Using Bloom Filters to Refine Web Search Results*

Navendu Jain†
Department of Computer Sciences
University of Texas at Austin
Austin, TX, 78712
nav@cs.utexas.edu

Mike Dahlin
Department of Computer Sciences
University of Texas at Austin
Austin, TX, 78712
dahlin@cs.utexas.edu

Renu Tewari
IBM Almaden Research Center
650 Harry Road
San Jose, CA, 95111
tewarir@us.ibm.com

ABSTRACT

Search engines have primarily focused on presenting the most relevant pages to the user quickly. A less well explored aspect of improving the search experience is to remove or group all near-duplicate documents in the results presented to the user. In this paper, we apply a Bloom filter based similarity detection technique to address this issue by refining the search results presented to the user. First, we present and analyze our technique for finding similar documents using content-defined chunking and Bloom filters, and demonstrate its effectiveness in compactly representing and quickly matching pages for similarity testing. Later, we demonstrate how a number of results of popular and random search queries retrieved from different search engines, Google, Yahoo, MSN, are similar and can be eliminated or re-organized.

1. INTRODUCTION

Enterprise and web search has become a ubiquitous part of the web experience. Numerous studies have shown that the ad-hoc distribution of information on the web has resulted in a high degree of content aliasing (i.e., the same data contained in pages from different URLs) [14] and which adversely affects the performance of search engines [6]. The initial study by Broder et al., in 1997 [7], and the later one by Fetterly et al. [11], shows that around 29.2% of data is common across pages in a sample of 150 million pages. This common data when presented to the user on a search query degrades user-experience by repeating the same information on every click.

Similar data can be grouped or eliminated to improve the search experience. Similarity based grouping is also useful for organizing the results presented by meta-crawlers (e.g., vivisimo, metacrawler, dogpile, copernic). The findings by `searchenginejournal.com` [2] show a significant overlap of search results returned by Google and Yahoo search engines—the top 20 keyword searches from Google had about 40% identical or similar pages to the Yahoo results. Sometimes search results may appear different purely due to the restructuring and reformatting of data. For example, one site may format a document into multiple web pages, with the top level page only containing a fraction of the document along with a "next" link to follow to the remaining part, while another site may have the entire document in the same web page. An effective similarity detection technique should find these "contained" documents and label them as similar.

Although improving search results by identifying near-duplicates had been proposed for Altavista [6], we found that popular search engines, Google, Yahoo, MSN, even today have a significant fraction of near-duplicates in their top results[1]. For example, consider the results of the query "emacs manual" using the Google search engine. We focus on the top 20 results (i.e., first 2 pages) as they represent the results most likely to be viewed by the user. Four of the results, `www.delorie.com/gnu/docs/emacs/emacs_toc.html`, `www.cs.utah.edu/dept/old/texinfo/emacs19/emacs_toc.html`, `www.dc.urkuamk.fi/docs/gnu/emacs/emacs_toc.html`, and `www.linuxselfhelp.com/gnu/emacs/html_chapter/emacs_toc.html`, on the first page (top-10 results), were highly similar—in fact, they had nearly identical content but different page headers, disclaimers, and logo images. For this particular query, on the whole, 7 out of 20 documents were redundant (3 identical pairs and 4 similar to one top page document). Similar results were found using Yahoo, MSN[2], and A9[3] search engines.

In this paper, we study the current state of popular search engines and evaluate the application of a Bloom filter based near-duplicate detection technique on search results. We demonstrate, using multiple search engines, how a number of results (ranging from 7% to 60%) on search queries are similar and can be eliminated or re-organized. Later, we explore the use of Bloom filters for finding similar objects and demonstrate their effectiveness in compactly representing and quickly matching pages for similarity testing. Although Bloom filters have been extensively used for set membership checks, they have not been analyzed for similarity detection between text documents. Finally, we apply our Bloom filter based technique to effectively remove similar search results and improve user experience. Our evaluation of search results shows that the occurrence of near-duplicates is strongly correlated to: i) the relevance of the document and ii) the popularity of the query. Documents that are considered more relevant and have a higher rank also have more near-duplicates compared to less relevant documents. Similarly, results from the more popular queries have more near-duplicates compared to the less popular ones.

Our similarity matcher can be deployed as a filter over

*This work was supported in part by the Texas Advanced Technology Program, the National Science Foundation (CNS-0411026), and an IBM Faculty Partnership Award.

†This work was done during an internship at IBM Almaden.

Eighth International Workshop on the Web and Databases (WebDB 2005), June 16-17, 2005, Baltimore, Maryland.

[1]Google does have a patent [17] for near-duplicate detection although it is not clear which approach they use.

[2]Results for a recently popular query, "ohio court battle" from both Google and MSN search had a similar behavior, with 10 and 4 out of the top 20 results being identical resp.

[3]A9 states that it uses a Google back-end for part of its search.

any search engine's result set. The overhead of integrating our similarity detection algorithm with search engines only associates about 0.4% extra bytes per document and provides fast matching on the order of milliseconds as described later in section 3. Note that we focus on one main aspect of similarity—text content. This might not completely capture the human-judgement notion of similarity in all cases. However, our technique can be easily extended to include link structure based similarity measures by comparing Bloom filters generated from hyperlinks embedded in web pages.

The rest of the paper is organized as follows. Similarity detection using Bloom filters is described and analyzed in Section 2. Section 3 evaluates and compares our similarity technique to improve search results from multiple engines and for different workloads. Finally, Section 4 covers related work and we conclude with Section 5.

2. SIMILARITY DETECTION USING BLOOM FILTERS

Our similarity detection algorithm proceeds in three steps as follows. First, we use content-defined chunking (CDC) to extract document features that are resilient to modifications. Second, we use these features as set elements for generating Bloom filters[4]. Third, we compare the Bloom filters to detect near-duplicate documents above a certain similarity threshold (say 70%). We start with an overview of Bloom filters and CDCs, and later present and analyze the similarity detection technique for refining web search results.

2.1 Bloom Filter Overview

A Bloom filter of a set U is implemented as an array of m bits [4]. Each element u ($u \in U$) of the set is hashed using k independent hash functions $h_1, \ldots, h_k$. Each hash function $h_i(u)$ for $1 \leq i \leq k$ maps to one bit in the array $\{1 \ldots m\}$. Thus, when an element is added to the set, it sets k bits, each bit corresponding to a hash function, in the Bloom filter array to 1. If a bit was already set it stays 1. For set membership checks, Bloom filters may yield a *false positive*, where it may appear that an element v is in U even though it is not. From the analysis in [8], given $n = |U|$ and the Bloom filter size m, the optimal value of k that minimizes the false positive probability, p^k, where p denotes that probability that a given bit is set in the Bloom filter, is $k = \frac{m}{n} \ln 2$. Previously, Bloom filters have primarily been used for finding set-membership [8].

2.2 Content-defined Chunking Overview

To compute the Bloom filter of a document, we first need to split it into a set of elements. Observe that splitting a document using a fixed block size makes it very susceptible to modifications, thereby, making it useless for similarity comparison. For effective similarity detection, we need a mechanism that is more resilient to changes in the document. CDC splits a document into variable-sized blocks whose boundaries are determined by its Rabin fingerprint matching a predetermined marker value [18]. The number of bits in the Rabin fingerprint that are used to match the marker determine the expected chunk size. For example, given a marker 0x78 and an expected chunk size of 2^k, a rolling (overlapping sequence) 48-byte fingerprint is computed. If the lower k bits of the fingerprint equal 0x78, a new chunk boundary is set. Since the chunk boundaries are content-based, any modifications should affect only a couple of neighboring chunks and not the entire document. CDC has been used in LBFS [15], REBL [13] and other systems for redundancy elimination.

2.3 Bloom Filters for Similarity Testing

Observe that we can view each document to be a set in Bloom filter parlance whose elements are the CDCs that it is composed of[5]. Given that Bloom filters compactly represent a set, they can also be used to approximately match two sets. Bloom filters, however, cannot be used for exact matching as they have a finite false-match probability but they are naturally suited for similarity matching.

For finding similar documents, we compare the Bloom filter of one with that of the other. In case the two documents share a large number of 1's (bit-wise AND) they are marked as similar. In this case, the bit-wise AND can also be perceived as the dot product of the two bit vectors. If the set bits in the Bloom filter of a document are a complete subset of that of another filter then it is highly probable that the document is included in the other. Web pages are typically composed of fragments, either static ones (e.g., logo images), or dynamic (e.g., personalized product promotions, local weather) [19]. When targeting pages for a similarity based "grouping", the test for similarity should be on the fragment of interest and not the entire page.

Bloom filters, when applied to similarity detection, have several advantages. First, the compactness of Bloom filters is very attractive for storage and transmission whenever we want to minimize the meta-data overheads. Second, Bloom filters enable fast comparison as matching is a bitwise-AND operation. Third, since Bloom filters are a complete representation of a set rather than a deterministic sample (e.g., shingling), they can determine inclusions effectively.

To demonstrate the effectiveness of Bloom filters for similarity detection, consider, for example, the pages from the Money/CNN web server (money.cnn.com). We crawled 103 MB of data from the site that resulted in 1753 documents. We compared the top-level page marsh_ceo/index.html with all the other pages from the site. For each document, we converted it into a canonical representation as described later in Section 3. The CDCs of the pages were computed using an expected and maximum chunk size of 256 bytes and 64 KB respectively. The corresponding Bloom filter was of size 256 bytes. Figure 1 shows that two other copies of the page one with the URI `/2004/10/25/news/fortune500/marsh\_ceo/index.htm` and another one with a dynamic URI `/2004/10/25/news/fortune500/marsh_ceo/index.htm?cnn=yes` matched with all set bits in the Bloom filter of the original document.

As another example, we crawled around 20 MB of data (590 documents) from the ibm web site (www.ibm.com). We compared the page `/investor/corpgovernance/index.phtml` with all the other crawled pages from the site. The chunk sizes were chosen as above. Figure 2 shows that two other pages with the URIs `/investor/corpgovernance/cgcoi.phtml` and `/investor/corpgovernance/cgblaws.phtml` appeared similar, matching in 53% and 69% of the bits in the Bloom filter, respectively.

To further illustrate that Bloom filters can differentiate between *multiple* similar documents, we extracted a technical documentation file 'foo' (say) (of size 17 KB) incrementally from a CVS archive, generating 20 different versions, with 'foo' being the original, 'foo.1' being the first version (with a change of 415 bytes from 'foo') and 'foo.19' being the last. As shown in Figure 3, the Bloom filter for 'foo' matched the most (98%) with the closest version 'foo.1'.

[4]Within a search engine context, the CDCs and the Bloom filters of the documents can be computed offline and stored.

[5]For multisets, we make each CDC unique before Bloom filter generation to differentiate multiple copies of the same CDC.

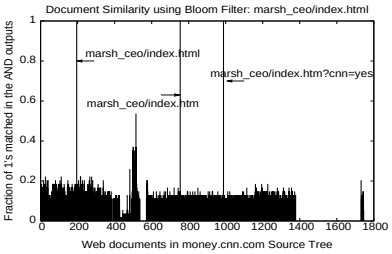

Figure 1: Comparison of the document marsh_ceo/index.html with all pages from the money.cnn.com web site

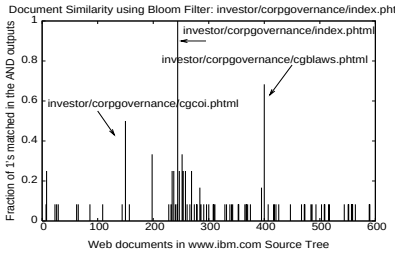

Figure 2: Comparison of the document investor/corpgovernance/index.phtml with pages from www.ibm.com

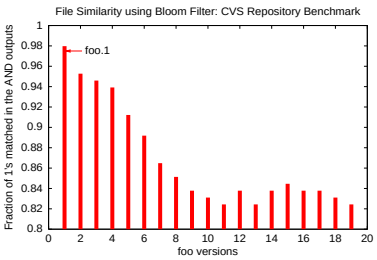

Figure 3: Comparison of the original file 'foo' with later versions 'foo.1', 'foo.2' ··· 'foo.19'

2.3.1 Analysis

The main consideration when using Bloom filters for similarity detection is the false match probability of the above algorithm as a function of similarity between the source and a candidate document. Extending the analysis for membership testing in [4] to similarity detection, we proceed to determine the expected number of *inferred* matches between the two sets. Let A and B be the two sets being compared for similarity. Let m denote the number of bits (size) in the Bloom filter. For simplicity, assume that both sets have the same number of elements. Let n denote the number of elements in both sets A and B i.e., $|A| = |B| = n$. As before, k denotes the number of hash functions. The probability that a bit is set by a hash function h_i for $1 \leq i \leq k$ is $\frac{1}{m}$. A bit can be set by any of the k hash functions for each of the n elements. Therefore, the probability that a bit is not set by any hash function for any element is $(1 - \frac{1}{m})^{nk}$. Thus, the probability, p, that a given bit is set in the Bloom filter of A is given by:

$$p \;=\; \left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right) \;\approx\; 1 - e^{-\frac{nk}{m}} \qquad (1)$$

For an element to be considered a member of the set, all the corresponding k bits should be set. Thus, the probability of a false match, i.e., an outside element is inferred as being in set A, is p^k. Let C denote the intersection of sets A and B and c denote its cardinality, i.e., $C = A \cap B$ and $|C| = c$.

For similarity comparison, let us take each element in set B and check if it belongs to the Bloom filter of the given set A. We should find that the c common elements will definitely match and a few of the other $(n - c)$ may also match due to the false match probability. By Linearity of Expectation, the expected number of elements of B inferred to have matched with A is

$$\text{E}[\#\text{ of inferred matches}] = (c) \;+\; (n - c)p^k$$

To minimize the false matches, this expected number should be as close to c as possible. For that $(n-c)p^k$ should be close to 0, i.e., p^k should approach 0. This happens to be the same as minimizing the probability of a false positive. Expanding p and under asymptotic analysis, it reduces to minimizing $(1 - e^{-\frac{nk}{m}})^k$. Using the same analysis for minimizing the false positive rate given in [8], the minima obtained after differentiation is when $k = \frac{m}{n} \ln 2$. Thus, the expected number of inferred matches for this value of k becomes

$$\text{E}[\#\text{ of inferred matches}] = c \;+\; (n - c)(0.6185)^{\frac{m}{n}}$$

Thus, the expected number of bits set corresponding to inferred matches is

$$\text{E}[\#\text{ of matched bits}] = m\left[1 - \left(1 - \frac{1}{m}\right)^{k\left(c \;+\; (n-c)(0.6185)^{\frac{m}{n}}\right)}\right]$$

Under the assumption of perfectly random hash functions, the expected number of total bits set in the Bloom filter of the source set A, is mp. The ratio, then, of the expected number of matched bits corresponding to inferred matches in $A \cap B$ to the expected total number of bits set in the Bloom filter of A is:

$$\frac{\text{E}[\#\text{ of matched bits}]}{\text{E}[\#\text{ total bits set}]} = \frac{\left(1 - e^{-\frac{k}{m}\left(c \;+\; (n-c)(0.6185)^{\frac{m}{n}}\right)}\right)}{\left(1 - e^{-\frac{nk}{m}}\right)}$$

Observe that this ratio equals 1 when all the elements match, i.e., $c = n$. If there are no matching elements, i.e., $c = 0$, the ratio $= 2(1 - (0.5)^{(0.6185)^{\frac{m}{n}}})$. For $m = n$, this evaluates to 0.6973, i.e., 69% of matching bits may be false. For larger values, $m = 2n, 4n, 8n, 10n, 11n$, the corresponding ratios are 0.4658, 0.1929, 0.0295, 0.0113, 0.0070 respectively. Thus, for $m = 11n$, on an average, less than 1% of the bits set may match incorrectly. The expected ratio of matching bits is highly correlated to the expected ratio of matching elements. Thus, if a large fraction of the bits match, then it's highly likely that a large fraction of the elements are common.

2.4 Discussion

Previous work on document similarity has mostly been based on shingling or super fingerprints. Using this method, for each object, all the k consecutive words of a document (called k-shingles) are hashed using Rabin fingerprint [18] to create a set of fingerprints (also called features or pre-images). These fingerprints are then sampled to compute a super-fingerprint of the document. Many variants have been proposed that use different techniques on how the shingle fingerprints are sampled (min-hashing, Mod_m, Min_s etc.) and matched [7, 6, 5]. While Mod_m selects all fingerprints whose value modulo m is zero; Min_s selects the set of s fingerprints with the smallest value. The min-hashing approach further refines the sampling to be the min values of say 84 random min-wise independent permutations (or hashes) of the set of all shingle fingerprints. This results in a fixed size sample of 84 fingerprints that is the resulting feature vector. To further simplify matching, these 84 fingerprints can be grouped as 6 "super-shingles" by concatenating 14 adjacent fingerprints [11]. In [13] these are called super-fingerprints. A pair of objects are then considered similar if either all or a large fraction of the values in the super-fingerprints match.

Our Bloom filter based similarity detection differs from the shingling technique in several ways. It should be noted, however, that the variants of shingling discussed above improve upon the original approach and we provide a comparison of our technique with these variants wherever applicable. First, shingling (Mod_m, Min_s) computes document similarity using the intersection of the two feature sets. In our approach, it requires only the bit-wise AND of the two Bloom filters (e.g., two 128 bit vectors). Next, shingling has a higher computational overhead as it first segments the document into k-word shingles ($k = 5$ in [11]) resulting in shingle set size

of about $S - k + 1$, where S is the document size. Later, it computes the image (value) of each shingle by applying set (say H) of min-wise independent hash functions (|H|=84 as used in [11]) and then for each function, selecting the shingle corresponding to the minimum image. On the other hand, we apply a set of independent hash functions (typically less than 8) to the chunk set of size on average $\lceil \frac{S}{c} \rceil$ where c is the expected chunk size (e.g., $c = 256$ bytes for $S = 8$ KB document). Third, the size of the feature set (number of shingles) depends on the sampling technique in shingling. For example, in Mod_m, even some large documents might have very few features whereas small documents might have zero features. Some shingling variants (e.g., Min_s, Mod_{2^i}) aim to select roughly a constant number of features. Our CDC based approach only varies the chunk size c, to determine the number of chunks as a trade-off between performance and fine-grained matching. We leave the empirical comparison with shingling as future work.

In general, a compact Bloom filter is easier to attach as a document tag and can be compared simply by matching the bits. Thus, Bloom filter based matching is more suitable for meta crawlers and can be added on to existing search engines without any significant changes.

3. EXPERIMENTAL EVALUATION

In this section, we evaluate Bloom filter-based similarity detection using several types of query results obtained from querying different search engines using the keywords posted on Google Zeitgeist `www.google.com/press/zeitgeist.html`, Yahoo Buzz `buzz.yahoo.com`, and MSN Search Insider `www.imagine-msn.com/insider`.

3.1 Methodology

We have implemented our similarity detection module using C and Perl. The code for content defined chunking is based on the CDC implementation of LBFS [15]. The experimental testbed used a 933 MHz Intel Pentium III workstation with 512 MB of RAM running Linux kernel 2.4.22. The three commercial search engines used in our evaluation are Google *www.google.com*, Yahoo Search *www.yahoo.com*, and MSN Search *www.msnsearch.com*. The Google search results were obtained using the GoogleAPI [1], for each of the search queries, the API was called to return the top 1000 search results. Although we requested 1000 results, the API, due to some internal errors, always returned less than 1000 entries varying from 481 to 897.

For each search result, the document from the corresponding URL was fetched from the original web server to compute its Bloom filter. Each document was converted into a canonical form by removing all the HTML markups and tags, bullets and numberings such as "a.1", extra white space, colons, replacing dashes, single-quotes and double-quotes with single space, and converting all the text to lower case to make the comparison case insensitive. In many cases, due to server unavailability, incorrect document links, page not found errors, and network timeouts, the entire set of requested documents could not always be retrieved.

3.1.1 Size of the Bloom Filter

As we discussed in the section 2, the fraction of bits that match incorrectly depends on the size of the Bloom filter. For a 97% accurate match, the number of bits in the Bloom filter should be 8x the number of elements (chunks) in the set (document). When applying CDC to each document, we use the expected chunk size of 256 bytes, while limiting the maximum chunk size to 64 KB. For an average document of size 8 KB, this results in around 32 chunks. The Bloom filter is set to be 8x this value i.e., 256 bits. To accommodate large documents, we set the maximum document size to 64 KB (corresponding to the maximum chunk size). Therefore, the Bloom filter size is set to be 8x the expected number of chunks (256 for document size 64 KB) i.e., 2048 bits or 256 bytes, which is a 3.2% and 0.4% overhead for document size of 8 KB and 64 KB respectively.

Example. When we applied the Bloom filter based matcher to the "emacs manual" query (Section 1), we found that the page `www.linuxselfhelp.com/gnu/emacs/html_chapter/emacs_toc.html` matched the other three, `www.delorie.com/gnu/docs/emacs/emacs_toc.html`, `www.cs.utah.edu/dept/old/texinfo/emacs19/emacs_toc.html`, and `www.dc.turkuamk.fi/docs/gnu/emacs/emacs_toc.html`, with 74%, 81% and 95% of the Bloom filter bits matching, respectively. A 70% matching threshold would have identified and grouped all these 4 pages together.

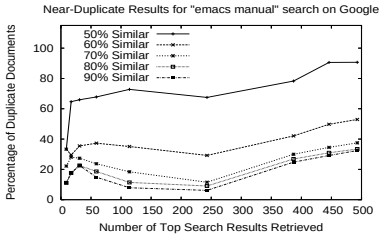

Figure 4: "emacs manual" query search results (Google)

3.2 Effect of the Degree of Similarity

In this section, we evaluate how the degree of similarity affects the number of documents that are marked similar. The degree of similarity is the percentage of the document data that matches (e.g., a 100% degree of similarity is an identical document). Intuitively, the higher the degree of similarity, the lower the number of documents that should match. Moreover, the number of documents that are similar depends on the total number of documents retrieved by the query. Although, we initially expected a linear behavior, we observed that the higher ranked results (the top 10 to 20 results) were also the ones that were more duplicated.

Using GoogleAPI, we retrieved 493 results for the "emacs manual" query. To determine the number of documents that are similar among the set of retrieved documents, we use a union-find data structure for clustering Bloom filters of the documents based on similarity. Figure 4 shows that for 493 documents retrieved, the number of document clusters were 56, 220, 317, 328, 340, when the degree of similarity was 50, 60, 70, 80, 90%, respectively. Each cluster represents a set of similar documents (or a single document if no similar ones are found). We assume that a document belongs to a cluster if it is similar to a document in the cluster, i.e., we assume that similarity is transitive for high values of the degree of similarity (as in [9]). The fraction of duplicate documents as shown in figure 4, decreases from 88% to 31% as the degree of similarity increases from 50% to 90%. As the number of retrieved queries increase from 10 to 493, the fraction of duplicate documents initially decrease and then increase forming a minima around 250 results. The decrease was due to the larger aliasing of "better" ranked documents. However, as the number of results increase, the initial set of documents get repeated more frequently, increasing the number of duplicates. Similar results were obtained for a number of other queries that we evaluated.

3.3 Effect of the Search Query Popularity

To get a representative collection of the types of queries

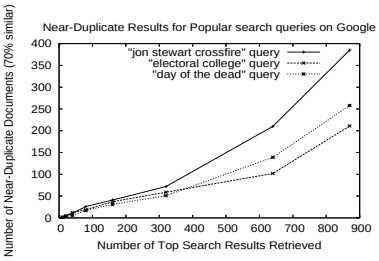

Figure 5: Search results for the top 3 queries on Google

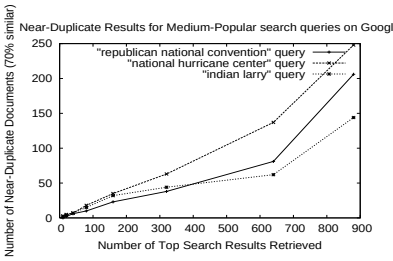

Figure 6: Search results for 3 medium-popular queries on Google

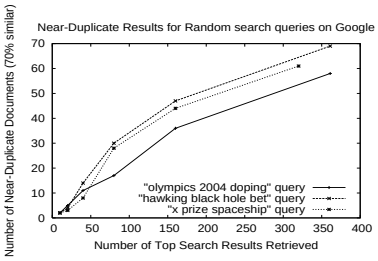

Figure 7: Search results for 3 random queries on Google

performed on search engines, we selected samples from Google Zeitgeist (Nov. 2004) of three different query popularities: i) Most Popular, ii) Medium-Popular, and iii) Random.

For most-popular search queries, the three queries selected in order were—"jon stewart crossfire"(TP1), "electoral college"(TP2) and "day of the dead"(TP3). We computed the number of duplicates having 70% similarity (atleast 70% of the bits in the filter matched) in the search results. Figure 5 shows the corresponding number of duplicates for a maximum of 870 search results from the Google search API. The TP1 query had the maximum fraction of near-duplicates, 44.3%, while the other two TP2 and TP3 had 29.7% and 24.3%, respectively. Observe that the most popular query TP1was the one with the most duplicates.

For the medium popular queries, we selected three queries from the list "Google Top 10 Gaining Queries" for the week ending Aug. 30, 2004 on the Google Zeitgeist—"indian larry" (MP1), "national hurricane center"(MP2) and "republican national convention"(MP3). Figure 6 shows the corresponding search results having 70% similarity for a maximum of 880 documents from the Google search engine. The fraction of near-duplicates among 880 search results ranged from 16% for MP1 to 28% for MP2.

For a non-popular query sample, we selected three queries at random—"olympics 2004 doping", "hawking black hole bet", and "x prize spaceship". The Google API retrieved only about 360 results for the first two queries and 320 results for the third query. Figure 7 shows the number of near-duplicate documents in the search results corresponding to the three queries. The fraction of near-duplicates in all these queries were in the same range, around 18%.

As we observed earlier, as the popularity of queries decrease so do the number of duplicate results. The most popular queries had the largest number of near-duplicate results, the medium ones fewer, and the random queries the lowest.

3.4 Behavior of different search engines

The previous experiments all compared the results from the Google search engine. We next evaluate the behavior of all three search engines, Google, Yahoo and MSN search in returning near-duplicate documents for the 10 popular queries featured on their respective web sites. To our knowledge, Yahoo and MSN search do not provide an API similar to the GoogleAPI for doing automated retrieval of search results. Therefore, we manually made HTTP requests to the URLs corresponding to the first 50 search results for a query.

We plot minimum, average and maximum number of near-duplicate (atleast 70% similar) search results in the 10 popular queries. The three whiskers on each vertical bar in Figures 8,9,10 represent min., avg., and max. in order. Figure 8 shows the results for Google, with average number of near-duplicates ranging from 7% to 23%. Figure 9 shows near-duplicates in Yahoo results ranging from 12% to 25%. Figure 10 shows the results for MSN, where the near-duplicates range from 18% to 26%. Comparing the earlier "emacs manual" query, MSN had 32% near duplicates while Yahoo had 22%. These experiments support our hypothesis that current search engines return a significant number of near-duplicates. However, these results do not in any way suggest that any particular search engine performs better than the others.

3.5 Analyzing Response Times

In this section, we analyze the response times for performing similarity comparisons using Bloom filters. The timings include (a) the (offline) computation time to compute the document CDC hashes and generating the Bloom filter, and (b) the (online) matching time to determine similarity using bitwise AND on Bloom filters and time for insertions and unions in a union-find data structure for clustering.

Exp. Chunk Sizes File Size	256 Bytes (ms)	512 Bytes (ms)	2 KB (ms)	8 KB (ms)
10 KB	0.3	0.3	0.2	0.2
100 KB	4	3	3	2
1 MB	29	27	26	24
10 MB	405	321	267	259

Table 1: CDC hash computation time for different files and expected chunk sizes

Document Size	# of chunks (n)	k = 2 (ms)	k = 4 (ms)	k = 8 (ms)
10 KB	35	11	12	14
100 KB	309	118	120	126
1 MB	2959	961	1042	1198
10 MB	30463	11792	11960	12860

Table 2: Time (ms) for Bloom filter generation for different document sizes (expected chunk size 256 bytes)

Bloom Filter Size (Bits)	100	300	625	1250	2500	5000
Time (μsec)	1.9	2.4	2.9	3.9	6.2	10.7

Table 3: Time (microseconds) for computing the bitwise AND of Bloom filters for different sizes

Table 1 shows the CDC hash computation times for a complete document (of size 10 KB, 100 KB, 1 MB, 10 MB) for different expected chunk sizes (256 bytes, 512 bytes, 2 KB, 8 KB). The Bloom filter generation times are shown in Table 2 for different values (2, 4, 8) of the number of hash functions (k) and different number of chunks (n). Although the Bloom filter generation times appear high relative to the CDC times, it is more an artifact of the implementation of the Bloom filter code in Perl instead of C and not due to any inherent complexity in the Bloom filter code. A preliminary implementation in C reduced the Bloom filter generation time by an order of magnitude.

For the matching time overhead, Table 3 shows the pairwise matching time for two Bloom filters for different filter

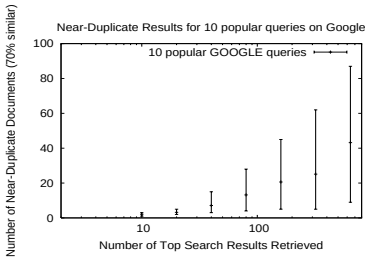

Figure 8: Search results for 10 popular queries on Google

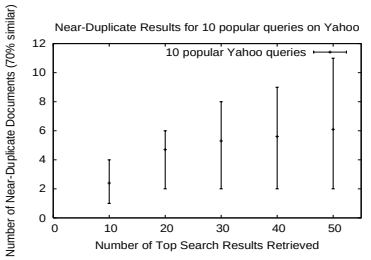

Figure 9: Search results for 10 popular queries on Yahoo Search

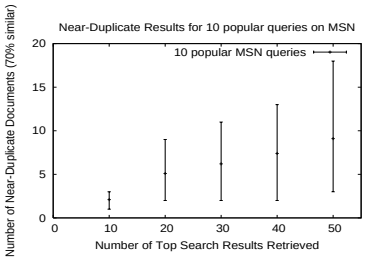

Figure 10: Search results for 10 popular queries on MSN Search

No. of Results Search Query	10	20	40	80	160	320
"emacs manual"	1	4	15	66	286	1233
"ohio court battle"	1	7	24	98	369	1426
"hawking black hole bet"	1	6	23	88	364	1407

Table 4: Matching and Clustering time (in ms)

sizes ranging from 100 bits to 5000 bits. The overall matching and clustering time for different query requests is shown in Table 4. Overall, using untuned Perl and C code, for clustering 80 results each of size 10 KB for the "emacs manual" query would take around 80*0.3 ms + 80* 14 ms + 66ms = 1210 ms. However, the Bloom filters can be computed and stored apriori reducing the time to 66 ms.

4. RELATED WORK

The problem of near-duplicate detection consists of two major components: (a) extracting document representations aka features (e.g., shingles using Rabin fingerprints [18], super-shingles [11], super-fingerprints [13]), and (b) computing the similarity between the feature sets. As discussed in Section 2, many variants have been proposed that use different techniques on how the shingle fingerprints are sampled (e.g., min-hashing, Mod_m, Min_s) and matched [7, 6, 5]. Google's patent for near-duplicate detection uses another shingling variant to compute fingerprints from the shingles [17].

Our similar detection algorithm uses CDC [15] for computing document features and then applies Bloom filters for similarity testing. In contrast to existing approaches, our technique is simple to implement, incurs only about 0.4% extra bytes per document, and performs faster matching using only bit-wise AND operations. Bloom filters have been proposed to estimate the cardinality of set intersection in [8] but have not been applied for near-duplicate elimination in web search. We recently learned about Bloom filter replacements [16] which we will explore in the future.

Page and site similarity has been extensively studied for web data in various contexts, from syntactic clustering of web data [7] and its applications for filtering near duplicates in search engines [6] to storage space and bandwidth reduction for web crawlers and search engines. In [9], replica identification was also proposed for organizing web search results. Fetterly et al. examined the amount of textual changes in individual web pages over time in the PageTurner study [12] and later investigated the temporal evolution of clusters of near-duplicate pages [11]. Bharat and Broder investigated the problem of identifying mirrored host pairs on the web [3]. Dasu et al. used min hashing and sketches to identify fields having similar values in database tables [10].

5. CONCLUSIONS

In this paper, we applied a Bloom filter based similarity detection technique to refine the search results presented to the user. Bloom filters compactly represent the entire document and can be used for quick matching. We demonstrated how a number of results of popular and random search queries retrieved from different search engines, Google, Yahoo, MSN, are similar and can be eliminated or re-organized.

6. ACKNOWLEDGMENTS

We thank Rezaul Chowdhury, Vijaya Ramachandran, Sridhar Rajagopalan, Madhukar Korupolu, and the anonymous reviewers for giving us valuable comments.

7. REFERENCES

[1] Google web apis (beta), *http://www.google.com/apis*.
[2] Yahoo results getting more similar to google `http://www.searchenginejournal.com/index.php?p=584&c=1`.
[3] K. Bharat and A. Broder. Mirror, mirror on the web: a study of host pairs with replicated content. *Comput. Networks*, 31(11-16):1579–1590, 1999.
[4] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
[5] A. Z. Broder. On the resemblance and containment of documents. In *SEQUENCES*, 1997.
[6] A. Z. Broder. Identifying and filtering near-duplicate documents. In *COM*, pages 1–10, 2000.
[7] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. In *WWW'97*.
[8] A. Z. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. In *Allerton'02*.
[9] J. Cho, N. Shivakumar, and H. Garcia-Molina. Finding replicated web collections. *SIGMOD Rec.*, 2000.
[10] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *SIGMOD*, 2002.
[11] D. Fetterly, M. Manasse, and M. Najork. On the evolution of clusters of near-duplicate web pages. In *LA-WEB*, 2003.
[12] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of web pages. In *WWW*, 2003.
[13] P. Kulkarni, F. Douglis, J. D. LaVoie, and J. M. Tracey. Redundancy elimination within large collections of files. In *USENIX Annual Technical Conference, General Track*, pages 59–72, 2004.
[14] J. C. Mogul, Y.-M. Chan, and T. Kelly. Design, implementation, and evaluation of duplicate transfer detection in HTTP. In *NSDI*, pages 43–56, 2004.
[15] A. Muthitacharoen, B. Chen, and D. Mazieres. A low-bandwidth network file system. In *SOSP*, 2001.
[16] R. Pagh, A. Pagh, and S. S. Rao. An optimal bloom filter replacement. In *SODA*, 2005.
[17] W. Pugh and M. Henzinger. Detecting duplicate and near-duplicate files, US Patent # 6658423.
[18] M. O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Harvard University, 1981.
[19] L. Ramaswamy, A. Iyengar, L. Liu, and F. Douglis. Automatic detection of fragments in dynamically generated web pages. In *WWW*, 2004.