A Comparison of Techniques to Find Mirrored Hosts on the WWW

Krishna Bharat
Compaq SRC
130 Lytton Ave.
Palo Alto, CA 94301
bharat@pa.dec.com

Andrei Broder
AltaVista Company
1825 S. Grant St.
San Mateo, CA 94402
andrei.broder@altavista.com

Jeffrey Dean
Google Inc.
165 University Ave
Palo Alto, CA 94301
jdean@google.com

Monika R. Henzinger
Compaq SRC
130 Lytton Ave.
Palo Alto, CA 94301
monika@pa.dec.com

ABSTRACT

We compare several algorithms for identifying mirrored hosts on the World Wide Web. The algorithms operate on the basis of URL strings and linkage data: the type of information easily available from web proxies and crawlers.

Identification of mirrored hosts can improve web-based information retrieval in several ways: First, by identifying mirrored hosts, search engines can avoid storing and returning duplicate documents. Second, several new information retrieval techniques for the Web make inferences based on the explicit links among hypertext documents – mirroring perturbs their graph model and degrades performance. Third, mirroring information can be used to redirect users to alternate mirror sites to compensate for various failures, and can thus improve the performance of web browsers and proxies.

We evaluated 4 classes of "top-down" algorithms for detecting mirrored host pairs (that is, algorithms that are based on page attributes such as URL, IP address, and connectivity, and not on the page content) on a collection of 140 million URLs (on 230,000 hosts) and their associated connectivity information. Our best approach is one which combines 5 algorithms and achieved a precision of 0.57 for a recall of 0.86 considering 100,000 ranked host pairs.

1 Introduction

The explosive growth of the World Wide Web (WWW) has created both challenges and opportunities for the information retrieval discipline. As many papers in the SIGIR'98 "Workshop on Hypertext Information Retrieval for the Web" made clear, searching for information on the WWW differs from information retrieval in classical collections in many respects. Two of the most obvious are the very large size of the Web, estimated at 275 million pages as of March 1998, and growing at the rate of 20 million pages a month [3], and the prevalence of systematic content duplication. The fraction of the total WWW collection consisting of duplicates and near-duplicates has been estimated at 30 to 45%. (See [7] and [16].)

Duplication has both positive and negative aspects. On one hand the redundancy makes retrieval easier: if a search engine has missed one copy, maybe it has the other; or if one page has become unavailable, maybe a replica can be retrieved. On the other hand, from the point of view of search engines storing duplicate content is a waste of resources and from the user's point of view, getting duplicate answers in response to a query is a nuisance.

The principal reason for duplication on the Web is the systematic replication of content across distinct hosts, a phenomenon known as "mirroring" (These notions are defined more precisely below.) It is estimated that at least 10% of the hosts on the WWW are mirrored [2, 12]. The aim of this paper is to present and evaluate algorithms for detecting mirroring on the WWW in an information retrieval framework. We start with some definitions.

Each document on the WWW has a unique name called the Universal Resource Locator (URL). The URL consists of three disjoint parts, namely the *access method*, a *hostname*, and a *path*. For example in the URL

`http://www.research.digital.com/SRC/`

the string `http` defines the access method,

`www.research.digital.com`

is the hostname, and `SRC/` is the path.

The hostname identifies a *host* also known as a *web server* on which the document is stored. In general it may also contain a port identifier. A host stores a collection of documents which all share the same hostname. Each document on a host is identified by its path.

The hostname gets translated by a *name server* into an "Internet Protocol (IP) address" represented as four octets (bytes). This relation is many to many, that is, several hosts with different hostnames can share the same IP address ("virtual hosting") or a host may have a set of associated IP addresses, in which case the name server returns an arbitrary member of the set.

Two hosts are mirrors if they contain the same set of documents identified by the same paths. However, replicated documents on the Web tend to differ slightly (for example, because of local customizations or dynamically generated parts of the content such as time-stamps or advertisements), and hence we use the following relaxed definition of a mirror:

> Two hosts A and B are mirrors iff for every document on A there is a *highly similar* document on B with the same path, and vice versa.

Highly similar is a subjective measure. We made this notion precise by adopting the resemblance distance described in [7] that experimentally captures well the informal notion of "roughly the same." The technique efficiently computes the syntactic resemblance between two documents as a fractional score between 0 and 1. The higher the score, the greater the resemblance. (See section 4.1.3 for more details.) Any edit-distance measure that computes a similar resemblance score can be substituted.

Reliable algorithms for finding mirrors can improve web-based information access in several ways:

- Although search engines tend to filter out exact duplicates and even some near duplicates, considerable duplicate content stored on mirrored hosts escapes detection. This wastes index space and annoys users. Mirror detection can alleviate this problem.

- Text based techniques that work well on classical collections can return non-authoritative results when applied to the Web. Recent web IR research has begun exploiting connectivity (i.e., linkage between documents) to compute a document's authority score. Examples of such techniques are PageRank [13], CLEVER [9, 8, 11], and Topic Distillation [4]. The assumption underlying these algorithms is that each linking document provides an independent testimonial to the quality of the linked document. Mirroring violates this independence assumption. Hence documents pointed to by mirrors get artificially higher scores. Conversely, mirrored documents get lower scores than they normally would. Collapsing all mirrored hosts into one solves both problems.

- Transient web server failures frustrate Web users. If a mirroring relationship is known for the failed server, such "broken links" can be made invisible by fetching the document from a mirrored host.

- Web browsers and web proxies cache documents to improve the speed at which information is accessed. A document in the cache can potentially be used to satisfy a request for the same path on a mirrored host as well.

If not for the size of the Web, detecting mirrors would be a simple information retrieval problem: imagine that we have for each host a pseudo document consisting of all the pages on that host labeled by their paths. Then finding mirrors reduces to computing pair-wise content similarity between these pseudo documents. However, the scale of the Web makes this prohibitively expensive if not impossible.

We believe that low-cost techniques based solely on URLs, and requiring no access to the content of the pages, may be sufficient for this problem. Search engine crawlers and large Web proxies encounter URLs from a significant fraction of the Web, but no list of URLs is complete and most hosts tend to be only partially sampled. Typically the content associated to these URLs is not stored, only their names and possibly some information about the hyperlinks among them. This makes our problem more challenging: Given an incomplete list of URLs on the Web and possibly some information about their connectivity, how does one compute a list of mirrors on the Web? If term vector matching is to be used, how should the terms be selected and weighted? How does one evaluate such mirror detection algorithms? Since relevance judgements are tedious to perform manually on the scale of the Web, how can this be automated?

Fortunately the classic measures of precision and recall are perfectly suited for evaluating mirror detection methods: we simply require each algorithm to extract from the input those pairs of hosts most likely to be mirrors under its detection criteria and list in the order of likelihood. An automated, but relatively slow, validation technique plays the role of relevance judgements and allows us to analyze the ranked output in the usual manner. In practice this technique might be used to filter the output of a ranking algorithm to produce a clean list of mirrors.

In this paper we discuss and evaluate four classes of algorithms for finding mirrored hosts on the Web. It turns out that the best approach is a combination of 5 algorithms: on our test data achieved a precision of 0.57 for a recall of 0.86 considering 100,000 results.

Our algorithms are all "top-down" algorithms, that is, all our algorithms are based on page attributes such as URL, IP address, and connectivity, and not on the page content. We are essentially using the replicated structure of mirrors to identify them. There is an alternative "bottom up" approach presented by Cho, Shivakumar, and Hector Garcia-Molina in [10] whereby in the first stage of the algorithm copies of a given pages are clustered together, and then these clusters are grown until they represent an entire site.

There are advantages and disadvantages to each approach: the "top down" structural approach has the advantage that it needs only the URLs of pages, not the pages themselves. A crawler typically "knows" about many more url's than it crawls. Similarly a smart cache can accumulate a collection of valid url's much larger than the collection of pages ever stored. Another advantage is that mirrors can be discovered even when very few of their duplicate pages are simultaneously present in the collection; it suffices that enough of the replicated structure be visible (that is, enough shared paths prefixes). A third advantage is that if replicated pages are crawled at different times and they have changed enough in the meantime it might thwart the "bottom-up" approach. An advantage of the bottom-up approach is that it might discover mirrors even under renaming of paths, and it might discover mirrors that are too small for the "top down" approach.

It would be interesting to combine and/or compare the two approaches and we intend to pursue this idea in future research.

The remainder of the paper is organized as follows: Section 2 presents an overview, and section 3 describes our algorithms. The performance of the algorithms is compared in section 4. We discuss related work in section 5.

2 Methodology

As described above the input to our algorithms is a set of URLs occurring on the Web. Our approach does not assume that all URLs from a given host are present in the input. For some algorithms we also consider the hyperlinks between documents.

In our experiments the input contained approximately 179 million URLs which were among those found on web pages by AltaVista during a crawl. Not all these URLs were valid pages, because this set included URLs that were not crawled. We tested algorithms which use this input to generate a list of mirrored hosts ranked in decreasing order of likelihood.

We filtered the list of URLs to only include URLs from hosts that had at least 100 URLs in the input. We felt that hosts with fewer URLs were either too small to be mirrors or were inadequately represented in the input. Additionally, URLs with a small number of samples were often misspellings of valid URLs and sometimes the host did not exist. This makes it hard to determine the number of valid hosts in the input. Restricting the set of hosts in this way reduced the input to 140.6 million URLs on 233,035 hosts and hyperlinks between them (amounting to 11.2 Gigabytes of storage uncompressed). Only mirroring relations between host pairs drawn from this set were determined.

We used four different approaches:

1. *IP Address Based:* These algorithms list hosts that have identical or highly similar IP addresses.
2. *URL String Based:* These algorithms list host pairs that have URL strings that are highly similar. Term vector matching is done with terms being generated from URL strings in four different ways.
3. *URL String and Connectivity Based:* This algorithm exploits both the URL strings of the documents in the input as well as their hyperlinks. The intuition here is that two hosts are mirrors if they share many common paths, and documents with the same path have similar outlinks.
4. *Host Connectivity Based:* These algorithms consider all documents on a host as a single large document and analyze the linkage between these pseudo-documents. The basic idea is that two hosts are mirrors if they link to similar sets of hosts.

Each algorithm was executed on the input and generated a ranking list of probable mirrored hosts. We tested the top 30,000 host pairs from each list to see if they were indeed mirrors, and used the resulting pool of mirrors to measure precision and relative recall. (See section 4.2.2 for our definition of relative recall at a rank.) The testing was done automatically using a method that is designed to be correct with high probability, but that is comparatively very slow. Details are in the evaluation section below.

3 Algorithms

We describe each algorithm in turn:

3.1 IP Address Based (Algorithms *IP3* and *IP4*)

If exactly two hosts translate into the same IP address, it is likely that they are mirrors accessing the same server, and that the two hostnames are just aliases. A match on the first 3 octets signifies web servers on the same subnet. These usually part of the same organization, and are often mirrored hosts, replicated to handle the traffic.

However, when many hosts resolve to the same IP address it is indicative of virtual hosting by an internet service provider: for instance, at the time of writing, Webcom uses the same IP address, 209.1.28.62, to implement 18,000 distinct web sites.

These considerations led us to two algorithms:

- Algorithm *IP4:* We cluster hosts with the same IP address. The larger the size of the cluster, the less likely it is that all the hosts in the clusters are mirrors. Processing clusters in the increasing order of size we enumerate up to 200 host pairs from each cluster (If a cluster induces more than 200 pairs we pick 200 pairs at random.)

- Algorithm *IP3:* Same as the above case but we instead cluster based solely on the first three octets of the IP address. Since now there are many more clusters we list at most 5 pairs of hosts per cluster.

3.2 URL String Based

URL strings provide information about a potential mirroring relationship between hosts in two different ways:

1. Similar hostnames suggest that hosts belong to the same or related organizations.
2. Similar paths indicate a possible replication of directories. This is because paths usually reflect the host's directory structure.

Term vector matching [15] is used to compute the likelihood that a pair of hosts are mirrors. Based on the type of term used we get four different algorithms, each with a corresponding weighting scheme. These are summarized below:

3.2.1 Hostname Matching (Algorithm *hosts*)

- *Term Selection*: Substrings of the hostname are used as terms. Specifically, substrings delimited by a period ('.') or the start or end of the hostname. When the host is specified as an IP address we use the first two, three and four octets as terms.

  Only terms occurring in less than 100 hosts (i.e., with document frequency less than 100) are used in the host term vectors. As a further optimization, if document frequency is greater than 25 we restrict the term to appear in the vectors of only 25 of the hosts chosen at random. We had 73,736 distinct terms appearing in more than one host. Of these 185 were rejected and 937 were restricted.

- *Term Weighting*: The weight of term t is computed in terms of its document frequency $df(t)$, and $len(t)$, which is the number of segments obtained by breaking the term at '.' characters. The weight is computed as $\frac{\log(len(t))}{1+\log(df(t))}$. This favors substrings composed of many segments which are likely to be very specific.

3.2.2 Full Path Matching (Algorithm *paths*)

- *Term Selection*: The entire path is used as a term. The same document frequency based restrictions in assigning terms to vectors are applied as in the previous cases. We had 6,897,285 distinct paths occurring in more than one host. Of these 4,121 were rejected and 24,430 were restricted.

- *Term Weighting*: The weight of term t is computed in terms of its document frequency $df(t)$, and $maxdf$, which is the maximum document frequency of any term in the collection. Since we discard all terms with document frequencies greater than 100, $maxdf$ is effectively 100. The weight is computed as $1 + \log(\frac{maxdf}{df(t)})$.

3.2.3 Prefix Matching (Algorithm *prefix*)

- *Term Selection*: Prefixes of the path that either end in a '/' or terminate the path are enumerated. This results in a large number of terms per host of which we keep only the $10 \log p$ terms with largest term frequency, where p is the number of documents from the host in the input set. The same document frequency based restrictions in assigning terms to vectors are applied as in the previous cases. We had 646,291 distinct prefixes selected from more than one host. Of these 1,996 were rejected and 8,241 were restricted.

- *Term Weighting*: We used the same term weighting scheme as in the case of path matching. However, to compensate for the fact that smaller hosts tend to contribute fewer terms we normalized the host similarity weight w by a multiplication factor $\frac{1}{0.1+0.15(\log(n_1)+log(n_2))}$, where n_1 and n_2 represent the number of URL strings in the input from each of the two hosts. In the average case a host contributes about 1000 URLs to the input, which corresponds to a scaling factor of 1. All hosts contribute at least 100 URLs to the input, and the largest host was $www.geocities.com$ which contributed about 12 million URLs.

3.2.4 Positional Word Bigram Matching (Algorithm *shingles*)

- *Term Selection*: The path is broken into a list of words by treating '/' and '.' as breaks. These symbols are commonly used separators in URL naming. Each resulting word is normalized by eliminating non-alphanumeric characters replacing every sequence of digits with '*'. This has an effect similar to stemming. To create terms we combine successive pairs of words in the list. To make the terms location specific within the path we also append the ordinal position of the first word in the list to the term. We call these terms, *positional word bigrams*.

  Thus, given the path

  `conferences/dl99/advanceprogram.html`,

  we first create the list (`conferences`, `dl*`, `advanceprogram`, `html`). Then we combine successive pairs of words with the index of the first word to generate the positional word bigrams:

  ```
  conferences_dl*_0
  dl*_advanceprogram_1
  advanceprogram_html_2
  ```

  Our attempts to use single words and pairs of words resulted in a lot of false matches, due to the repetitive nature of URL names. Adding position makes the terms more effective.

  The same tf based selection of terms, and document frequency based restrictions in assigning terms to vectors are applied as in prefix matching. We had 734,968 distinct terms selected from more than one host. Of these 2,148 were rejected and 9,682 were restricted.

- *Term Weighting*: The same weighting and normalization scheme is used as in the case of prefix matching.

3.3 URL String and Connectivity Based (Algorithm *conn*)

The identification of mirrors can be aided by the use of connectivity information. We extend the URL string based *paths* algorithm (which selects host pairs with many common paths), with a connectivity based filtering stage. This involves testing for each common path if it has the same set of out-links on both hosts under consideration. Host pairs that fail the test are eliminated. There are two considerations however:

- Since content changes over time (e.g., at news sites) and mirrors engage in local customizations, the set of out-links in replicated documents often varies slightly. Hence, when testing a common path for equivalence we require only that a fraction f of the union of the set of out-links from the two versions be common to both. Since we wanted a high degree of agreement, f was 0.9 in our experiment.

- Links to other documents on the same host will involve the name of the local host and will hence appear to differ. We compensate by removing the hostname from such URLs, effectively turning them into relative URLs.

Given a host pair $< host_1, host_2 >$ from the output of *paths* we test some of the common paths for equivalence as described above. In particular, we select n paths from each host with the highest outdegree. The intuition is that documents with many links are likely to provide more evidence of a mirroring relationship than those with a small number of links. For efficiency, we chose n to be 10. Anecdotal evidence showed that larger values of n did not improve the performance by much.

After testing a total of $2 * n$ common paths, if a fraction m of the paths were found to be equivalent (m was 0.75 in our experiment) then the host pair is allowed to remain in the ranking. Otherwise, it is eliminated.

3.4 Host Connectivity-Based (Algorithms *hconn1* and *hconn2*)

The host connectivity based algorithms constructs an edge-weighted graph where each node corresponds to a host. The graph contains a directed edge from node A to node B if there exists a document on host A that points to a document on host B. The weight of the edge is the total number of such links. This graph is called the *host graph*. Two hosts are likely to be mirrors if their nodes point to the same nodes in the host graph. However, since our input is incomplete and due to local and temporal variations in content, nodes for mirrored hosts tend to have a *similar* rather than identical set of out-edges.

As before, we use term vector matching to compute the likelihood of a host pair being mirrors. The set of nodes that a host's node points to are used as terms. Note that the *document frequency* of a term is precisely the indegree of the node it corresponds to. The *term weight* we use is based solely on inverse document frequency but not on *term frequency*. This is because many hosts tend to be poorly sampled and hence we only have a lower bound on term frequencies. However, we do use term frequency in term selection. Initially our host graph had 126,853,032 edges. For efficiency we pruned the graph by keeping only $10 \log p$ edges with largest weight per host (corresponding to terms with highest *term frequency*), where p is the number of documents on that host in the input set. This reduced the number of edges in the host graph to 9,519,003.

For algorithm *hconn1* we assigned the following term weight to each term t: If the indegree $in(t)$ of node t is at most 25, the term weight is 1. Otherwise the term weight of t is $25/in(t)$ for $in(t) <= 125$. For $in(t) > 125$ it is $\min\big(1/5, 200/in(t)\big)$. The term weight is linear in $1/in(t)$ (as opposed to $\log\big(1/in(t)\big)$) since computing an approximation of this term weight can be implemented efficiently by allowing the term to appear only in vectors for 25 hosts chosen at random for $25 < in(t) <= 125$ and $\min(200, in(t)/5)$ random hosts beyond that.

For algorithm *hconn2* we multiplied the above term weight with $1+\log(\frac{maxin}{in(t)})$, where $maxin$ is the maximum indegree of any node in the host graph.

4 Evaluation

4.1 Methodology

We adopted a relevance judgement technique similar to TREC pooling [17] to evaluate the list of ranked host pairs returned by our various mirror-detection algorithms. The top 30,000 host pairs from each ranking were tested for a mirroring relationship. The limit of 30,000 was a product of time restrictions. However, the threshold was picked to be at least more than 23,000, which is the number of hosts with a mirroring relationship discovered in a previous experiment [2]. Some rankings returned several million host pairs. Testing all of them would have been impossible. Even considering 30,000 from each host, after accounting for duplicates this gave us a total of 118,790 unique pairs of hosts to check. If human judgement were required the task would have been prohibitive. Instead we used the *mirroring test* described below. Since the test requires fetching documents from the WWW this approach is more attractive as a testing or filtering technique than as a host pair selection technique. The results of the tests allowed us to compute precision vs rank (Figure 1), and relative recall vs rank (Figure 3) for ranks up to 25,000. Also, using the combined pool of known mirrors from the various tests we were able to plot precision vs recall (Figure 4) for the top 100,000 results from each ranking.

4.1.1 *Mirroring Test*

To test if $host_1$ and $host_2$ are mirrors we alternately sample pages from each of the hosts, and for each sample check if the other host has an equivalent page. Rather than insist on identical content, we use the scheme described in [7] to check if the documents have a *high syntactic similarity*. See section 4.1.3 for more on this. We use a 50% threshold for similarity, which corresponds to an estimation that with high probability at least 50% of the unique k-word-sequences found in the two documents are common to both documents. Since the tests are repeated on multiple paths the threshold can afford to be liberal. For the same reason the performance was not sensitive to the exact value chosen. Given dynamic content however, a threshold of 75% was found to be too conservative. This compensates for variations in content due to local server-side includes, variable banner ads, local URL names, transaction-ids, and small amounts of dynamic content.

Initially we compare the root pages for equivalence. If the root pages are not equivalent, the test reports a mismatch. If one of the root pages refuses to load we record a failure of the corresponding host and abandon further testing. Otherwise, we continue to check if a set of paths sampled from $host_1$ are all valid on $host_2$ and yield highly similar documents, and vice versa.

To select paths from a host we sort its paths lexicographically, and sample 15 paths at equal intervals in the list. This ensures a broad coverage of the host's directory structure. We permute both lists randomly and test paths in alternation from the two lists. Paths failing on both hosts are are ignored. If a path succeeds on both hosts but the pages are not highly similar, we record a mismatch and abandon further testing. If it fails on one host and not on the other it suggests that the hosts are only partial mirrors. Further testing is abandoned. We record only a tentative mismatch since the access failure may be a temporary phenomenon, as is common on the WWW. All tentative mismatches are tested again at a later point in time. If a mismatch is seen again it is taken as final. If 10 paths yield highly similar documents we conclude that the hosts are indeed mirrors.

The automatic testing procedure takes from a few seconds to a maximum of 2 minutes per URL (since we had a 2 minute timeout on URL fetches). Because of slow and heavily loaded servers the timeout was often exceeded. When several paths from a host timeout the host pair is rescheduled for later testing.

4.1.2 Pooling Evaluations Transitive inferences are possible of the form:

```
if MIRROR(A,B) AND MIRROR(B,C)
THEN MIRROR(A,C)
if MIRROR(A,B) AND MISMATCH(B,C)
THEN MISMATCH(A,C)
```

Transitive inferences allow us to avoid testing some of the host pairs. To do this we pool the results of the tests as they become available and compute transitive relations as described above. For every host that is part of a cluster of mirrors, we maintain its *cluster-id* using a union-find technique. Mismatches are represented as a relation on cluster-ids and/or hosts. Before validating a pair of hosts we first map each host to a cluster-id if possible. If they map to the same cluster-id, or a mismatch can be detected the test is skipped. Another advantage of maintaining cluster information on-line is that if a host becomes unavailable temporarily during testing we can substitute another host from its cluster.

In evaluating precision and recall, failures pose a problem that is not encountered with static collections. To avoid making assumptions about the failed host pair that could bias the outcome, we chose to remove host pairs involving failures from our rankings.

4.1.3 Computing Document Resemblance We compute the *resemblance* of a pair of documents as a number between 0 and 1, such that when the resemblance is close to 1 it is likely that the documents are roughly the same. We view each document as a sequence of words, and lexically transform it into a canonical sequence of tokens. This canonical form ignores minor details such as formatting, HTML mark-up, and capitalization. We then associate with every document a set of continguous subsequences of tokens of length w, which we call *w-shingles*

For example the 4-shingling of

$$(\mathtt{a}, \mathtt{rose}, \mathtt{is}, \mathtt{a}, \mathtt{rose}, \mathtt{is}, \mathtt{a}, \mathtt{rose})$$

is the set

$$\{(\mathtt{a}, \mathtt{rose}, \mathtt{is}, \mathtt{a}), (\mathtt{rose}, \mathtt{is}, \mathtt{a}, \mathtt{rose}), (\mathtt{is}, \mathtt{a}, \mathtt{rose}, \mathtt{is})\}$$

Let $S(D)$ denote the w-shingling of document D for a certain shingle size. The resemblance r of two documents A and B is defined as

$$r(A, B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}.$$

To compute the resemblance of two documents it suffices to keep for each document a "sketch" of a few (three to eight) hundred bytes consisting of a collection of fingerprints (probabilistic unique id's) of shingles. The sketches can be computed reasonably fast (time linear in the size of the documents) and given two sketches the resemblance of the corresponding documents can be computed in time lineasr in the size of the sketches. Furthermore, clustering a collection of m documents into sets of closely resembling documents can be done in time proportional to $m \log m$ rather than m^2. For further details see [7, 5, 6].

4.2 Experimental Results

4.2.1 Precision vs Rank Figure 1 plots precision vs. rank up to 25000 for all our algorithms.

The naive *hosts* algorithm, which uses the least information performs the worst, with a terminal precision of 0.27 at rank 25000. The *hconn1* and *hconn2* algorithms, which compare out-edges in the host graph, have terminal precisions of 0.43 and 0.45 respectively. These algorithms can be misled into believing two hosts are mirrors if their pages point to a common set of hosts, which can happen with web sites that deal with the same or related topic.

The *paths* algorithm performs well on our input, with a terminal precision of 0.59 at 25000. However, the performance of *paths* depends on the presence of common paths which cannot always be relied upon with incomplete input sets. The *prefix* and *shingles* algorithms provide a big improvement over *paths* with terminal precision of 0.73 and 0.72 respectively. In both cases top-level directory paths, which occur in many URLs, are most likely to be selected as terms by virtue of their high term frequency. This makes these algorithms more effective and robust than *paths*. The IDF-based term weighting performed by these algorithms shows that the presence of a similar directory structure with similar names on two hosts is good evidence that the hosts are mirrors. The *conn* algorithm which filters the output of *paths* performs quite well, with a precision of 0.78 at 25000. By requiring link structure conformance of common paths *conn* is able to filter out many of the incorrect host pairs produced by *paths*. This indicates that a similar link structure among a selection of paths on two hosts is a good indicator of mirroring.

The *IP3* algorithm has a good initial yield with a precision of 0.92 at rank 7000. However, performance quickly degrades to a precision of 0.49 at 25000. This is because for larger 3 octet IP clusters, the assumption that all hosts on the same subnet are mirrors becomes untenable. Algorithm *IP4* performs best with a terminal precision of 0.89. However, since *IP4* requires an exact IP address match, and a good fraction of the mirrors we encounter do not meet this criteria, this algorithm is limited in its potential recall. Also, as in the case of *IP3*, as cluster size increases virtual hosting rather than mirroring becomes the dominant phenomenon. This is illustrated in Figure 2, which shows the cumulative number of hosts (solid line) and cumulative number of mirrors (dotted line) as the 4 octet IP cluster size is increased. To compute this we tested a sample of three hosts from every 4-octet IP cluster and, if they were mirrors, concluded that all hosts in the cluster are mirrors. Note that mirror count levels out almost completely after a cluster size of around 10, while the hosts count continues to rise as cluster size increases. This suggests that as larger and larger cluster sizes are reached, the precision of the IP4 algorithm will decrease substantially. (Figure 4 confirms this as we discuss below).

4.2.2 Relative Recall vs Rank Figure 3 plots relative recall at rank k vs. rank up to 25000. We define relative recall at a rank k for an algorithm as the number of mirrors found at rank k expressed as a fraction of the total number of distinct mirrors found in the union of all rankings up to rank k. Note that even an ideal algorithm may not achieve a relative recall of 1, since there may be more than k mirrors in the union of all top k rankings. For ranks up to 25000, the ordering of algorithms by relative recall performance is similar to that obtained when considering precision. The algorithms that rely on common paths and host connectivity are hurt by the fact that the input set is incomplete.

Surprisingly, even the best relative recall at 25000 (namely *IP4*) was only 53.9%. Thus, the task of finding mirrors cannot be entrusted to just a single algorithm. To find more mirrors it is possible to use the output of two or more algorithms.

Table 1 shows the fraction of the total pool of mirrors found by all algorithms to rank 25,000 present in the union of the top 25,000 results of a pair of algorithms. The total pool of mirrors consisted of 41,388 host pairs at rank 25,000. As the table shows, *IP4* singly finds only 53.9% of these mirrors. However, if we count the number of correct answers found by *IP4* or any one of the algorithms that consider the URL structure within hosts (*conn*, *paths*, *prefix* and *shingles*) up to rank 25,000, at least 74.5% of the 41,388 host pairs were found for each combination. The table also shows that there is not much to be gained by combining *shingles* and *prefix* together, or *hconn1* and *hconn2* together because they behave similarly and find nearly the same sets of mirrors.

To experiment further with the idea of merging the output of algorithms we combined the top 100,000 results from the output of five different algorithms *IP4*, *prefix*, *hconn2*, *paths* and *hostnames*. The host pairs in the resulting union were sorted in the decreasing order of the number of algorithms that found them. Ties between host pairs were resolved in favor of the host pair with the lowest rank in any of the five algorithms. This resulted in a ranking which we call *combined*.

Using the transitive closure of all mirrors found in validating the top 30,000 host pairs, we rated host pairs in the top 100,000 results of all algorithms. Host pairs which could not be rated were treated as mismatches. This gave us the precision-recall graph in Figure 4. As predicted previously *IP4*'s precision drops after a recall of 0.5 illustrating the limitations of using IP address matching to find mirrors. *Prefix* is the best of the individual algorithms, achieving a precision of 0.49 for a recall of 0.8. *Conn* seems potentially better but as only 30,000 host pairs were available this curve is incomplete. The combined algorithm *combined* had the best performance overall, with a precision at 0.57 for a recall of 0.86 on considering 100,000 host pairs.

5 Related Work

The problem of computing mirrored host pairs on the WWW was introduced in [2]. They use a more generalized definition of mirroring, which includes *partial mirroring* as well. They tested a single algorithm corresponding to a weighted

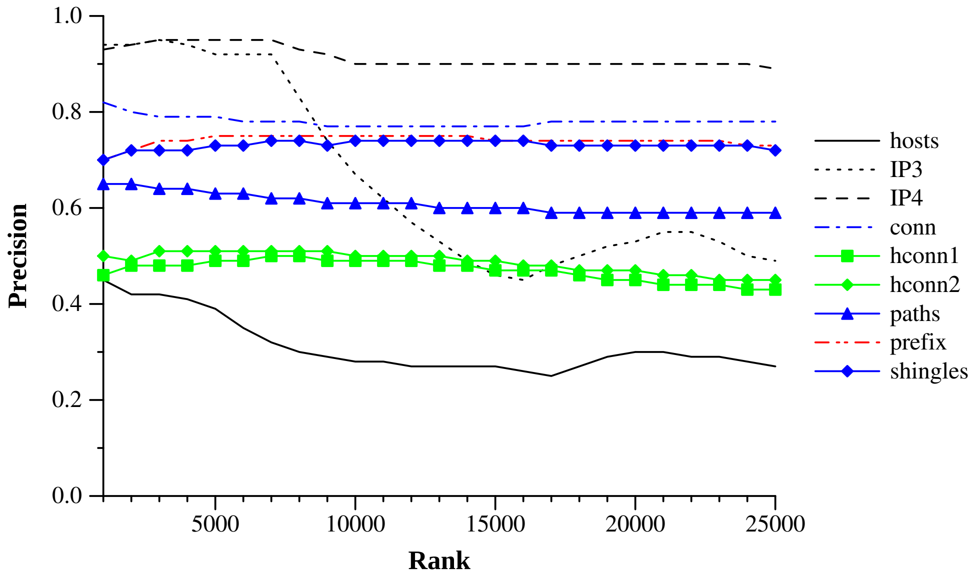

Figure 1: Precision at $rank$ for the algorithms

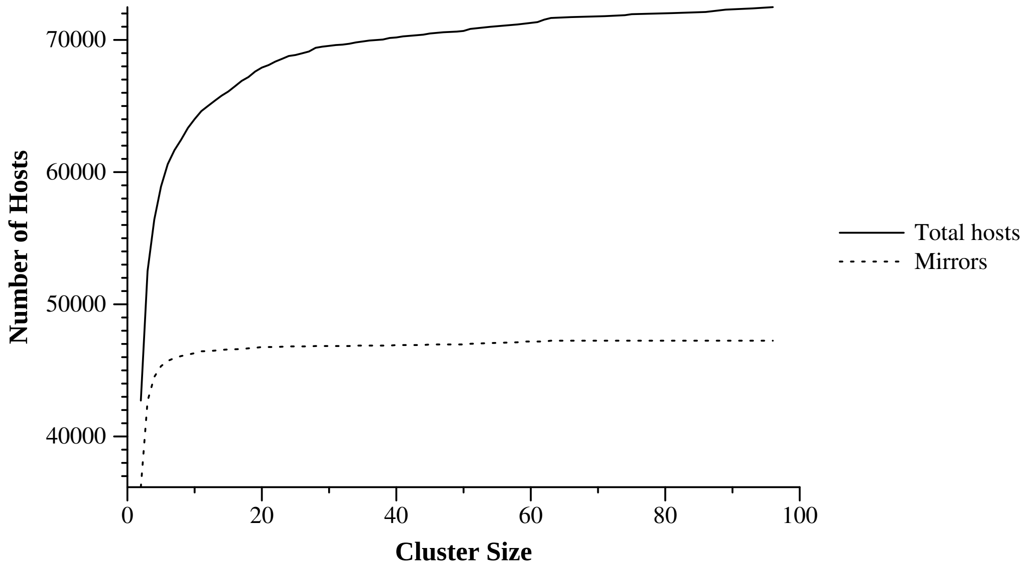

Figure 2: Cum. Count of Hosts and Mirrors vs. IP Cluster Size

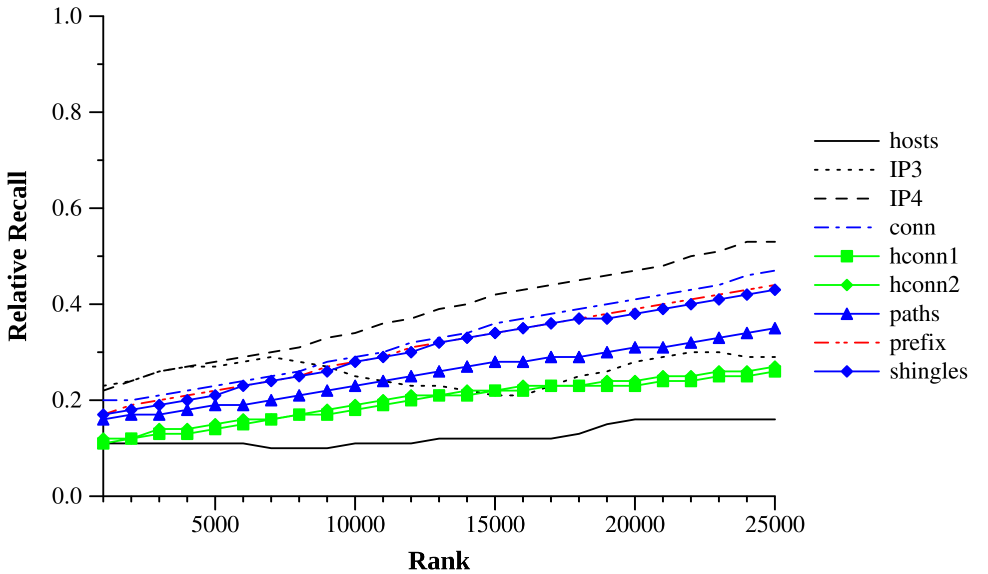

Figure 3: Relative Recall at $rank$ for the algorithms

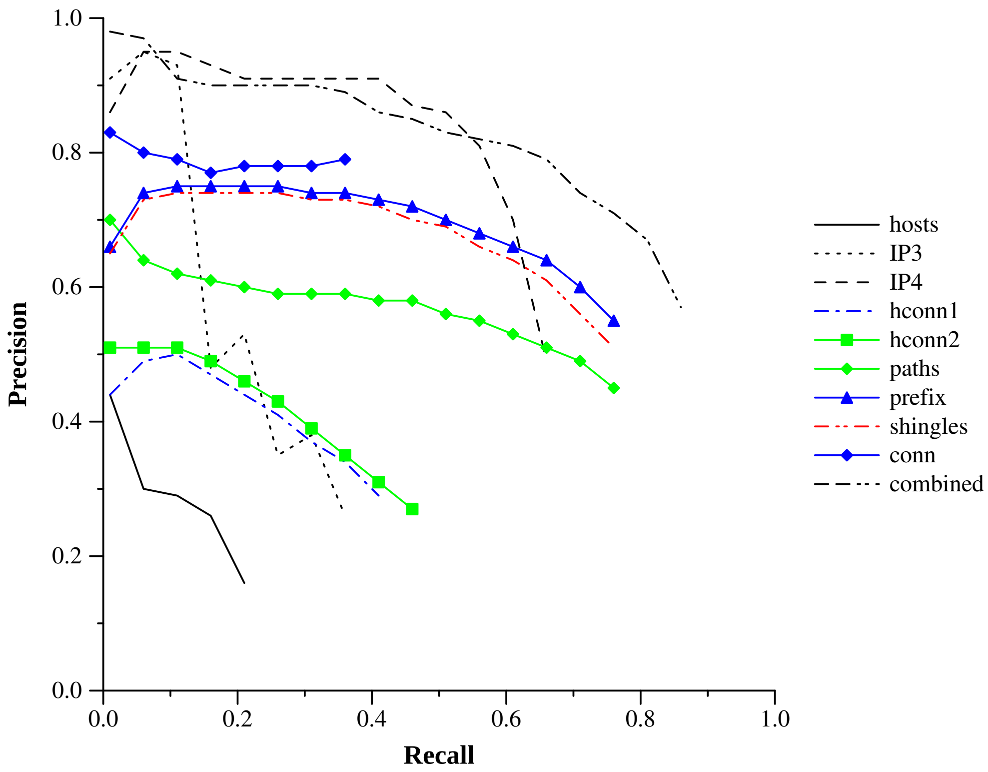

Figure 4: Precision vs Recall

	hosts	*IP3*	*IP4*	*conn*	*hconn1*	*hconn2*	*paths*	*prefix*	*shingles*
hosts	16.8%								
IP3	38.9%	29.7%							
IP4	60.6%	57.7%	53.9%						
conn	58.2%	65.4%	79.6%	47.3%					
hconn1	40.2%	51.1%	69.4%	59.3%	26.1%				
hconn2	41.2%	51.9%	69.8%	60.0%	29.0%	27.3%			
paths	48.4%	59.1%	77.8%	55.0%	50.7%	51.5%	35.8%		
prefix	53.7%	60.8%	74.5%	64.6%	57.5%	58.1%	57.1%	44.4%	
shingles	53.4%	60.6%	74.6%	64.2%	57.1%	57.7%	56.7%	48.4%	44.0%

Table 1: Percentage of correct answers found by combining the output of pairs of algorithms at rank 25000

combination of our *hosts* and *shingles* algorithms, but did not investigate the use of IP addresses or connectivity in finding mirrors. Since we were interested in a comparative study of algorithms we extended their validation scheme to include the pooling of validated results from various algorithms, and also the use of transitive inferences to expedite validation.

At first sight the problem of finding mirrors seems like it could be cast as a clustering problem in which each host is represented by the contents of the documents it contains. There has been considerable prior work in document clustering (see e.g. [18, 14]). As mentioned earlier, owing to the scale of the Web comparing the full contents of hosts is neither feasible, nor (as we believe) necessarily more useful than comparing URL strings and connectivity. See also [19] for a discussion on the applicability of clustering algorithms to the WWW.

Connectivity information has been applied to the problem of improving precision of WWW search results [13, 9, 8, 11, 4]. The graph model used by these methods is perturbed by duplicate pages. We believe our *hconn** and *conn* algorithms represent the first attempt to use connectivity information to finding mirrors. Furthermore, these two algorithms can be easily integrated within the methods cited above to alleviate the above perturbation problem.

6 Conclusion

Finding mirrors on the WWW is an information retrieval problem of interest to the search engine and web caching communities. In this paper we evaluated 4 classes of algorithms for ranking potential mirrored host pairs, namely: (i) IP address based approaches, (ii) URL string comparison techniques, (iii) host connectivity based approaches, and (iv) an approach that compares connectivity of documents with shared paths. These were evaluated on a collection of 140 million URLs (on 230,000 hosts) and associated connectivity information. *IP4* and *prefix* were our best single algorithms. We concluded that single approaches are limited in terms of recall. Our best approach is one which combines 5 algorithms and achieves a precision of 0.57 for a recall of 0.86 considering the top 100,000 results.

REFERENCES

1. *Proceedings of the Seventh International World Wide Web Conference*, Brisbane, Australia, April 1998.

2. K. Bharat and A. Z. Broder. Mirror, mirror on the web: A study of host pairs with replicated content. Personal communication. Submitted to WWW8.

3. K. Bharat and A. Z. Broder. A technique for measuring the relative size and overlap of public web search engines. In *Proceedings of the Seventh International World Wide Web Conference* [1], pages 379–388. See also URL: `http://www.research.digital.com/SRC/whatsnew/sem.html`.

4. K. Bharat and M. Henzinger. Improved algorithms for topic distillation in hyperlinked environments. In *Proceedings of the 21st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98)*, pages 111–104, 1998.

5. A. Z. Broder. On the resemblance and containment of documents. In *Proceedings of Compression and Complexity of Sequences 1997*, pages 21–29. IEEE Computer Society, 1988.

6. A. Z. Broder. Filtering near-duplicate documents. In *Proceedings of FUN 98*, 1998. To appear.

7. A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the Web. In *Proceedings of the Sixth International World Wide Web Conference*, pages 391–404, Santa Clara, California, April 1997.

8. S. Chakrabarti, B. Dom, D. Gibson, S. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Experiments in topic distillation. In *ACM–SIGIR'98 Post-Conference Workshop on Hypertext Information Retrieval for the Web*, 1998.

9. S. Chakrabarti, B. Dom, R. P., S. Rajagopalan, D. Gibson, and J. Kleinberg. Automatic resource compilation by analyzing hyperlink structure and associated text. In *Proceedings of the Seventh International World Wide Web Conference* [1], pages 65–74.

10. J. Cho, N. Shivakumar, and H. Garcia-Molina. Computing document clusters on the web. In *private communication (submitted to VLDB '99)*, 1999.

11. J. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 668–677, January 1998.

12. E. T. O'Neill, P. D. McClain, and B. F. Lavoie. A methodology for sampling the world wide web. Technical report, OCLC Annual Review of Research, 1997. URL: `http://www.oclc.org/oclc/research/publications/review97/oneill/o'neilla%r980213.htm`.

13. L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. Work in progress. URL: `http://google.stanford.edu/~backrub/pageranksub.ps`.

14. E. Rasmussen. Clustering algorithms. In W. Frakes and R. Baeza-Yates, editors, *Information Retrieval*, pages 419–42, 1992.

15. G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

16. N. Shivakumar and H. García-Molina. Finding near-replicas of documents on the web. In *Proceedings of Workshop on Web Databases (WebDB'98)*, March 1998.

17. E. Voorhees and D. Harman, editors. *Proceedings of the 6th Text Retrieval Conference(TREC-6)*, Gaithesburg-MD, Nov 1997.

18. P. Willet. Recent trends in hierarchical document clustering: a critical review. *Information Processing and Management*, 24(5):577–597, 1988.

19. O. Zamir and O. Etzioni. Web document clustering: A feasibility demonstration. In *Proceedings of the 21st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98)*, pages 46–53, 1998.