Finding Near-Duplicate Web Pages: A Large-Scale Evaluation of Algorithms

Monika Henzinger
Google Inc. & Ecole Fédérale de Lausanne (EPFL)
monika@google.com

ABSTRACT

Broder et al.'s [3] shingling algorithm and Charikar's [4] random projection based approach are considered "state-of-the-art" algorithms for finding near-duplicate web pages. Both algorithms were either developed at or used by popular web search engines. We compare the two algorithms on a very large scale, namely on a set of 1.6B distinct web pages. The results show that neither of the algorithms works well for finding near-duplicate pairs *on the same site*, while both achieve high precision for near-duplicate pairs *on different sites*. Since Charikar's algorithm finds more near-duplicate pairs on *different* sites, it achieves a better precision overall, namely 0.50 versus 0.38 for Broder et al. 's algorithm. We present a combined algorithm which achieves precision 0.79 with 79% of the recall of the other algorithms.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing; H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia

General Terms

Algorithms, Measurement, Experimentation

Keywords

Near-duplicate documents, content duplication, web pages

1. INTRODUCTION

Duplicate and near-duplicate web pages are creating large problems for web search engines: They increase the space needed to store the index, either slow down or increase the cost of serving results, and annoy the users. Thus, algorithms for detecting these pages are needed.

A naive solution is to compare *all* pairs to documents. Since this is prohibitively expensive on large datasets, Manber [11] and Heintze [9] proposed first algorithms for detecting near-duplicate documents with a reduced number of comparisons. Both algorithms work on sequences of adjacent characters. Brin et al. [1] started to use word sequences to detect copyright violations. Shivakumar and Garcia-Molina [13, 14] continued this research and focused on scaling it up to multi-gigabyte databases [15]. Broder et al. [3] also used word sequences to efficiently find near-duplicate web pages. Later, Charikar [4] developed an approach based on random projections of the words in a document. Recently Hoad and Zobel [10] developed and compared methods for identifying versioned and plagiarised documents.

Both Broder et al. 's and Charikar's algorithm have elegant theoretical justifications, but neither has been experimentally evaluated and it is not known which algorithm performs better in practice. In this paper we evaluate both algorithms on a very large real-world data set, namely on 1.6B distinct web pages. We chose these two algorithms as both were developed at or used by successful web search engines and are considered "state-of-the-art" in finding near-duplicate web pages. We call them Algorithm B and C.

We set all parameters in Alg. B as suggested in the literature. Then we chose the parameters in Alg. C so that it uses the same amount of space per document and returns about the same number of correct near-duplicate pairs, i.e., has about the same recall. We compared the algorithms according to three criteria: (1) precision on a random subset, (2) the distribution of the number of term differences per near-duplicate pair, and (3) the distribution of the number of near-duplicates per page.

The results are: (1) Alg. C has precision 0.50 and Alg. B 0.38. Both algorithms perform about the same for pairs on the *same* site (low precision) and for pairs on *different* sites (high precision.) However, 92% of the near-duplicate pairs found by Alg. B belong to the *same* site, but only 74% of Alg. C. Thus, Alg. C finds more of the pairs for which precision is high and hence has an overall higher precision. (2) The number of term differences per near-duplicate pair are very similar for the two algorithms, but Alg. B returns fewer pairs with extremely large term differences. (3) The distribution of the number of near-duplicates per page follows a power-law for both algorithms. However, Alg. B has a higher "spread" around the power-law curve. A possible reason for that "noise" is that the bit string representing a page in Alg. B is based on a randomly selected subset of terms in the page. Thus, there might be "lucky" and "unlucky" choices, leading to pages with an artificially high or low number of near-duplicates. Alg. C does not select a subset of terms but is based on *all* terms in the page.

SIGIR'06, August 6–11, 2006, Seattle, Washington, USA.

Finally, we present a combined algorithm that allows for different precision-recall tradeoffs. The precision of one tradeoff is 0.79 with 79% of the recall of Alg. B.

It is notoriously hard to determine which pages belong to the same site. Thus we use the following simplified approach. The *site* of a page is (1) the domain name of the page if the domain name has at most one dot, i.e., at most two levels; and (2) it is the domain name minus the string before the first dot, if the domain name has two or more dots, i.e., three or more levels. For example, the site of `www.cs.berkeley.edu/index.html` is `cs.berkeley.edu`.

The paper is organized as follows: Section 2 describes the algorithms in detail. Section 3 presents the experiments and the evaluation results. We conclude in Section 4.

2. DESCRIPTION OF THE ALGORITHMS

For both algorithms every HTML page is converted into a token sequence as follows: All HTML markup in the page is replaced by white space or, in case of formatting instructions, ignored. Then every maximal alphanumeric sequence is considered a term and is hashed using Rabin's fingerprinting scheme [12, 2] to generate *tokens*, with two exceptions: (1) Every URL contained in the text of the page is broken at slashes and dots, and is treated like a sequence of individual terms. (2) In order to distinguish pages with different images the URL in an IMG-tag is considered to be a term in the page. More specifically, if the URL points to a different host, the whole URL is considered to be a term. If it points to the host of the page itself, only the filename of the URL is used as term. Thus if a page and its images on the same host are mirrored on a different host, the URLs of the IMG-tags generate the same tokens in the original and mirrored version.

Both algorithms generate a bit string from the token sequence of a page and use it to determine the near-duplicates for the page. We compare a variant of Broder et al.'s algorithm as presented by Fetterly et al. [7][1] and a slight modification of the algorithm in [4] as communicated by Charikar [5]. We explain next these algorithms.

Let n be the length of the token sequence of a page. For Alg. B every subsequence of k tokens is fingerprinted using 64-bit Rabin fingerprints, which results in a sequence of $n-k+1$ fingerprints, called *shingles*. Let $S(d)$ be the set of shingles of page d. Alg. B makes the assumption that the percentage of unique shingles on which the two pages d and d' agree, i.e. $\frac{|S(d)\cap S(d')|}{|S(d)\cup S(d')|}$, is a good measure for the similarity of d and d'. To approximate this percentage every shingle is fingerprinted with m different fingerprinting functions f_i for $1 \le i \le m$ that are the same for all pages. This leads to $n-k+1$ values for each f_i. For each i the smallest of these values is called the *i-th minvalue* and is stored at the page. Thus, Alg. B creates an m-dimensional vector of minvalues. Note that multiple occurrences of the same shingle will have the same effect on the minvalues as a single occurrence. Broder et al. showed that the expected percentage of entries in the minvalues vector that two pages d and d' agree on is equal to the percentage of unique shingles on which d and d' agree. Thus, to estimate the similarity of two pages it suffices to determine the percentage of agreeing entries in the minvalues vectors. To save space and speed up the similarity computation the m-dimensional vector of minvalues is reduced to a m'-dimensional vector of *supershingles* by fingerprinting non-overlapping sequences of minvalues: Let m be divisible by m' and let $l = m/m'$. The concatentation of minvalue $j*l, \ldots, (j+1)*l-1$ for $0 \le j < m'$ is fingerprinted with yet another fingerprinting function and is called *supershingle*.[2] This creates a supershingle vector. The number of identical entries in the supershingle vectors of two pages is their *B-similarity*. Two pages are *near-duplicates of Alg. B* or *B-similar* iff their B-similarity is at least 2.

The parameters to be set are m, l, m', and k. Following prior work [7, 8] we chose $m = 84$, $l = 14$, and $m' = 6$. We set $k = 8$ as this lies between $k = 10$ used in [3] and $k = 5$ used in [7, 8]. For each page its supershingle vector is stored, which requires m' 64-bit values or 48 bytes.

Next we describe Alg. C. Let b be a constant. Each token is projected into b-dimensional space by randomly choosing b entries from $\{-1, 1\}$. This projection is the same for all pages. For each page a b-dimensional vector is created by adding the projections of all the tokens in its token sequence. The final vector for the page is created by setting every positive entry in the vector to 1 and every non-positive entry to 0, resulting in a random projection for each page. It has the property that the cosine similarity of two pages is proportional to the number of bits in which the two corresponding projections agree. Thus, the *C-similarity* of two pages is the number of bits their projections agree on. We chose $b = 384$ so that both algorithms store a bit string of 48 bytes per page. Two pages are *near-duplicates of Alg. C* or *C-similar* iff the number of agreeing bits in their projections lies above a fixed threshold t. We set $t = 372$, see Section 3.3.

We briefly compare the two algorithms. In both algorithms the same bit string is assigned to pages with the same token sequence. Alg. C ignores the order of the tokens, i.e., two pages with the same *set* of tokens have the same bit string. Alg. B takes the order into account as the shingles are based on the order of the tokens. Alg. B ignores the frequency of shingles, while Alg. C takes the frequency of terms into account. For both algorithms there can be false positives (non near-duplicate pairs returned as near-duplicates) as well as false negatives (near-duplicate pairs not returned as near-duplicates.) Let T be the sum of the number of tokens in all pages and let D be the number of pages. Alg. B takes time $O(Tm + Dm') = O(Tm)$. Alg. C needs time $O(Tb)$ to determine the bit string for each page. As described in Section 3.3 the C-similar pairs are computed using a trick similar to supershingles. It takes time $O(D)$ so that the total time for Alg. C is $O(Tb)$.

3. EXPERIMENTS

Both algorithms were implemented using the mapreduce framework [6]. Mapreduce is a programming model for simplified data processing on machine clusters. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The algorithms were executed on a set of 1.6B unique pages collected during a crawl of Google's crawler. A preprocessing step grouped pages with the *same* token sequence into *identity sets* and removed for every identity set all but one page.

[1] The only difference is that we omit the wrapping of the shingling "window" from end to beginning described in [7].

[2] *Megashingles* were introduced in [3] to speed up the algorithm even further. Since they do not improve precision or recall, we did not implement them.

About 25-30% of the pages were removed in this step - we do not know the exact number as the preprocessing was done before we received the pages.

Alg. B and C found that between 1.7% and 2.2% of the pages after duplicate removal have a near-duplicate. Thus the total number of near-duplicates and duplicates is roughly the same as the one reported by Broder et al. [3] (41%) and by Fetterly et al. [7] (29.2%) on their collections of web pages. The exact percentage of duplicates and near-duplicates depends on the crawler used to gather the web pages and especially on its handling of session-ids, which frequently lead to exact duplicates. Note that the focus of this paper is *not* on determining the percentage of near-duplicates on the web, but to compare Alg. B and C on the same large real-world data set. The web pages resided on 46.2M hosts with an average of 36.5 pages per host. The distribution of the number of pages per host follows a power-law. We believe that the pages used in our study are fairly representative of the publically available web and thus form a useful large-scale real-world data set for the comparison.

As it is not possible to determine by hand all near-duplicate pairs in a set of 1.6B pages we cannot determine the recall of the algorithms. Instead we chose the threshold t in Alg. C so that both algorithms returned about the same number of correct near-duplicate pairs, i.e., they have about the same recall (without actually knowing what it is). Then we compared the algorithms based on (1) precision, (2) the distribution of the number of term differences in the near-duplicate pairs, and (3) the distribution of the number of near-duplicates per page. Comparison (1) required human evaluation and will be explained next.

3.1 Human Evaluation

We randomly sampled B-similar and C-similar *pairs* and had them evaluated by a human[3], who labeled each near-duplicate pair either as *correct*, *incorrect*, or *undecided*. We used the following definition for a *correct* near-duplicate: *Two web pages are correct near-duplicates if (1) their text differs only by the following: a session id, a timestamp, an execution time, a message id, a visitor count, a server name, and/or all or part of their URL (which is included in the document text), (2) the difference is invisible to the visitors of the pages, (3) the difference is a combination of the items listed in (1) and (2), or (4) the pages are entry pages to the same site.* The most common example of URL-only differences are "parked domains", i.e. domains that are for sale. In this case the URL is a domain name and the HTML page retrieved by the URL is an advertisement page for buying that domain. Pages of domains for sale by the same organization differ usually only by the domain name, i.e., the URL. Examples of Case (4) are entry pages to the same porn site with some different words.

A near-duplicate pair is *incorrect* if *the main item(s) of the page was (were) different.* For example, two shopping pages with common boilerplate text but a different product in the page center is an incorrect near-duplicate pair.

[3]A different approach would be to check the correctness of near-duplicate *pages*, not pairs, i.e., sample *pages* for which the algorithm found at least one near-duplicate and then check whether at least one of its near-duplicates is correct. However, this approach seemed to require too many human comparisons since a page with at least one B-similar page has in the average 135 B-similar pages.

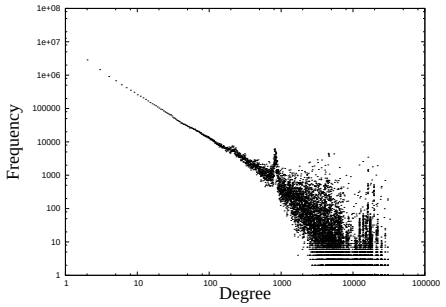

Figure 1: The degree distribution in the B-similarity graph in log-log scale.

B-similarity	Number of near-duplicates	Percentage
2	958,899,366	52.4%
3	383,076,019	20.9%
4	225,454,277	12.3%
5	158,989,276	8.7%
6	104,628,248	5.7%

Table 1: Number of near-duplicate pairs found for each B-similarity value.

The remaining near-duplicate pairs were rated *undecided.* The following three reasons covered 95% of the undecided pairs: (1) prefilled forms with different, but erasable values such that erasing the values results in the same form; (2) a different "minor" item, like a different text box on the side or the bottom; (3) pairs which could not be evaluated. To evaluate a pair the pages as stored at the time of the crawl were visually compared and a Linux `diff` operation was performed on the two token sequences. The `diff` output was used to easily find the differences in the visual comparison. However, for some pages the `diff` output did not agree with visual inspection. This happened, e.g., because one of these pages automatically refreshed and the fresh page was different from the crawled page. In this case the pair was labeled as "cannot evaluate". A pair was also labeled as "cannot evaluate" when the evaluator could not discern whether the difference in the two pages was major or minor. This happened mostly for Chinese, Japanese, or Korean pages.

3.2 The Results for Algorithm B

Alg. B generated 6 supershingles per page, for a total of 10.1B supershingles. They were sorted and for each pair of pages with an identical supershingle we determined its B-similarity. This resulted in 1.8B B-similar pairs, i.e., pairs with B-similarity at least 2.

Let us define the following *B-similarity graph*: Every page is a node in the graph. There is an edge between two nodes iff the pair is B-similar. The label of an edge is the B-similarity of the pair, i.e., 2, 3, 4, 5, or 6. The graph has 1.8B edges, about half of them have label 2 (see Table 1.) A node is considered a *near-duplicate page* iff it is incident to at least one edge. Alg. B found 27.4M near-duplicate pages. The average degree of the B-similarity graph is abount 135. Figure 1 shows the degree distribution in log-log scale. It follows a power-law with exponent about -1.3.

We randomly sampled 96556 B-similar pairs. In 91.9% of the cases both pages belonged to the same site. We then

Near dups	Number of pairs	Correct	Not correct	Undecided
all	1910	0.38	0.53	0.09 (0.04)
same site	1758	0.34	0.57	0.09 (0.04)
diff. sites	152	0.86	0.06	0.08 (0.01)
B-sim 2	1032	0.24	0.68	0.08 (0.03)
B-sim 3	389	0.42	0.48	0.1 (0.04)
B-sim 4	240	0.55	0.36	0.09 (0.05)
B-sim 5	143	0.71	0.23	0.06 (0.02)
B-sim 6	106	0.85	0.05	0.1 (0.08)

Table 2: Alg. B: Fraction of correct, not correct, and undecided pairs, with the fraction of prefilled, erasable forms out of *all* pairs in that row in parenthesis.

	All pairs	Same site
URL only	302 (41%)	194 (32%)
Time stamp only	123 (17%)	119 (20%)
Combination	161 (22%)	145 (24%)
Execution time only	118 (16%)	114 (19%)
Visitor count only	22 (3%)	20 (3%)
Rest	5 (0%)	5 (1%)

Table 3: The distribution of differences for correct B-similar pairs.

subsampled these pairs and checked each of the resulting 1910 pairs for correctness (see Table 2.) The overall precision is 0.38. However, the errors arise mostly for pairs on the same site: There the precision drops to 0.34, while for pairs on different sites the precision is 0.86. The reason is that very often pages on the same site use the same boilerplate text and differ only in the main item in the center of the page. If there is a large amount of boilerplate text, chances are good that the algorithm cannot distinguish the main item from the boilerplate text and classifies the pair as near-duplicate.

Precision improves for pairs with larger B-similarity. This is expected as larger B-similarity means more agreement in supershingles. While the correctness of B-similarity 2 is only 0.24, this value increases to 0.42 for B-similarity 3, and to 0.66 for B-similarity larger than 3. However, less than half of the pairs have B-similarity 3 or more.

Table 3 analyzes the correct B-similar pairs. It shows that URL-only differences account for 41% of the correct pairs. For pairs on *different* sites 108 out of the 152 B-similar pairs differ only in the URL. This explains largely the high precision in this case. Time stamps-only differences, execution time-only differences, and combinations of differences are about equally frequent. The remaining cases account for less than 4% of the correct pairs.

Only 9% of all pairs are labeled undecided. Table 4 shows that 92% of them are on the same site. Almost half the cases are pairs that could not be evaluated. Prefilled, erasable forms are the reason for 41% of the cases. Differences in minor items account for only 11%.

Next we analyze the distribution of term differences for the 1910 B-similar pairs. To determine the term difference of a pair we executed the Linux `diff` command over the two token sequences and used the number of tokens that were returned. The average term difference is 24, the mean is

	All pairs	Same site
Cannot evaluate	78 (47%)	66 (43%)
Form	69 (41%)	67 (43%)
Different minor item	19 (11%)	19 (12%)
Other	1 (1%)	1 (1%)

Table 4: Reasons for undecided B-similar pairs.

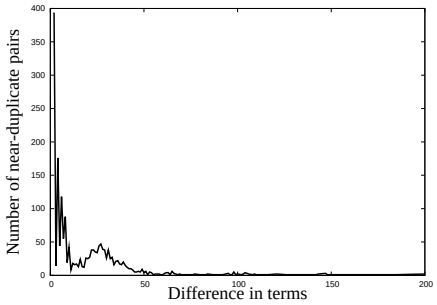

Figure 2: The distribution of term differences in the sample of Alg. B.

11, 21% of the pairs have term difference 2, 90% have term difference less than 42. For 17 pairs the term difference is larger than 200. None of them are correct near-duplicates. They mostly consist of repeated text in one page (like repeated lists of countries) that is completely missing in the other page and could probably be avoided if the frequency of shingles was taken into account. Figure 2 shows the distribution of term difference up to 200. The spike around 19 to 36 consists of 569 pages and is mostly due to two data bases on the web. It contains 326 pairs from the NIH Nucleotide database and 103 pairs from the Historic Herefordshire database. These two databases are a main source of errors for Alg. B. All of the evaluated pairs between pages of one of these database were rated as incorrect. However, 19% of the pairs in the random sample of 96556 pairs came from the NIH database and 9% came from the Herefordshire database[4]. The pages in the NIH database consist of 200-400 tokens and differ in 10-30 consecutive tokens, the pages in the Herefordshire database consist of 1000-2000 tokens and differ in about 20 consecutive tokens. In both cases Alg. B has a good chance of picking two out of the six supershingles from the long common token sequences originating from boilerplate text. However, the number of different tokens is large enough so that Alg. C returned only three of the pairs in the sample as near-duplicate pairs.

3.3 The Results for Algorithm C

We partitioned the bit string of each page into 12 non-overlapping 4-byte pieces, creating 20B pieces, and computed the C-similarity of all pages that had at least one piece in common. This approach is guaranteed to find *all* pairs of pages with difference up to 11, i.e., C-similarity 373, but might miss some for larger differences.

Alg. C returns all pairs with C-similarity at least t as near-duplicate pairs. As discussed above we chose t so that both algorithms find about the same number of correct near-

[4]A second independent sample of 170318 near-duplicate pairs confirmed these percentages.

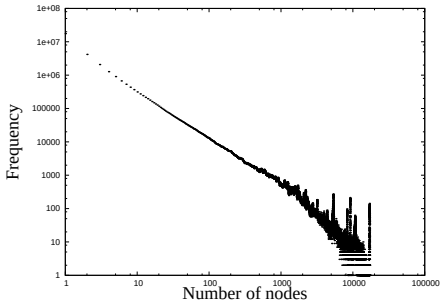

Figure 3: The degree distribution in the C-similarity graph in log-log scale.

duplicate pairs. Alg. B found 1,831M near-duplicate pairs containing about $1,831M{*}0.38 \approx 696M$ correct near-duplicate pairs. For $t = 372$ Alg. C found 1,630M near-duplicate pairs containing about $1,630M * 0.5 = 815M$ correct near-duplicate pairs. Thus, we set $t = 372$. In a slight abuse of notation we call a pair *C-similar* iff it was returned by our implementation. The difference to before is that there might be some pairs with C-similarity 372 that are not C-similar because they were not returned by our implementation.

We define the *C-similarity graph* analog to the B-similarity graph. There are 35.5M nodes with at least one incident edge, i.e., near-duplicate pages. This is almost 30% more than for Alg. B. The average degree in the C-similarity graph is almost 92. Figure 3 shows the degree distribution in log-log scale. It follows a power-law with exponent about -1.4.

We randomly sampled 172,464 near-duplicate pairs. Out of them 74% belonged to the same site. In a random subsample of 1872 near-duplicate pairs Alg. C achieves an overall precision of 0.50 with 27% incorrect pairs and 23% undecided pairs (see Table 5.) For pairs on *different* sites the precision is 0.9 with only 5% incorrect pairs and 5% undecided pairs. For pairs on the *same* site the precision is only 0.36 with 34% incorrect pairs and 30% undecided pairs. The number in parenthesis gives the percentage of pairs out of all pairs that were marked *Undecided* because of prefilled, but erasable forms. It shows that these pages are the main reason for the large number of undecided pairs of Alg. C.

Table 5 also lists the precision for different C-similarity ranges. Surprisingly precision is highest for C-similarity between 372 and 375. This is due to the way we break URLs at slashes and dots. Two pages that differ only in the URL usually differ in 2 to 4 tokens since these URLs are frequently domain names. This often places the pair in the range between 372 and 375. Indeed 57% of the pairs that differ only in the URL fall into this range. This explains 387 out of the 470 correct near-duplicates for this C-similarity range.

Table 6 analyzes the correct near-duplicate pairs. URL-only differences account for 72%, combinations of differences for 11%, time stamps and execution time for 9% together with about half each. The remaining reasons account for 8%. For near-duplicate pairs on the *same* site only 53% of the correct near-duplicate pairs are caused by URL-only differences, while 19% are due to a combination of reasons and 8% to time stamps and execution time. For pairs on *different* sites 406 of the 479 C-similar pairs differ only in the URL. This explains the high precision in that case.

Table 7 shows that 95% of the undecided pairs are on the same site and 80% of the undecided pairs are due to forms. Only 15% are could not be evaluated. The total number of such cases (64) is about the same as for Alg. B (78).

Near dups	Number of pairs	Correct	Not correct	Undecided
all	1872	0.50	0.27	0.23 (0.18)
same site	1393	0.36	0.34	0.30 (0.25)
different site	479	0.90	0.05	0.05 (0)
C-sim ≥ 382	179	0.47	0.37	0.16 (0.10)
$382 >$ C-sim ≥ 379	407	0.40	0.37	0.23 (0.18)
$379 >$ C-sim ≥ 376	532	0.37	0.27	0.35 (0.30)
C-sim < 376	754	0.62	0.19	0.19 (0.12)

Table 5: Alg. C: Fraction of correct, not correct, and undecided pairs, with the fraction of prefilled, erasable forms out of *all* pairs in that row in parenthesis.

	All pairs	Same site
URL only	676 (72%)	270 (53%)
Combination	100 (11%)	95 (19%)
Time stamp only	42 (4%)	40 (8%)
Execution time only	41 (4%)	39 (8%)
Message id only	21 (2%)	18 (4%)
Rest	57 (6%)	43 (8%)

Table 6: The distribution of differences for correct C-similar pairs.

We also analyzed the term difference in the 1872 near-duplicate pairs sampled from Alg. C. Figure 4 shows the number of pairs for a given term difference up to 200. The average term differences is 94, but this is only due to outliers, the mean is 7, 24% had a term difference of 2, and 90% had a term difference smaller than 44. There were 90 pairs with term differences larger than 200.

The site which causes the largest number of incorrect C-similar pairs is `http://www.businessline.co.uk/`, a UK business directory that lists in the center of each page the phone number for a type of business for a specific area code. Two such pages differ in about 1-5 not necessarily consecutive tokens and agree in about 1000 tokens. In a sample of 172,464 C-similar pairs, 9.2% were pairs that came from this site[5], all such pairs that were evaluated in the subsample of 1872 were incorrect. Since the token differences are non-consecutive, shingling helps: If x different tokens are separated by y common tokens s.t. any consecutive sequence of common tokens has length less than k, then $x + y + k - 1$ different shingles are generated. Indeed, none of these pairs in the sample was returned by Alg. B.

3.4 Comparison of Algorithms B and C

3.4.1 Graph Structure

The average degree in the C-similarity graph is smaller than in the B-similarity graph (92 vs. 135) and there are fewer high-degree nodes in the C-similarity graph: Only

[5]In a second independent random sample of 23475 near-duplicate pairs we found 9.4% such pairs.

	All pairs	Same site
Form	345 (80%)	343 (84%)
Cannot evaluate	64 (15%)	54 (13%)
Other	21 (5%)	12 (3%)
Different minor item	2 (0%)	1 (0%)

Table 7: Reasons for undecided C-similar pairs.

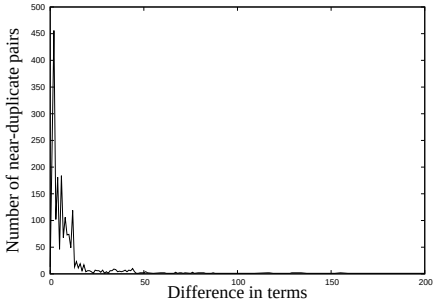

Figure 4: The distribution of term differences in the sample of Alg. C.

7.5% of the nodes in the C-similarity graph have degree at least 100 vs. 10.8% in the B-similarity graph. The plots in Figures 1 and 3 follow a power-law distribution, but the one for Alg. B "spreads" around the power-law curve much more than the one for Alg. C. This can be explained as follows: Each supershingle depends on 112 terms (14 shingles with 8 terms each.) Two pages are B-similar iff two of its supershingles agree. If the page is "lucky", two of its supershingles consist of common sequences of terms (like lists of countries in alphabetical order) and the corresponding node has a high degree. If the page is "unlucky", all supershingles consist of uncommon sequences of terms, leading to low degree. In a large enough set of pages there will always be a certain percentage of pages that are "lucky" and "unlucky", leading to a deviation from the power-law. Alg. C does not select subsets of terms, its bit string depends on all terms in the sequence, and thus the power-law is stricter followed.

3.4.2 Manual Evaluation

Alg. C outperforms Alg. B with a precision of 0.50 versus 0.38 for Alg. B. A more detailed analysis shows a few interesting differences.

Pairs on different sites: Both algorithms achieve high precision (0.86 resp. 0.9), 8% of the B-similar pairs and 26% of the C-similar pairs are on different sites. Thus, Alg. C is superior to Alg. B (higher precision *and* recall) for pairs on different sites. The main reason for the higher recall is that Alg. C found 406 pairs with URL-only differences on different sites, while Alg. B returned only 108 such pairs.

Pairs on the same site: Neither algorithm achieves high precision for pages on the same site. Alg. B returned 1752 pairs with 597 correct pairs (precision 0.34), while Alg. C' returns 1386 pairs with 500 correct pairs (precision 0.37). Thus, Alg. B has 20% higher recall, while Alg. C achieves slightly higher precision for pairs on the same site. However, a combination of the two algorithms as described in the next section can achieve a much higher precision for pairs on the same site without sacrificing much recall.

Evaluation	Alg B	Alg C
Incorrect	11	20
Undecided	8	53
Correct	0	17

Table 8: The evaluations for the near-duplicate pairs with term difference larger than 200.

B-similarity	C-similarity average
2	331.5
3	342.4
4	352.4
5	358.9
6	370.9

Table 9: For a given B-similarity the average C-similarity.

All pairs: Alg. C found more near-duplicates with URL-only differences (676 vs. 302), while Alg. B found more correct near-duplicates whose differences lie only in the time stamp or execution time (241 vs. 83). Alg. C found many more undecided pairs due to prefilled, erasable forms than Alg. B (345 vs. 69). In many applications these forms would be considered correct near-duplicate pairs. If so the overall precision of Alg. C would increase to 0.68, while the overall precision of Alg. B would be 0.42.

3.4.3 Term Differences

The results for term differences are quite similar, except for the larger number (19 vs. 90) of pairs with term differences larger than 200. To explain this we analyzed each of these 109 pairs (see Table 8). However, there are mistakes in our counting methods due to a large number of image URLs that were used for layout improvements and are counted by `diff`, but are invisible for the user. They affect 9 incorrect pairs and 25 undecided pairs of Alg. C, but no pair of Alg. B. Subtracting them leaves both algorithms with 11 incorrect pairs with large term differences, and Alg. B with 8 and Alg. C with 28 undecided pairs with large term differences. Seventeen pairs of Alg. C with large term differences were rated as correct - they are entry pages to the same porn site.

Altogether we conclude that the performance of the two algorithms with respect to term differences is quite similar, but Alg. B returns fewer pairs with very large term differences. However, with 20 incorrect and 17 correct pairs with large term difference the precision of Alg. C is barely influenced by its performance on large term differences.

3.4.4 Correlation

To study the correlation of B-similarity and C-similarity we determined the C-similarity of each pair in a random sample of 96,556 B-similar pairs. About 4% had C-similarity at least 372. The average C-similarity was almost 341. Table 9 gives for different B-similarity values the average C-similarity. As can be seen the larger the B-similarity the larger the average C-similarity. To show the relationship more clearly Figure 5 plots for each B-similarity level the distribution of C-similarity values. For B-similarity 2, most of the pairs have C-similarity below 350 with a peak at 323. For higher B-similarity values the peaks are above 350.

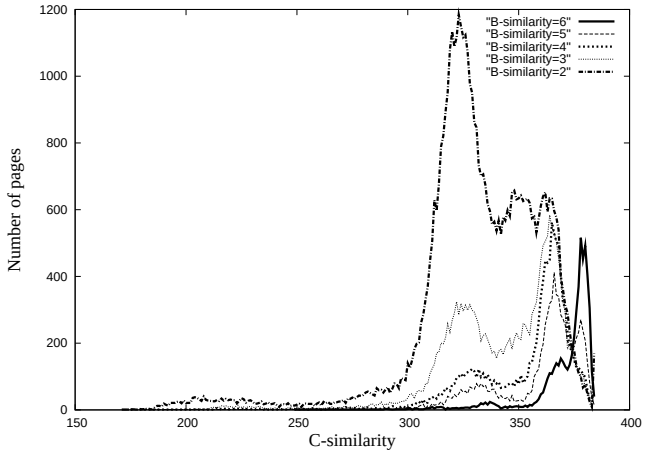

Figure 5: The C-similarity distribution for various fixed B-similarities.

C-similarity	B-similarity average
372-375	0.10
376-378	0.22
379-381	0.32
382-384	0.32

Table 10: For a given C-similarity range the average B-similarity.

We also determined the B-similarity for a random sample of 169,757 C-similar pairs. Again about 4% of the pairs were B-similar, but for 95% of the pairs the B-similarity was 0. Table 10 gives the details for various C-similarity ranges.

3.5 The Combined Algorithm

The algorithms wrongly identify pairs as near-duplicates either (1) because a small difference in tokens causes a large semantic difference or (2) because of unlucky random choices. As the bad cases for Alg B showed pairs with a large amount of boilerplate text and a not very small number (like 10 or 20) of different tokens that are all consecutive are at risk of being wrongly identified as near-duplicates by Alg. B, but are at much less risk by Alg. C. Thus we studied the following combined algorithm: First compute all B-similar pairs. Then filter out those pairs whose C-similarity falls below a certain threshold. To choose a threshold we plotted the precision of the combined algorithm for different threshold values in Figure 6. It shows that precision can significantly improve if a fairly high value of C-similarity, like 350, is used.

To study the impact of different threshold values on recall let us define R to be the number of correct near-duplicate pairs returned by the combined algorithm divided by the number of correct near-duplicate pairs returned by Alg. B. We chose Alg. B because the combined algorithm tries to filter out the false positives of Alg. B. We randomly subsampled 948 pairs out of the 1910 pairs that were scored for Alg. B, creating the sample S_1. The remaining pairs from the 1910 pairs form sample S_2. We used S_1 to choose a cutoff threshold and S_2 as testing set to determine the resulting precision and R-value. Figure 7 plots for S_1 precision versus R for all C-similar thresholds between 0 and 384. As expected precision decreases with increasing R. The long flat range corresponds to different thresholds between 351 and 361 for which precision stays roughly the same while recall increases significantly. For the range between 351 and 359 Table 11 gives the resulting precision and R-values. It shows that a C-similarity threshold of 353, 354, or 355 would be a good choice, achieving a precision of 0.77 while keeping R over 0.8. We picked 355.

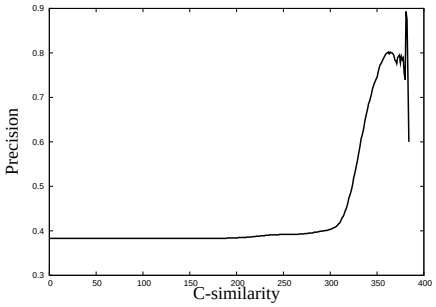

Figure 6: For each C-similarity threshold the corresponding precision of the combined algorithm.

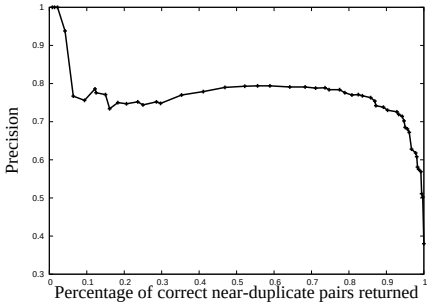

Figure 7: For the training set S_1 the R-value versus precision for different cutoff thresholds t.

The resulting algorithm returns on the testing set S_2 363 out of the 962 pairs as near-duplicates with a precision of 0.79 and an R-value of 0.79. For comparison consider using Alg. B with a B-similarity cutoff of 3. That algorithm

Percentage of correct pairs returned	Precision	C-similarity threshold
0.869	0.754	351
0.858	0.763	352
0.836	0.768	353
0.825	0.771	354
0.808	0.770	355
0.789	0.776	356
0.775	0.784	357
0.747	0.784	358
0.736	0.789	359

Table 11: On the training set S_1 for different C-similarity thresholds the corresponding precision and percentage of returned correct pairs.

Near dups	Number of pairs	Correct	Incorrect	Undecided	R
all	363	0.79	0.15	0.06	0.79
same site	296	0.74	0.19	0.07	0.73
different site	65	0.99	0.00	0.01	0.97

Table 12: The combined Algorithm on the testing set S_2: Fraction of correct, incorrect, and undecided pairs and R-value.

only returns 244 correct near-duplicates with a precision of 0.56, while the combined algorithm returns 287 correct near-duplicates with a precision of 0.79. Thus, the combined algorithm is superior in both precision and recall to using a stricter cutoff for Alg. B. Note that the combined algorithm can be implemented with the same amount of space since the bit strings for Alg. C can be computed "on the fly" during filtering.

Table 12 also shows that 82% of the returned pairs are on the same site and that the precision improvement is mostly achieved for these pairs. With 0.74 this number is much better than either of the individual algorithms.

A further improvement could be achieved by running both Alg. C and the combined algorithm and returning the pairs on different sites from Alg. C and the pairs for the same site from the combined algorithm. This would generate 1.6B$*$0.26 $\approx$ 416M pairs one the same site with 374M correct pairs and 1.6B$*$0.919 $*$ 0.79 $\approx$ 1163M pairs on different sites with about 919M correct pairs. Thus approximately 1335M correct pairs would be returned with a precision of 0.85, i.e., both recall and precision would be superior to the combined algorithm alone.

4. CONCLUSIONS

We performed an evaluation of two near-duplicate algorithms on 1.6B web pages. Neither performed well on pages from the same site, but a combined algorithm did without sacrificing much recall.

Two changes might improve the performance of Alg. B and deserve further study: (1) A weighting of shingles by frequency and (2) using a different number k of tokens in a shingle. For example, following [7, 8] one could try $k = 5$. However, recall that 28% of the incorrect pairs are caused by pairs of pages in two databases on the web. In these pairs the difference is formed by one consecutive sequence of tokens. Thus, reducing k would actually increase the chances that pairs of pages in these databases are incorrectly identified as near-duplicates.

Note that Alg. C also could work with much less space. It would be interesting to study how this affects its performance. Additionally it would be interesting to explore whether applying Alg. C to sequences of tokens, i.e., shingles, instead of individual tokens would increase its performance.

As our results show both algorithms perform poorly on pairs from the same site, mostly due to boilerplate text. Using a boilerplate detection algorithm would probably help. Another approach would be to use a different, potentially slower algorithm for pairs on the same site and apply (one of) the presented algorithms to pairs on different sites.

5. ACKNOWLEDGMENTS

I want to thank Mike Burrows, Lars Engebretsen, Abhay Puri and Oren Zamir for their help and useful discussions.

6. REFERENCES

[1] S. Brin, J. Davis, and H. Garcia-Molina. Copy Detection Mechanisms for Digital Documents. In *1995 ACM SIGMOD International Conference on Management of Data* (May 1995), 398–409.

[2] A. Broder. Some applications of Rabin's fingerprinting method. In Renato Capocelli, Alfredo De Santis, and Ugo Vaccaro, editors, *Sequences II: Methods in Communications, Security, and Computer Science,* 1993:143–152.

[3] A. Broder, S. Glassman, M. Manasse, and G. Zweig. Syntactic Clustering of the Web. In *6th International World Wide Web Conference* (Apr. 1997), 393–404.

[4] M. S. Charikar. Similarity Estimation Techniques from Rounding Algorithms. In *34th Annual ACM Symposium on Theory of Computing* (May 2002).

[5] M. S. Charikar. Private communication.

[6] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *6th Symposium on Operating System Design and Implementation* (Dec. 2004), 137–150.

[7] D. Fetterly, M. Manasse, and M. Najork. On the Evolution of Clusters of Near-Duplicate Web Pages. In *1st Latin American Web Congress* (Nov. 2003), 37–45.

[8] D. Fetterly, M. Manasse, and M. Najork. Detecting Phrase-Level Duplication on the World Wide Web. To appear in *28th Annual International ACM SIGIR Conference* (Aug. 2005).

[9] N. Heintze. Scalable Document Fingerprinting. In *Proc. of the 2nd USENIX Workshop on Electronic Commerce* (Nov 1996).

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

[11] U. Manber. Finding similar files in a large file system. In *Proc. of the USENIX Winter 1994 Technical Conference* (Jan. 1994).

[12] M. Rabin. Fingerprinting by random polynomials. Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[13] N. Shivakumar and H. Garcia-Molina. SCAM: a copy detection mechanism for digital documents. In *Proc. International Conference on Theory and Practice of Digital Libraries* (June 1995).

[14] N. Shivakumar and H. Garcia-Molina. Building a scalable and accurate copy detection mechanism. In *Proc. ACM Conference on Digital Libraries* (March 1996), 160–168.

[15] N. Shivakumar and H. Garcia-Molina. Finding near-replicas of documents on the web. In *Proc. Workshop on Web Databases* (March 1998), 204–212.