, 11, 11, 11, 10 and 10 bits respectively. There are $\binom{6}{3} = 20$ ways of choosing 3 out of these 6 blocks. For each such choice, permutation π corresponds to making the bits lying in the chosen blocks the leading bits (there are several such permutations; we choose one of them uniformly at random). The value of p_i is the total number of bits in the chosen blocks. Thus $p_i = 31, 32$ or 33. On average, a probe retrieves at most $2^{34-31} = 8$ (permuted) fingerprints.*

16 tables: *Split 64 bits into 4 blocks, each having 16 bits. There are $\binom{4}{1} = 4$ ways of choosing 1 out of these 4 blocks. For each such choice, we divide the remaining 48 bits into four blocks having 12 bits each. There are $\binom{4}{1} = 4$ ways of choosing 1 out of these 4 blocks. The permutation for a table corresponds to placing the bits in the chosen blocks in the leading positions. The value of p_i is 28 for all blocks. On average, a probe retrieves $2^{34-28} = 64$ (permuted) fingerprints.*

10 tables: *Split 64 bits into 5 blocks having 13, 13, 13, 13 and 12 bits respectively. There are $\binom{5}{2} = 10$ ways of choosing 2 out of these 5 blocks. For each such choice, permutation π corresponds to making the bits lying in the chosen blocks the leading bits. The value of p_i is the total number of bits in the chosen blocks. Thus $p_i = 25$ or 26. On average, a probe retrieves at most $2^{34-25} = 512$ (permuted) fingerprints.*

4 tables: *Split 64 bits into 4 blocks, each having 16 bits. There are $\binom{4}{1} = 4$ ways of choosing 1 out of these 4 blocks. For each such choice, permutation π corresponds to making the bits lying in the chosen blocks the leading bits. The value of p_i is the total number of bits in the chosen blocks. Thus $p_i = 16$. On average, a probe retrieves at most $2^{34-16} = 256K$ (permuted) fingerprints.*

3.1.2 Optimal Number of Tables

Example 3.1 shows that many different design choices are possible for a fixed choice of f and k. Increasing the number of tables increases p_i and hence reduces the query time. Decreasing the number of tables reduces storage requirements, but reduces p_i and hence increases the query time.

A reasonable approach to fix the trade-off between space and time is to ask the following question: How many tables do we need if we restrict the minimum value of p_i to some constant? For a fixed number of documents 2^d, size of fingerprint f, and maximum allowed hamming distance k, the general solution to the problem is given by the following expression:

$$X(f,k,d) = \begin{cases} 1 & \text{if } d < \tau \\ \min_{r>k} \binom{r}{k} \cdot X(\frac{fk}{r}, k, d - \frac{(r-k)f}{r}) & \text{otherwise} \end{cases}$$

where $X(f,k,d)$ represents the number of tables required, and the threshold τ is determined by the minimum value allowed value of p_i: If the minimum value is p_{min}, $\tau = d - p_{min}$.

Alternately, one could ask what the maximum value of p_i is if we restrict the total number of tables to some number. This problem can be solved similarly.

3.2 Compression of Fingerprints

Compression can shrink the sizes of individual tables. For example, table sizes for 8B documents and 64-bit fingerprints can be shrunk to approximately half their sizes.

The main insight is that successive fingerprints share the top d bits in expectation. We exploit this fact as follows. Let h denote the position of the most-significant 1-bit in the XOR of two successive fingerprints. Thus h takes values between 0 and $f-1$. For a given table, we first compute the distribution of h values and then compute a Huffman code [37] over $[0, f-1]$ for this distribution. Next, we choose a parameter B denoting the *block size*. A typical value for B would be 1024 bytes. A block with B bytes has $8B$ bits. We scan the sorted sequence of (permuted) fingerprints in a table and populate successive blocks as follows:

Step 1: The first fingerprint in the block is remembered in its entirety. This consumes $8f$ bits. Thereafter, Step 2 is repeated for successive fingerprints until a block is full, i.e., we cannot carry out Step 2 without needing $8B + 1$ bits or more.

Step 2: Compute the XOR of the current fingerprint with the previous fingerprint. Find the position of the most-significant 1-bit. Append the Huffman code for this bit-position to the block. Then append the bits to the right of the most-significant 1-bit to the block.

The *key* associated with a block is the *last fingerprint* that was remembered in that block. When a (permuted) fingerprint arrives, an *interpolation search* [52] on the keys helps us figure out which block to decompress. Depending upon the value of p_i and d, and on the distribution of fingerprints (simhash tends to cluster similar documents together), we occasionally have to decompress multiple blocks.

3.3 Algorithm for Batch Queries

As mentioned at the beginning of §3, in the batch version of the Hamming Distance Problem, we have a *batch* of query fingerprints instead of a solitary query fingerprint.

Assume that existing fingerprints are stored in file F and that the batch of query fingerprints are stored in file Q. With 8B 64-bit fingerprints, file F will occupy 64GB. Compression (see §3.2) shrinks the file size to less than 32GB. A batch has of the order of 1M fingerprints, so let us assume that file Q occupies 8MB.

At Google, for example, files F and Q would be stored in a shared-nothing distributed file system called GFS [29]. GFS files are broken into 64MB chunks. Each chunk is replicated at three (almost) randomly chosen machines in a cluster; each chunk is stored as a file in the local file system.

Using the MapReduce framework [24], the overall computation can be split conveniently into two phases. In the first phase, there are as many computational tasks as the number of chunks of F (in MapReduce terminology, such tasks are called *mappers*). Each task solves the Hamming Distance Problem over some 64-MB chunk of F and the entire file Q as inputs. A list of near-duplicate fingerprints discovered by a task is produced as its output. In the second phase, MapReduce collects all the outputs, removes duplicates and produces a single sorted file.

We would like to mention a couple of points about efficiency. First, MapReduce strives to maximize *locality*, i.e., most mappers are co-located with machines that hold the chunks assigned to them; this avoids shipping chunks over the network. Second, file Q is placed in a GFS directory with replication factor far greater than three. Thus copying file Q to various mappers does not become a bottleneck (please see the GFS paper for a discussion of this issue).

How do we solve the Hamming Distance Problem with file Q and a 64-MB chunk of file F? We build tables, as outlined in §3.1, corresponding to file Q (note that for the online mode, the tables were built for file F). Since each individual uncompressed table occupies 8MB, we can easily build 10 such tables in main memory, without worrying about compression. After building the tables, we scan the chunk sequentially, probing the tables for each fingerprint encountered in the scan.

3.4 Previous Work

A generalized version of the Hamming Distance Problem was first proposed by Minsky and Papert [44]: Given a set of n f-bit strings (chosen by an adversary), and a string $\mathcal{F}$, the goal is to identify strings in the set which differ from $\mathcal{F}$ in at most d bit-positions. No efficient solutions are known for general n, f and d. A theoretical study was initiated by Yao and Yao [53], who developed an efficient algorithm for $d = 1$. Their algorithm was improved by Brodal and Gąsienec [10] and Brodal and Venkatesh [11]. For large d, some progress is reported by Greene, Parnas and Yao [31], Dolev *et al* [28] and Arslan and Eğecioğlu [3].

Our problem differs from the one addressed by the theory community in two aspects. First, we assume that the input consists of bit-strings chosen uniformly at random (with some non-uniformity introduced by simhash which hashes similar documents to similar values). Second, we deal with a very large number of bit-strings that do not fit in the main memory of one machine; this limits us to simple external memory algorithms that work well in a distributed setting.

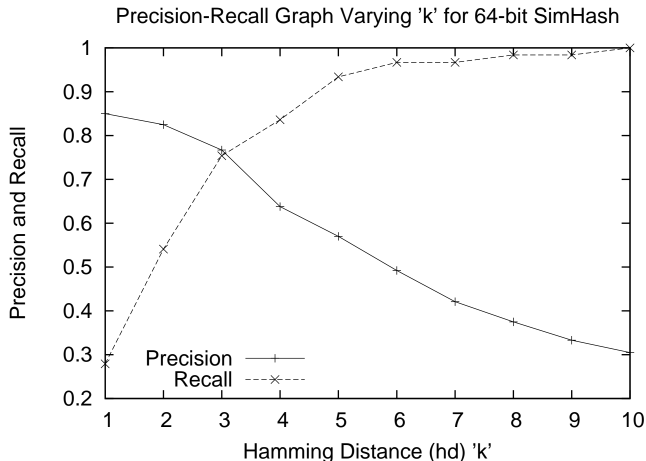


Figure 1: Precision vs recall for various k.

4. EXPERIMENTAL RESULTS

No previous work has studied the trade-off between f and k for the purpose of detection of near-duplicate web-pages using simhash. So our first goal was to ascertain whether simhash is a reasonable fingerprinting technique for near-duplicate detection in the first place. We study simhash in §4.1. Next, we wanted to make sure that the clusters produced by simhash do not impact our algorithms significantly. We analyze distributions of fingerprints in §4.2. Finally, we touch upon running times and scalability issues in §4.3.

4.1 Choice of Parameters

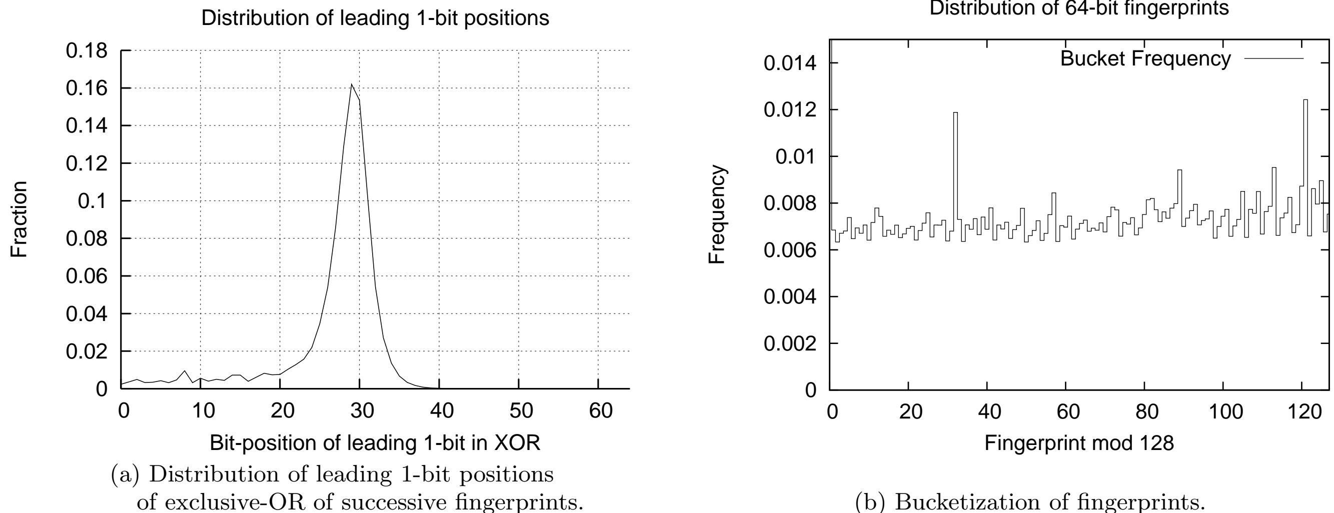
We experimented with $2^{34} = 8B$ simhash fingerprints. We varied k from 1 to 10. For each k, we randomly sampled an equal number of pairs of fingerprints that are at Hamming distance exactly k. We manually tagged each pair as: (1) true positive; (2) false positive; or (3) unknown. We used guidelines from [35] for deciding which of the three categories to put a pair in — radically different pairs are false positive; pages that differ slightly, such as only in counters, ads, or timestamps are true positive; and, pages that cannot be evaluated, e.g., because of content in non-English language, or because a login is needed to access the page, are tagged as unknown.

Figure 1 plots the precision-recall graph for our experiments. *Precision* is defined as the fraction of reported near-duplicates (i.e., having hamming distance at most k) that are true positives. *Recall* denotes the fraction of the total number of near-duplicate pairs (in our sample) that get detected with Hamming distance at most k.

Figure 1 clearly shows the trade-offs for various values of k: A very low value misses near-duplicates (false negatives), and a very high value tags incorrect pairs as near-duplicates (false positives). Choosing $k = 3$ is reasonable because both precision and recall are near 0.75. So, for 64-bit fingerprints, declaring two documents as near-duplicates when their fingerprints differ in at most 3 bits gives fairly high accuracy.

4.2 Distribution of Fingerprints

We designed our algorithm assuming that simhash fingerprints of documents over the web are uniformly random. However, simhash tends to cluster similar documents to-



(a) Distribution of leading 1-bit positions of exclusive-OR of successive fingerprints.

(b) Bucketization of fingerprints.

Figure 2: Analysis of fingerprint distributions.

gether. Figure 2(a) illustrates this phenomenon quantitatively. In Figure 2(a), we plot the distribution of bit-positions of the leading 1-bit in XOR's of successive fingerprints. If the fingerprints were truly random, we would have seen a symmetric distribution which would decay exponentially (the y-value would diminish by half for every increment/decrement of the x-value). Note that the right-half of the distribution indeed exhibits this behavior. However, the left-half of the distribution does not drop off rapidly; there is significant density. This is clearly due to clustering of documents; there are pairs of documents whose simhash values differ by a moderate number of bits because they contain similar content.

In Figure 2(b), we plot the distribution of fingerprints in 128 buckets; bucket boundaries are defined by dividing the space of 2^f fingerprints into 128 equi-sized contiguous intervals. Fingerprints are more or less equally spaced out. Curiously, some spikes exist. These occur due to a variety of reasons. Some examples: (i) several pages are empty; all of these have simhash value of 0, (ii) there are several instances of "File not Found" pages, and (iii) many websites use the same bulletin board software; the login pages for these websites are similar.

4.3 Scalability

For the batch mode algorithm, a compressed version of File Q occupies almost 32GB (as compared with 64GB uncompressed). With 200 mappers, we can scan chunks at a combined rate of over 1GBps. So the overall computation finishes in fewer than 100 seconds. Compression plays an important role in speedup because for a fixed number of mappers, the time taken is roughly proportional to the size of file Q.

5. DUPLICATE DETECTION: A SURVEY

A variety of techniques have been developed to identify pairs of documents that are "similar" to each other. These differ in terms of the end goal, the corpus under consideration, the feature-set identified per document and the signature scheme for compressing the feature-set. In this Section, we present a comprehensive review of near-duplicate detection systems. In the process of summarizing the overall design-space, we highlight how our problem differs from earlier work and why it merits a simhash-based approach.

5.1 The nature of the corpus

Broadly speaking, duplicate-detection systems have been developed for four types of document collections:

a) **Web Documents**: Near-duplicate systems have been developed for finding related-pages [25], for extracting structured data [2], and for identifying web mirrors [6,7].

b) **Files in a file system**: Manber [42] develops algorithms for near-duplicate detection to reduce storage for files. The Venti file system [48] and the Low-bandwidth file system [45] have similar motivations.

c) **E-mails**: Kolcz *et al* [40] identify near-duplicates for spam detection.

d) **Domain-specific corpora**: Various groups have developed near-duplicate detection systems for legal documents (see Conrad and Schriber [22]), TREC benchmarks, Reuters news articles, and Citeseer data.

Our work falls into the first category (Web Documents). We experimented with 8B pages – this is way larger than collection sizes tackled by previous studies: web-clustering by Broder *et al* [14] (30M URLs in 1996), "related pages" by Dean and Henzinger [25] (180M URLs in 1998), web-clustering by Haveliwala *et al* [33] (35M URLs in 2000).

5.2 The end goal: why detect duplicates?

a) **Web mirrors**: For web search, successful identification of web mirrors results in smaller crawling/storage/indexing costs in the absence of near-duplicates, better top-k results for search queries, improvement in page-rank by reducing the in-degree of sites resulting from near-duplicates, cost-saving by not asking human evaluators to rank near-duplicates. See Bharat *et al* [6, 7] for a comparison of techniques for identifying web-mirrors.

b) **Clustering for "related documents" query**: For example, given a news article, a web-surfer might be interested in locating news articles from other sources that report the same event. The notion of "similarity" is at a high level – one could say that the notion of similarity is "semantic" rather than "syntactic", quite different from the notion of duplicates or near-duplicates discussed above. One approach is to use Latent Semantic Indexing [26]. Another approach is to exploit the linkage structure of the web (see Dean and Henzinger [25] who build upon Kleinberg's idea of hubs and authorities [39]). Going further along these lines, Kumar *et al* [41] have proposed discovering "online communities" by identifying dense bipartite sub-graphs of the web-graph.

c) **Data extraction**: Given a moderate-sized collection of similar pages, say reviews at `www.imdb.com`, the goal is to identify the schema/DTD underlying the collection so that we can extract and categorize useful information from these pages. See Joshi *et al* [38] (and references therein) for a technique that clusters web-pages on the basis of structural similarity. See Arasu and Garcia-Molina [2] for another technique that identifies templates underlying pages with similar structure. Also note that metadata (HTML tags) was ignored in a) and b) above.

d) **Plagiarism**: Given a set of reports, articles or assignment-submissions (both source-code and textual reports), the goal is to identify pairs of documents that seem to have borrowed from each other significantly. For some early work in this area, see articles by Baker [4, 5], the COPS system by Brin *et al* [8] and SCAM by Shivakumar and Garcia-Molina [51].

e) **Spam detection**: Given a large number of recently-received e-mails, the goal is to identify SPAM before depositing the e-mail into recipients' mailboxes. The premise is that spammers send similar e-mails *en masse*, with small variation in the body of these e-mails. See Kolcz *et al* [40], who build upon previous work by Chowdhury *et al* [20].

f) **Duplicates in domain-specific corpora**: The goal is to identify near-duplicates arising out of revisions, modifications, copying or merger of documents, etc. See Conrad and Schriber [22] for a case-study involving legal documents at a firm. Manber [42] initiated an investigation into identification of similar files in a file system.

Our near-duplicate detection system improves web crawling, a goal not shared with any of the systems described above.

5.3 Feature-set per document

a) **Shingles from page content**: Consider the sequence of words in a document. A shingle is the hash-value of a k-gram which is a sub-sequence of k successive words. The set of shingles constitutes the set of features of a document. The choice of k is crucial[3]. Hashes of successive k-grams can be efficiently computed by using Rabin's fingerprinting technique [49]. Manber [42] created shingles over characters. The COPS system by Brin *et al* [8] used sentences for creating shingles. Broder *et al* [12, 14] created shingles over words. The total number of shingles per document is clearly large. Therefore, a small-sized signature is computed over the set of shingles, as described in the next sub-section.

b) **Document vector from page content**: In contrast to shingles, a document can be characterized by deploying traditional IR techniques. The idea is to compute its "document-vector" by case-folding, stop-word removal, stemming, computing term-frequencies and finally, weighing each term by its *inverse document frequency* (IDF). Next, given two documents, a "measure" of similarity is defined. Hoad and Zobel [36] argue that the traditional cosine-similarity measure is inadequate for near-duplicate detection. They define and evaluate a variety of similarity measures (but they do not develop any signature-scheme to compress the document-vectors).

   A different approach is taken by Chowdhury *et al* [20] who compute a lexicon (the union of all terms existing in the collection of documents). The lexicon is then pruned (a variety of schemes are studied by the authors). Each document-vector is then modified by removing terms that have been pruned from the lexicon. The resulting document-vectors are fingerprinted. Two documents are said to be near-duplicates iff their fingerprints match. This scheme is rather brittle for near-duplicate detection – a follow-up paper [40] ameliorates the problem by constructing multiple lexicons (these are random subsets of the original lexicon). Now multiple fingerprints per document are computed and two documents are said to be duplicates iff most of their fingerprints match.

   An issue to keep in mind when dealing with document-vectors is that the IDF of any term is global information which changes as the collection changes.

c) **Connectivity information**: For the purpose of finding "related pages", Dean and Henzinger [25] exploited the linkage structure of the web. The premise is that similar pages would have several incoming links in common. Haveliwala *et al* [34] point out that the quality of duplicate detection is poor for pages with very few incoming links. This can be ameliorated by taking anchor text and anchor windows into account.

d) **Anchor text, anchor window**: Similar documents should have similar anchor text. Haveliwala *et al* [34] study the impact of anchor-text and anchor-windows, where an anchor-window is the text surrounding the anchor-text, for example, the paragraph it belongs to. The words in the anchor text/window are folded into the document-vector itself. A weighing function that diminishes the weight of words that are farther away from the anchor text is shown to work well.

e) **Phrases**: Cooper *et al* [23] propose identification of phrases using a phrase-detection system and computing a document-vector that includes phrases as terms. They have tested their ideas on a very small collection (tens of thousands). The idea of using phrases also appears in the work of Hammouda and Kamel [32] who build sophisticated indexing techniques for web-clustering.

We chose to work with the document vector model; simhash converts document vectors into fingerprints. Augmenting the document vector by other signals (anchor text and connectivity information, for example) might improve the quality of our system. We leave these possibilities as future work.

[3]Small k makes dissimilar documents appear similar. Large k makes similar documents appear dissimilar.

5.4 Signature schemes

a) **Mod-p shingles**: A simple compression scheme for shingle-based fingerprints is to retain only those fingerprints whose remainder modulus p is 0, for a sufficiently large value of p. The number of fingerprints retained is variable-sized. Moreover, it is important to ignore commonly-occurring fingerprints since they contribute to false-matches. A drawback of this scheme is that the distance between successive shingles that are retained, is unbounded. This problem has been ameliorated by the "winnowing" technique by Schliemer *et al* [50]. Hoad and Zobel [36] compare a variety of other ideas for pruning the set of shingle-based fingerprints.

b) **Min-hash for Jaccard similarity of sets**: For two sets A and B, let the measure of similarity be $\frac{|A \cap B|}{|A \cup B|}$, also known as the Jaccard measure. Interestingly, it is possible to devise a simple signature scheme such that the probability that the signatures of A and B match is exactly the Jaccard measure [13, 14].

   Several experimental studies have tested the efficacy of min-hash in various settings (Cohen *et al* [21] for association-rule mining, Chen *et al* [18] for selectivity estimation of boolean queries, Gionis *et al* [30] for indexing set-value predicates and Haveliwala [33] for web-clustering).

c) **Signatures/fingerprints over IR-based document vectors**: Charikar's simhash [17] is a fingerprinting technique for compressing document vectors such that two fingerprints are similar iff the document vectors are similar. Another technique for computing signatures over document-vectors is the I-Match algorithm by Chowdhury *et al* [20] that we described earlier. An improved I-Match algorithm appears in [40]. These algorithms have been tested on small document-collections (of the order of tens of thousands) and appear fairly brittle.

d) **Checksums**: Pugh and Henzinger's patent [47] contains the following idea: we divide words in a document into k buckets (by hashing the words, for example), and compute a checksum of each bucket. The set of checksums of two similar documents should agree for most of the buckets.

We chose to work with simhash primarily because it allows us to work with small-sized fingerprints.

Summary

Most algorithms for near-duplicate detection run in batch-mode over the entire collection of documents. For web crawling, an *online* algorithm is necessary because the decision to ignore the hyper-links in a recently-crawled page has to be made quickly. The scale of the problem (billions of documents) limits us to small-sized fingerprints. Luckily, Charikar's simhash technique with 64-bit fingerprints seems to work well in practice for a repository of 8B web pages.

6. FUTURE EXPLORATIONS

Using simhash is a good first step for solving the near-duplicate detection problem. Many other ideas hold promise of improving the quality of near-duplicate detection, and/or making the system more efficient. We list a few:

A. Document size has been shown to play an important role in near-duplicate detection in certain contexts. For example, in Conrad and Schriber [22], two legal documents are deemed to be duplicates iff they have 80% overlap in terminology and ±20% variation in length (these were arrived at by consulting the Library Advisory Board who are trained in the field of Library Science). Perhaps we should devise different techniques for small and large documents. Or perhaps, we should reserve a few bits of the 64-bit fingerprint to hold document length.

B. Is it possible to prune the space of existing fingerprints by asserting that certain documents never have duplicates?

C. Could we categorize web-pages into different categories (for example, by language type), and search for near duplicates only within the relevant categories.

D. Is it feasible to devise algorithms for detecting portions of web-pages that contains ads or timestamps? Perhaps such portions can be automatically removed so that exact checksums over the remaining page suffice for duplicate detection.

E. How sensitive is simhash-based near-duplicate detection to changes in the algorithm for feature-selection and assignment of weights to features?

F. How relevant are simhash-based techniques for focused crawlers [27, 43, 46] which are quite likely to crawl web pages that are similar to each other.

G. Can near-duplicate detection algorithms be developed further to facilitate *clustering* of documents?

7. ACKNOWLEDGEMENTS

We thank Corinna Cortes for a quick review of our paper close to the submission deadline. We thank an anonymous referee for pointing out prior work related to the Hamming Distance Problem; that constitutes §3.4 now. Thanks to the same referee for reminding us that the word "fingerprinting" was introduced by Rabin [49] to denote a scheme where different objects map to different fingerprints with high probability. So synonymizing "fingerprinting" with simhash (which maps similar objects to similar bit-strings) is in conflict with the original intent ⌣̈

8. REFERENCES

[1] A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, and S. Raghavan. Searching the web. *ACM Transactions on Internet Technology*, 1(1):2–43, 2001.

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

[3] A. N. Arslan and Ö. Eğecioğlu. Dictionary look-up within small edit distance. In *Proc. 8th Annual Intl. Computing and Combinatorics Conference (COCOON'02)*, pages 127–136, 2002.

[4] B. S. Baker. A theory of parameterized pattern matching algorithms and applications. In *Proc. 25th Annual Symposium on Theory of Computing (STOC 1993)*, pages 71–80, 1993.

[5] B. S. Baker. On finding duplication and near-duplication in large software systems. In *Proc. 2nd Working Conference on Reverse Engineering*, page 86, 1995.

[6] K. Bharat and A. Broder. Mirror, mirror on the Web: A study of hst pairs with replicated content. In *Proc.*

8th International Conference on World Wide Web (WWW 1999), pages 1579–1590, 1999.

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

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

[9] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[10] G. S. Brodal and L. Gąsieniec. Approximate dictionary queries. In *Proc 7th Combinatorial Pattern Matching Symposium*, pages 65–74, 1996.

[11] G. S. Brodal and S. Venkatesh. Improved bounds for dictionary look-up with one error. *Information Processing Letters*, 75(1-2):57–59, 2000.

[12] A. Broder. On the resemblance and containment of documents. *In Compression and Complexity of Sequences*, 1998.

[13] A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *Proc. 30th Annual Symposium on Theory of Computing (STOC 1998)*, pages 327–336, 1998.

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

[15] A. Z. Broder, M. Najork, and J. L. Wiener. Efficient URL caching for World Wide Web crawling. In *International conference on World Wide Web*, 2003.

[16] S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan-Kauffman, 2002.

[17] M. Charikar. Similarity estimation techniques from rounding algorithms. In *Proc. 34th Annual Symposium on Theory of Computing (STOC 2002)*, pages 380–388, 2002.

[18] Z. Chen, F. Korn, N. Koudas, and S. Muthukrishnan. Selectivity estimation for boolean queries. In *Proc. PODS 2000*, pages 216–225, 2000.

[19] J. Cho, H. García-Molina, and L. Page. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1–7):161–172, 1998.

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

[21] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang. Finding interesting associations without support pruning. In *Proc. 16th Intl. Conf. on Data Engineering (ICDE 2000)*, pages 489–499, 2000.

[22] J. G. Conrad and C. P. Schriber. Constructing a text corpus for inexact duplicate detection. In *SIGIR 2004*, pages 582–583, July 2004.

[23] J. W. Cooper, A. R. Coden, and E. W. Brown. Detecting similar documents using salient terms. In *Proc. 1st International Conf. on Information and Knowledge Management (CIKM 2002)*, pages 245–251, Nov. 2002.

[24] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. 6th Symposium on Operating System Design and Implementation (OSDI 2004)*, pages 137–150, Dec. 2004.

[25] J. Dean and M. Henzinger. Finding related pages in the World Wide Web. *Computer Networks*, 31(11–16):1467–1479, 1999.

[26] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *J American Society for Information Science*, 41(6):391–407, 1990.

[27] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused crawling using context graphs. In *26th International Conference on Very Large Databases, (VLDB 2000)*, pages 527–534, sep 2000.

[28] D. Dolev, Y. Harari, M. Linial, N. Nisan, and M. Parnas. Neighborhood preserving hashing and approximate queries. In *Proc. 5th ACM Symposium on Discrete Algorithms (SODA 1994)*, 1994.

[29] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, pages 29–43, Oct. 2003.

[30] A. Gionis, D. Gunopulos, and N. Koudas. Efficient and tunable similar set retrieval. In *Proc. SIGMOD 2001*, pages 247–258, 2001.

[31] D. Greene, M. Parnas, and F. Yao. Multi-index hashing for information retrieval. In *Proc. 35th Annual Symposium on Foundations of Computer Science (FOCS 1994)*, pages 722–731, 1994.

[32] K. M. Hammouda and M. S. Kamel. Efficient phrase-based document indexing for web document clustering. *IEEE Transactions on Knowledge and Data Engineering*, 16(10):1279–1296, Aug. 2004.

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

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

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

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

[37] D. A. Huffman. A method for the construction of minimum-redundancy codes. In *Proc. Institute of Radio Engineering*, volume 40, pages 1098–1102, Sept. 1952.

[38] S. Joshi, N. Agrawal, R. Krishnapuram, and S. Negi. A bag of paths model for measuring structural similarity in Web documents. In *Proc. 9th ACM Intl. Conf. on Knowledge Discovery and Data Mining (SIGKDD 2003)*, pages 577–582, 2003.

[39] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, Sept. 1999.

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

[41] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the Web for emerging cyber-communities. *Computer Networks: The Intl. J of Computer and Telecommunications Networks*, 31:1481–1493, 1999.

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

[43] F. Menczer, G. Pant, P. Srinivasan, and M. E. Ruiz. Evaluating topic-driven web crawlers. In *Proc. 24th Annual International ACM SIGIR Conference On Research and Development in Information Retrieval*, pages 241–249, 2001.

[44] M. Minsky and S. Papert. *Perceptrons*. MIT Press, 1969.

[45] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. In *Proc. 18th ACM Symposium on Operating System Principles (SOSP 2001)*, pages 174–187, Oct. 2001.

[46] S. Pandey and C. Olston. User-centric web crawling. In *Proc. WWW 2005*, pages 401–411, 2005.

[47] W. Pugh and M. R. Henzinger. Detecting duplicate and near-duplicate files. United States Patent 6,658,423, granted on Dec 2, 2003, 2003.

[48] S. Quinlan and S. Dorward. Venti: A new approach to archival storage. In *First USENIX Conference on File and Storage Technologies*, pages 89–101, 2002.

[49] M. O. Rabin. Fingerprinting by random polynomials. Technical Report Report TR-15-81, Center for Research in Computing Techonlogy, Harvard University, 1981.

[50] S. Schleimer, D. S. Wilkerson, and A. Aiken. Winnowing: Local algorithms for document fingerprinting. In *Proc. SIGMOD 2003*, pages 76–85, June 2003.

[51] N. Shivakumar and H. Garcia-Molina. SCAM: A copy detection mechanism for digital documents. In *Proceedings of the 2nd International Conference on Theory and Practice of Digital Libraries*, 1995.

[52] A. Yao and F. Yao. The complexity of searching in an ordered random table. In *Proc. 17th Annual Symposium on Foundations of Computer Science (FOCS 1976)*, pages 173–177, 1976.

[53] A. C. Yao and F. F. Yao. Dictionary look-up with one error. *J of Algorithms*, 25(1):194–202, 1997.