Paper Number 201

Finding replicated web collections

Junghoo Cho, Narayanan Shivakumar, Hector Garcia-Molina
Department of Computer Science
Stanford, CA 94305.
{*cho, shiva, hector*}*@db.stanford.edu*

Abstract

Many web documents (such as JAVA FAQs) are being replicated on the Internet. Often entire document collections (such as hyperlinked Linux manuals) are being replicated many times. In this paper, we make the case for identifying replicated documents and collections to improve web crawlers, archivers, and ranking functions used in search engines. The paper describes how to efficiently identify replicated documents and hyperlinked document collections. The challenge is to identify these replicas from an input data set of several tens of millions of web pages and several hundreds of gigabytes of textual data. We also present two real-life case studies where we used replication information to improve a crawler and a search engine. We report these results for a data set of 25 million web pages (about 150 gigabytes of HTML data) crawled from the web.

1 Introduction

Many documents are replicated across the web. For instance, a document containing JAVA Frequently Asked Questions (FAQs) is replicated at many sites. Furthermore, entire "collections" of hyperlinked pages are often copied across servers for fast local access, higher availability, or "marketing" of a site. We illustrate such replicated collections in Figure 1. The figure shows four actual sites that make available two collections: the JAVA 1.0.2 API documentation, and the Linux Documentation Project (LDP) manual. Other examples of commonly replicated collections include JAVA tutorials, Windows manuals, C++ manuals and even entire web sites such as ESPN's Sportszone and Yahoo. Even the Call For Papers for a database conference such as SIGMOD'00 is replicated in three web sites (`www.seas.smu.edu/sigmod2000/`, `www.dcc.unicamp.br/~mario/sigmod2000`, `http://www.dbai.tuwien.ac.at/pods`) across three continents. In this case, the collection is small and has a few hyperlinked documents such as the CFP, pages describing the conference committee, organizing committee and sponsorship information. In general, replicated collections constitute several hundreds or thousands of pages and are *mirrored*[1] in several tens or hundreds of web sites.

[1]The term mirror is often used to denote a replicated collection or web site; often a mirror site has the connotation of being a "secondary copy."

For instance, the LDP collection is a 25 megabyte collection of several thousand pages, mirrored in around 180 servers across the world.

Our goal in this paper is to study how to automatically identify mirrored "collections" on the web, so that a variety of tasks can be performed more effectively. These tasks include:

- **Crawling:** A crawler fetches web pages for use by search engines and other data mining applications. A crawler can finish its job faster if it skips replicated pages and entire collections that it has visited elsewhere.
- **Ranking:** A page that has many copies, in a sense, is more important than one that does not. Thus, search results may be ranked by replication factor. If a replicated page is a member of a collection, it may also be useful to indicate this in the search result, and perhaps to provide some main entry point to the collection (e.g., its root or home page).
- **Archiving:** An archive stores a subset of the web, for historical purposes [Ale99]. If the archive cannot store the entire web, it may give priority to collections that are known to be replicated, because of their importance and because they may represent coherent units of knowledge.
- **Caching:** A cache holds web pages that are frequently accessed by some organization. Again knowledge about collection replication can be used to save space. Furthermore, caching collections as a whole may help improve hit ratios (e.g., if a user is accessing a few pages in the LDP, chances are he will also access others in the collection).

As we will see in this paper, replication is widespread on the web, so the potential savings are very significant for each of the above tasks. For instance, consider the potential savings to the crawling task. A crawler that only visits one of the 180 LDP collections can avoid fetching about 4.5 gigabytes (25 MB * 180) of data. Indeed we will see in Section 5 that the crawler may save significant (over 40%!) bandwidth and time by avoiding major mirrored collections, such as LDP, JAVA API documents and Linux manuals. Recently, Bharat and Broder [BB99] observed similar results on data crawled by the AltaVista web crawler.

We have intuitively motivated the usefulness of detecting "replicated collections." However, there are important challenges in (1) defining the notion of a replicated collection precisely, (2) developing efficient algorithms that can identify such collections, and (3) effectively exploiting this knowledge of replication. One of the major difficulties in detecting replicated collections is that many replicas may not be strictly identical to each other. The reasons include:

1. **Update frequency:** The primary copy of a collection may often be constantly updated, while mirror copies are updated only daily, weekly, monthly, or by operator control. For

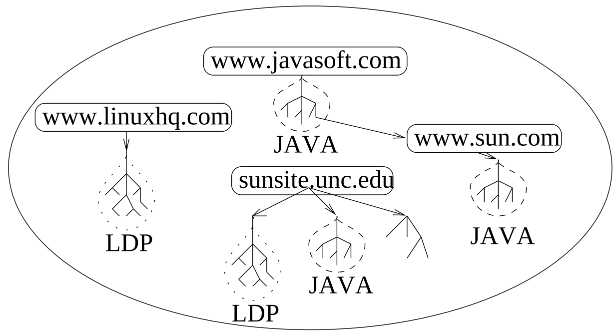


Figure 1: Mirrored document collections.

instance, the Javasoft corporation maintains the primary JAVA API manuals, and the University of Edinburgh hosts the LDP collection. These primary copies are regularly updated to incorporate the most recent bug fixes and documentation updates. However the mirrors of these collections are usually out of date, depending on how often they are updated.

2. **Mirror partial coverage:** Mirror collections differ in how much they overlap with the primary. In many cases, a collection is replicated entirely. In other cases, only a subset of a collection may be mirrored, and hyperlinks point to uncopied portions in another mirror site, or at the primary.

3. **Different formats:** The documents in a collection may *not* themselves appear as exact replicas in another collection. For instance, one collection may have documents in HTML while another collection may have them in PostScript, Adobe PDF or Microsoft Word. Similarly, the documents in one collection may have additional buttons, links and inlined images that make them slightly different from other versions of the document. Still, for some applications we may wish to consider two collections in different formats to be "similar."

4. **Partial crawls:** In most cases we need to identify replicated collections from a snapshot of the web that has been crawled or cached at one site, not by examining the original data. Thus, the data we are examining may be incomplete, e.g., because the crawler does not have the resources to fetch all data. So even if two collections are perfect replicas, their snapshots may not overlap fully. Furthermore, if the snapshots were crawled at different times, they may represent different versions, even if the originals were in perfect synchrony.

In such a scenario, it is challenging to even define what is a replicated collection, or even what is the "boundary" of a collection. For example, suppose that one site has a set S_1 of 10 pages, a second site has a set S_2 that is an exact copy of S_1, and a third site has a set S_3 containing only 5 of the S_1 pages. In this case we have at least two choices: we could say that S_1 is a replica of S_2 (ignoring S_3), or we could say that there are two additional replicas of the S_3 collection, each with

5 replicated pages. That is, in the former case we have a 10-page collection at two sites, and in the latter case we have a 5-page collection at 3 sites. If we wish to consider page and link structure similarity, we have even more choices. For instance, suppose that only 8 of the 10 pages in S_2 are identical copies of their S_1 counterparts. The remaining 2 are variants, e.g., they may have slightly different content, or may have missing or extra links. Do we still consider S_1 and S_2 to be replicas, or do we only consider corresponding sub-collections with 8 identical pages to be replicas? In the former case we have a larger collection, but in the latter case we have a more faithful replica. In this paper we study these issues, and propose a mechanism for making such decisions.

The amount of data on the web is staggering, on the order of hundreds of millions of pages and hundreds of gigabytes, and growing rapidly. Thus, whatever techniques we use for identifying replicated pages and collections must scale to very large sizes. Thus, we develop new techniques for analyzing and manipulating our data.

After replicated collections have been identified, a final challenge is how to exploit effectively this information to improve crawling, ranking, archiving, caching and other applications. For instance, should a crawler try to identify replicas as it performs its first crawl, or should it first do a full crawl, then identify replicas, and skip mirrors on subsequent crawls? If replicas are skipped, should the crawler still visit them infrequently to make sure they are still copies? In this paper while we do *not* focus on this category of challenges, we briefly touch on some of the crawling and ranking issues in Section 5, where we present our experimental results.

In summary, the contributions of this paper are the following:

- **Computable similarity measures:** We define a practical similarity measure for collections of web pages, and study options for defining boundaries of replicated collections. We carefully design these measures so that in addition to being "good" measures, we can efficiently compute them over hundreds of gigabytes of data on disk. Our work is in contrast to recent work in the Information Retrieval domain [PP97] where the emphasis is on accurately comparing link structures of document collections, when the document collections are small. We then develop efficient heuristic algorithms to identify replicated sets of pages and collections.

- **Improved crawling:** We discuss how we use replication information to improve web crawling by avoiding redundant crawling in the Google system.[2] We report on how our algorithms and similarity measures performed on over 150 gigabytes of textual data crawled from the web (about 25 million web pages).

- **Reduce clutter from search engines:** Finally, we discuss how we used replication information for another task, that of improving how web search engines present search results. We have built an alternative search interface to the Google web search engine. This prototype shows significantly less clutter than in current web search engines, by clustering together

[2] Google [BP99] is a search engine developed in our group; it is currently commercially available at `www.google.com`.

replicated pages and collections. Based on informal user testing, we have observed this simple new interface helps the user quickly focus on the key results for a search over web data.

In Section 2 we discuss our similarity measures for pages and collections. Then in Section 3 we present our algorithm for efficiently identifying groups of similar collections. In Section 4 we informally evaluate the quality of our similarity measure, while in Section 5 we report on our experience using our techniques for improving web crawling and result displaying.

2 Similarity of collections

Our goal is to identify collections of web pages that are "approximate replicas" or "similar." As a first step, it is instructive to define the notion of "identical collections." By relaxing that definition, we will then be able to obtain a useful notion of approximate collections.

The following definitions set the stage for our definition of identical collections. Note that when we refer to a web page we refer to its textual (or image) content. The hyperlinks are modeled separately, as arcs between pages.

Definition 1 Web graph Given a set of web pages, the web graph $G = (V, E)$ has a node v_i for each web page p_i, and a directed edge from v_i to v_j if there is a hyperlink from page p_i to p_j. □

Definition 2 Collection A collection is an induced subgraph of the web graph G. (Recall that an induced subgraph $G' = (V', E')$ of a graph $G = (V, E)$, only has edges between the vertices in the subgraph.) The number of pages in the collection is the *collection size*. Collection C' is a *subcollection* of C, if it is an induced subgraph of C. □

Notice that our graph model does not capture the position or anchor text of web hyperlinks within their pages. For example, our model does not distinguish between two pages with identical text and identical hyperlinks that are located in different places on the page. This is an acceptable simplification since our ultimate goal is to find similar clusters.

Also notice that we have not restricted a collection to reside at a single web site or to be connected. In practice, one may be only interested in collections at a single site (e.g., when improving crawling), but this single-site condition can be enforced later as (or after) collections are identified. Similarly, a collection whose pages are not connected via hyperlinks may be undesirable because it represents a set of unrelated pages. Again, this additional restriction can be enforced as collections are identified.

Definition 3 Identical[3] collections Equi-sized collections C_1 and C_2 are identical ($C_1 \equiv C_2$) if there is a one-to-one mapping M that maps all C_1 pages to C_2 pages (and vice-versa) such that:

[3] These are sometimes referred to as *label preserving isomorphs* in graph terminology. We use the term *identical* to be consistent with other definitions we introduce.

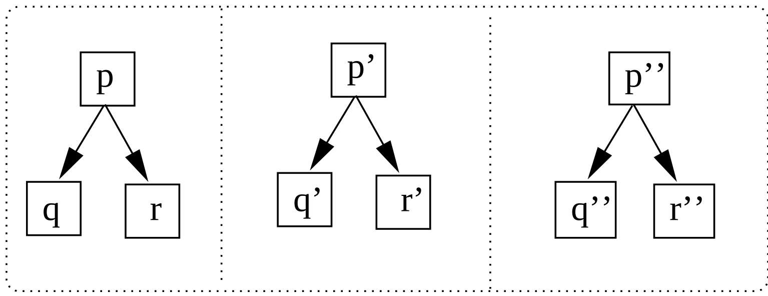


Figure 2: Collections with similar pages.

- **Identical pages:** For each page $p \in C_1$, $p \equiv M(p)$. That is, the corresponding pages have identical content.
- **Identical link structure:** For each link in C_1 from page a to b, we have a link from $M(a)$ to $M(b)$ in C_2. □

As discussed in the Introduction, many replicated collections of interest are not strictly identical. Thus, we need to relax Definition 3 so that "similar" collections can be identified. Since there are many different ways to relax this definition, we need to keep in mind two important objectives: (a) We want a notion of similarity that tracks our intuitive notions. That is, the collections we identify as similar should appear to a human to be "close copies" of each other. (b) It should be possible to efficiently and automatically check for similarity, so it is feasible to search for similar collections over huge number of web pages. In the rest of this section we discuss the relaxations we believe meet these criteria and that lead to our definition of similarity. In Sections 3 and 4 we use our definition to identify similar clusters on the web and to validate our reasoning.

2.1 Similarity of web pages

Definition 3 requires that matching pages in identical collections be identical. One first step in relaxing this definition is to allow the matching pages to be approximate replicas. For example, in Figure 2 we show three collections. Solid, small boxes represent pages, arrows between pages are the hyperlinks, and the dotted boxes encompass collections. We have labeled the top page in each collection as p, p' and p'' to suggest that their contents are similar. We have labeled the other pages analogously. Given that the link structure across these collections is identical, it seems natural to say that this cluster of three collections is similar.

One can determine the similarity of two pages in a variety of ways. For instance, one can use the information retrieval notion of textual similarity [Sal83]. One could also use data mining techniques to cluster pages into groups that share meaningful terms (e.g., [PE98]), and then define pairs of pages within a cluster to be similar. A third option is to compute textual overlap by counting the number of *chunks* of text (e.g., sentences or paragraphs) that pages share in common [SGM95, SGM96, BGM97, BB99]. In all schemes, there is a threshold parameter that indicates how close pages have to be (e.g., in terms of number of shared words, n-dimensional distance, number of overlapping chunks) to be considered similar. This parameter needs to be empirically adjusted according to the target application.

We expand on the textual overlap option since it is the one we use in our experiments. To compute text overlap, we first convert each page from its native format (e,g., HTML, PostScript) into simple textual information using standard converters.[4] The resulting page is then chunked into smaller textual units (e.g., lines or sentences). Each textual chunk is then hashed down to a 32-bit *fingerprint*. If two pages share more than some threshold T of chunks with identical fingerprints, then the pages are said to be similar.

In prior work [SGM95, SGM96], we have shown this definition to be effective in approximating the human notion of pages that share significant content. Broder et al. [BGM97, Bro97, BB99] also report similar results. Furthermore, these references also show that checking for similarity has a low cost, so it is feasible to compute page similarity over very large collections. Hence, this is the definition of similarity we use in our experiments. However, it is important to note that any definition of page similarity could be used in our forthcoming definition of similar collections, and that any definition could be used in our algorithm for identifying similar clusters (although performance would suffer if a more expensive definition were used).

Given that we have a function to determine similarity of two pages, we still need to discuss how it is used to compare collections. Returning to our example, suppose that p is similar to p', that p' is similar to p'', but that p and p'' are *not* similar. Do we still wish to consider the three collections shown in Figure 2 similar, or do we wish to consider them pair-wise similar (e.g., the leftmost collection is only similar to the center one)? Our decision here is to consider all three collections similar (transitive similarity), mainly because it significantly reduces the cost of computing similarity. If we wished to compute pair-wise similarities, we would need to consider all possible pairs of all possible collections (in our example we would have to try to match the center collection against the left one and the right one). With transitive similarities, we only need to figure out how to partition our pages into similar collections. (This tradeoff will become more apparent as we describe the cluster identification algorithm.)

In our experience, transitive similarity still identifies groups of pages that are closely related. For example, p' may have been obtained form p (e.g., by extracting the plain text from a postscript file), and then p'' may have been obtained from p' (e.g., by appending some comments to the file).

[4]Converters such as `ps2ascii` and `html2ascii` are not always exact converters and cannot handle non-ASCII objects such as figures and equations in the text. But for our application, we believe this problem is not significant.

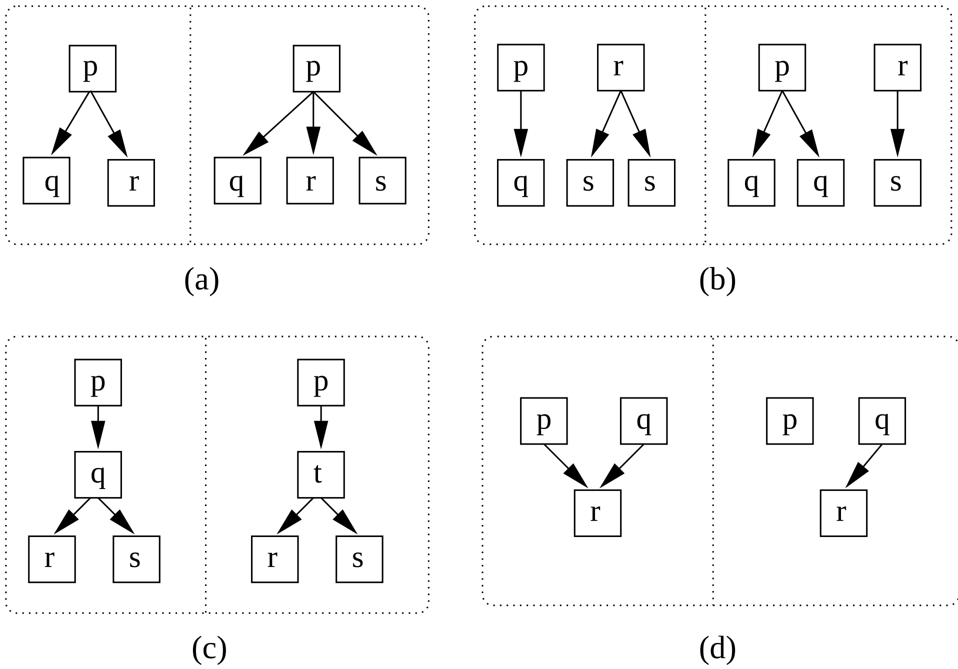


Figure 3: Example pairs of collections.

Even though the similarity test fails for p and p'', in most cases a human would still consider all three pages to be approximate copies.

The following definition formalizes the notion of transitive similarity that will be used in our definition of cluster similarity.

Definition 4 Page similarity We are given a test, $ST(p_i, p_j)$, for determining if a pair of pages p_i and p_j are pair-wise similar. Based on this test, we say that two pages are (transitively) similar, $p_i \approx p_j$, if there is some sequence of pages $p_1, p_2, \ldots, p_k$ in our web graph, such that $ST(p_i, p_1)$, $ST(p_1, p_2), \ldots$, $ST(p_k, p_j)$. □

2.2 Similarity of link structure

We also wish to consider collections as similar even if their link structure does not match exactly. The following four examples illustrate the decision that we must make in defining precisely what it means for link structures to match approximately. The examples are shown in Figures 3 (a) – (d). In each example, we refer to the first collection as $[i]$ and the second collection as $[ii]$. If two pages have the same label, then they are similar. For example, $[i].p \approx [ii].p$.

- **Collection sizes:** In Figure 3(a), we show two collections. They are identical except for the additional page $[ii].s$ and the link from $[ii].p$ to $[ii].s$. It seems natural to say that these two collections are similar, since the differences are "relatively small." That is, we can find a function M, analogous to the one in Definition 3, that maps "almost all" pages in one collection to similar pages in the other collection. However, we will require similar collections to be equi-sized for two reasons. The first is that identifying collections that are similar (and the corresponding mapping functions) is much simpler if we are only looking for collections of equal size. Trying to locate collections of different sizes means we need to compare many more collections. The second reason is that, if desired, one can add a second phase to the search process to identify collections of different sizes. In our example, we would first identify the core similar collections (with pages p, q and r). Then the second phase would search for pages (like s) that could be added to one or the other collection, while still maintaining some notion of similarity. In this paper we only cover the first phase where equi-sized collections are identified.

- **One-to-one:** In Figure 3(b) we show two equi-sized collections that, again, could be considered similar by some. Notice that $[ii].p$ points to two copies of $[ii].q$, while $[i].p$ points to one copy of $[i].q$. Similarly, $[i].r$ and $[ii].r$ point to two copies of $[i].s$ and one copy of $[ii].s$ respectively. In this case, we can easily compute a mapping M from all pages in one collection to some similar page in the other collection. However, this mapping is not one-to-one. In this paper we will restrict our search to one-to-one mappings in order to constrain complexity. Collections like the ones in Figure 3(b) will be deemed dissimilar, but we believe that this is tolerable in practice. The collections that might be missed because they cannot be matched by one-to-one mappings are not common because they contain replicas within the collection, which is unusual.

- **Break points:** In Figure 3(c) we see that the two link structures look similar, with the exception that pages $[i].q$ and $[ii].t$ are different. We could define the two collections to be similar since most of the pages in one collection can be mapped to a similar page in the other collection. For instance, we can compute a mapping M that maps $[i].p$ to $[ii].p$, $[i].r$ to $[ii].r$ and $[i].s$ to $[ii].s$ (and vice versa). This only omits $[i].q$ and $[ii].t$ from the mapping since they have different content. Thus, when defining collection similarity, we could require that only a "large fraction" of the pages be mapped (by a one-to-one mapping) to similar pages. However, to control complexity, we will again be more conservative and require that *all* pages be mapped to similar pages. The price we pay is that certain collections, like the one in Figure 3(c), will not be identified as similar. Instead, the dissimilar pages (like q and t) will break the collection into smaller collections. In our very simple example, pages $[i].p$ and $[ii].p$ would be identified as similar collections. The r and the s pages would form other collections. We call pages like q and r *break points* because they split similar collections. If break points are a problem, we could introduce a second phase that tries to combine similar collections that are connected via "short" segments. We do not discuss such a second phase further in this paper. In Section 5 we will informally evaluate the occurrance of break points.

- **Link similarity:** Figure 3(d) highlights another issue. We could say the two collections are similar since each similar page shares similar incoming links. For instance, $[ii].r$ has an incoming link from $[ii].q$, and the page that matches with $[ii].r$, i.e., $M([ii].r) = [i].r$, has one from the corresponding page $M([ii].q) = [i].q$. In the reverse direction, $[i].r$ has two incoming links, and at least of one them ($[i].q$ to $[i].r$) has a matching link. Note that the incoming links to the p and q pages match across the collections (no incoming links in either case).

  For our similarity test, we will require that matching pages have *at least one* matching incoming link, unless neither page has any incoming links. Having at least one link ensures that if one collection is a tree, then any similar collection must have the same tree structure. If one collection is a DAG with a single root page from which all pages can be reached, then the property ensures that in a similar collection all pages will be reachable from the node that matches the root.

  One could use a stronger definition requiring more matching incoming links. For instance, if $[i].r$ had 10 incoming links, we could require $[ii].r$ to have not just one analogous link, but say 8 (80%) analogous links. We do not use here such conservative definitions because they lead to more expensive similarity tests, and because most collections of interest have root pages or tree-like structures, and the at-least-one-link test is quite good at identifying what humans consider similar collections. Also note that we could phrase our test in terms of outgoing (as opposed to incoming) links, but we believe that this gives a test of equivalent usefulness.

We have described step by step the considerations that lead us to the following definition for similar collections. As we have stated, there are many different ways to define similarity, and out contribution here is to present one that is logically sound, that leads to efficient algorithms, and that identifies many of the collections that humans would call similar. After our definitions, we will present an algorithm for efficiently identifying similar collections, and in Section 5 we will present experimental results to substantiate our claim that our notions of similarity are useful and natural for the web.

Definition 5 Similar collections Equisized collections C_1 and C_2 are similar (i.e, $C_1 \cong C_2$) if there is a one-to-one mapping M (and vice-versa) that maps all C_1 pages to all C_2 pages such that:

- **Similar pages:** Each page $p \in C_1$ has a matching page $M(p) \in C_2$, such that $p \approx M(p)$.

- **Similar links:** For each page p in C_1, let $P_1(p)$ be the set of pages in C_1 that have a link to page p. Similarly define $P_2(M(p))$ for pages in C_2. Then we have pages $p_1 \in P_1(p)$ and $p_2 \in P_2(M(p))$ such that $p_1 \approx p_2$ (unless both $P_1(p)$ and $P_2(M(p))$ are empty). That is, two corresponding pages should have at least one parent (in their corresponding collections) that are also similar pages. □

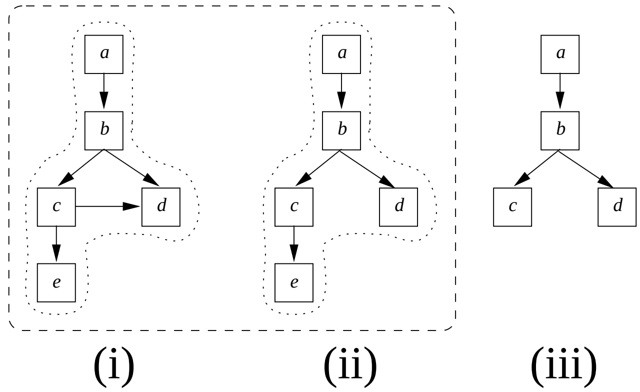


Figure 4: One possible similar cluster.

We will call a group of collections a *cluster*, and if the collections in a cluster are similar, then we will call it a *similar cluster*. The following definitions formalize these notions.

Definition 6 Cluster We define a set of equi-sized collections to be a *cluster*. The number of collections in the cluster is the *cluster cardinality*. If s is the collection size of each collection in the cluster, the *cluster size* is s as well. □

Definition 7 Identical cluster Cluster $R = \{C_1, C_2, \ldots, C_n\}$ is an *identical cluster* if for $\forall i, j$, $C_i \equiv C_j$. That is, all its collections are identical. □

Definition 8 Similar cluster A cluster $R = \{C_1, C_2, \ldots, C_n\}$ is similar if all of its collections are pairwise similar, i.e., if for $\forall i, j$, $C_i \cong C_j$. □

3 Computing similar clusters

Our goal is now to identify similar clusters within a given web graph. To illustrate the challenges, consider the graph shown in Figure 4. In this figure we have identified one similar cluster (dashed line), consisting of two collections (dotted lines). (Again, pages with the same label are similar.) Here the cluster cardinality is 2, while the collection size is 5. Figure 5 shows a different cluster in exactly the same web graph. This cluster has cardinality 3 and its collections have size 3. Our examples show that different types of similar clusters can be found in a web graph, and that there are complex interactions between the number of clusters found, their cardinalities, and the sizes of their collections.

In general, we could postulate an objective function to optimize in our cluster search process. This function would give a value to a particular set of clusters, based on the number of clusters, their cardinalities, and the sizes of the collections. Unfortunately, searching for the optimal clustering

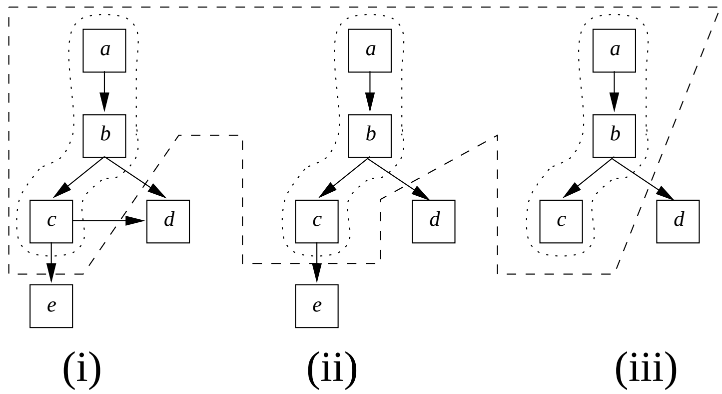


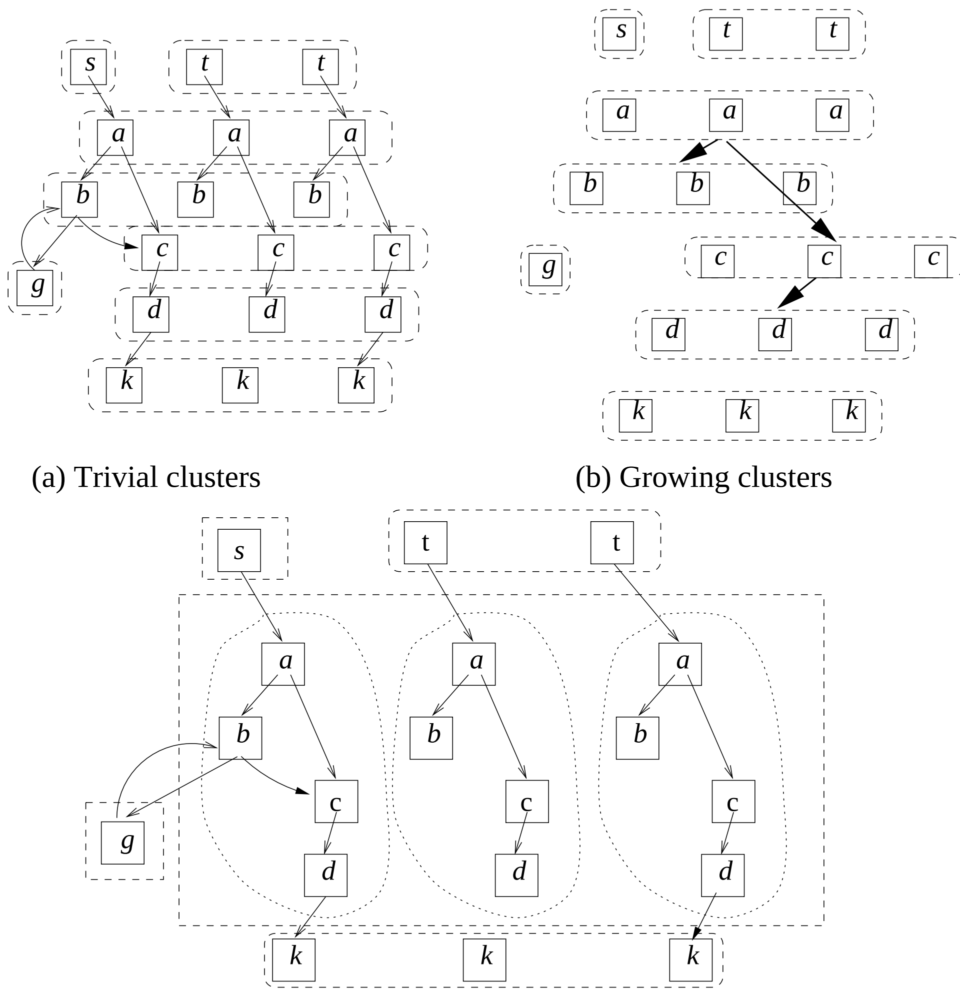
Figure 5: Another possible similar cluster.

would be extremely expensive, since in principle one would have to consider all possible ways to form collections and clusters.

Instead of such a generalized search for the best clustering, we propose here a clustering algorithm that "grows" clusters from smaller sized ones. As we will see, the algorithm favors larger cardinalities over larger collections.

Figure 6 illustrates how the algorithm works. The algorithm first computes *trivial clusters* on the given web graph. These clusters have collections of size one and are found by grouping similar pages. For example, the two pages labeled t are determined to be similar, and form one of the trivial clusters. (This cluster has two collections, each with a single page. We do not show the dotted lines that form the collections to avoid cluttering the figure.) Figure 6(a) shows the trivial clusters found in this example web graph.

Next, the algorithm merges trivial clusters that can lead to similar clusters with larger collections. We detail this process below (Figure 6(b)). The outcome is shown in Figure 6(c), where trivial clusters a, b, c and d have been merged into a larger-size cluster. (Again, clusters are shown with dashed lines, collections with dotted lines. Collections with a single page are not shown to avoid clutter.) The remaining trivial clusters are not merged. For example, the t cluster is not merged with the larger cluster because it would lead the collections to be of different sizes and hence not similar by our definition. Notice that the s and g clusters could form a larger cluster (with a single collection of size 2). However, our algorithm does *not* generate disconnected collections (see Section 2), so s and g are left as separate clusters.



(a) Trivial clusters

(b) Growing clusters

(c) Maximal cardinality clusters

Figure 6: Growing similar clusters.

3.1 Growth strategy

We now discuss how to merge clusters. Consider two trivial similar clusters $R_i = \{p_1, p_2, \dots, p_n\}$ and $R_j = \{q_1, q_2, \dots, q_m\}$, where p_k and q_k $(1 \le k \le n)$ are pages in R_i and R_j. We define:

1. $s_{i,j}$ to be the number of pages in R_i with links *to* at least one page in R_j,
2. $d_{i,j}$ to be the number of pages in R_j with links *from* at least one page in R_i,
3. $|R_i|$ to be the number of pages in R_i, and
4. $|R_j|$ to be the number of pages in R_j.

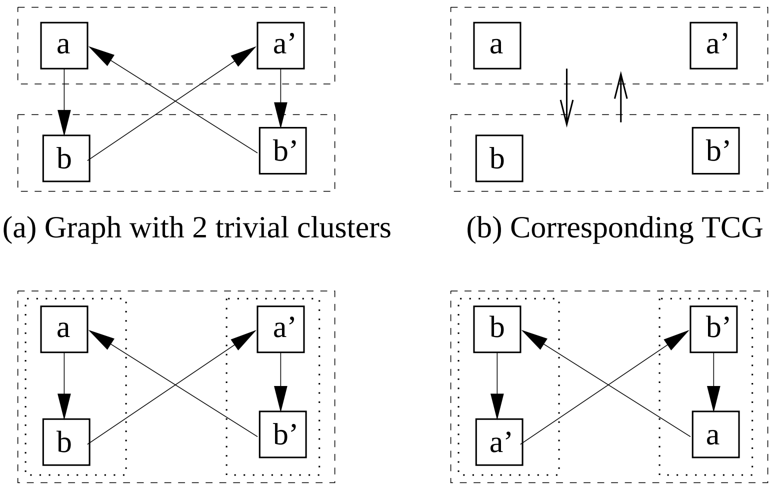
We say that two trivial clusters R_i and R_j can be joined if they satisfy the *merge condition* $|R_i| = s_{i,j} = d_{i,j} = |R_j|$. This condition implies that each page in R_i has a hyperlink to a unique page in R_j. For example, consider trivial clusters a and b in Figure 6(a). They each have three collections, and each of the three b pages has an incoming link from one of the a pages. Thus, these two trivial clusters satisfy the merge condition, and can form a larger cluster of three collections, where each collection has one of the a pages and the b page it points to. Such a larger cluster clearly meets the conditions of Definition 5.

Continuing with the example, notice that trivial cluster c can be joined with the $\{a, b\}$ cluster we have just formed. This is because the merge condition holds between c and one of the trivial clusters (b) that formed the $\{a, b\}$ cluster. The new cluster can be formed by adding each c page to the collection containing the b page that points to that c page. Similarly, we can merge the d cluster, to obtain the cluster shown in Figure 6(c).

We can see that the resulting cluster (with collections of size 4) satisfies Definition 8, that is, each collection is similar to the others in the cluster. For instance, consider page c in the leftmost collection of Figure 6(c). It has matching pages in the other two collections of the cluster, and at least one of the incoming arcs (the one from a) is found in the other collections. This guaranteed arc is the one that was used to grow trivial cluster c into this larger cluster. Similarly, all pages in the leftmost collection have matching pages in the other collections.

Our clustering algorithm proceeds in this fashion. Conceptually, it consists of the following steps:

1. Find all trivial clusters.
2. Form a trivial cluster graph (TCG) $G = (V, E)$, where each vertex v_i in V corresponds to one of the trivial clusters, and we have an edge from v_i to v_j if v_i, v_j satisfy the merge condition. The TCG for our running example is shown in Figure 6(b).



(a) Graph with 2 trivial clusters (b) Corresponding TCG

(c) One possible cluster (d) Another cluster

Figure 7: Impact of start point for merge.

3. Merge each connected component of the TCG into a similar cluster. The merge process for a connected component can start at any node from which all the others in the component can be reached.

In some cases there can be more than one place to start the merge process for a connected component. Figure 7 shows one example with two trivial clusters a and b. The TCG has a cycle between these two nodes, so we can start the process at either node. If we start at a, we obtain the cluster shown in Figure 7(c), while if we start with b we obtain the cluster in Figure 7(d). In practice, one of these clusters may be better than the other because the collections have a more natural entry point or root. One possible way to identify the more natural starting point is to select the page with the largest number of incoming links *from other web sites*. Thus, if more sites have links to a (either copy) than to b, then a may be a more natural starting point, and the cluster in Figure 7(c) would be preferable.

It can be shown that our clustering algorithm yields similar clusters, using the arguments outlined in our example. As illustrated by our example, our algorithm only produces clusters with connected collections (e.g., s and g were not merged in Figure 6). We believe this yields more natural results. However, the merge condition can easily be extended to merge collections that have no links between them. Finally, the algorithm gives preference to cardinality over collection size. That is, the algorithm will identify the largest-cardinality similar cluster that may be formed (with connected collections), even if its collections have only one page. From the remaining pages, it will identify the next largest-cardinality cluster, and so on.

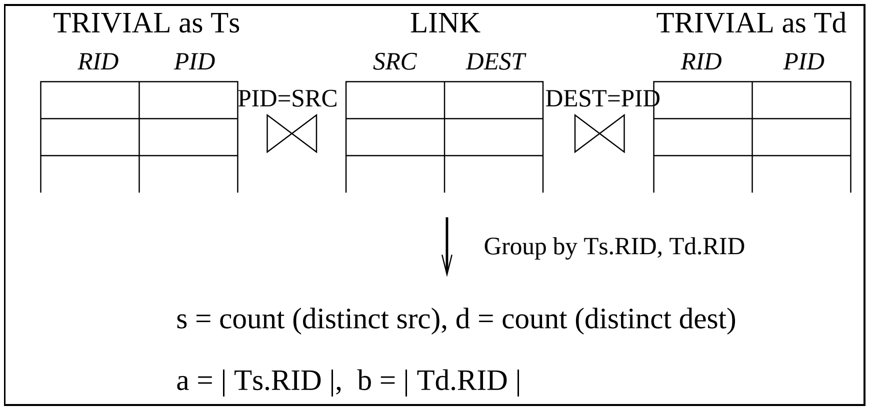


Figure 8: Join-based construction of *LinkSummary* table

3.2 Implementing the cluster growing algorithm

The clustering algorithm we have described can be implemented very efficiently by relying on good database algorithms. Initially we compute all similar page pairs using an iceberg algorithm [FSGM+98]. These algorithms efficiently find tuples in a table that occur more than some threshold number of times. In our case, the table contains each instance of a shared chunk between two pages, and the threshold is the number of common chunks needed between similar pages (Section 2). The output of the iceberg algorithm is a second table containing the pairs of pages that are directly similar (function ST of Definition 4). We then perform a simple disk-based UNION-FIND [CLR91] algorithm that performs sequential scans over this table and computes trivial similar clusters. The output of this phase is a table TRIVIAL(rid, pid), which contains tuple $\langle R_i, p_j \rangle$ to indicate that page p_j is in trivial cluster R_i. This table materializes all trivial similar clusters as per Definition 8.

Figure 9 shows how to implement the cluster growing algorithm efficiently. It expects as input table TRIVIAL(rid, pid) and a second table, LINK$(src, dest)$, giving the links in the web graph. (Tuple $\langle p_i, p_j \rangle$ is in LINK if p_i has a hyperlink to p_j.) In Step [1] we pre-process the data, using relational operators, to compute link statistics ($s_{i,j}, d_{i,j}$, $|R_i|$, and $|R_j|$) for every pair of trivial clusters, R_i and R_j. Well known relational query optimization techniques can be used to execute this step efficiently. In Steps [2] – [4] we compute all pairs of trivial clusters that satisfy the merge condition. Finally, we compute our maximal-cardinality similar clusters in Step [5] using the classical UNION-FIND algorithm [CLR91]. In this step, we are conceptually regaining the collection structure (i.e., Figure 6(c)) by merging collections based on computing connected components.

Algorithm 3.1 ***Cluster growing algorithm***
Inp : Tables *Trivial* and *Link*,
Out : The set of similar clusters
Procedure

[0] $\mathbf{S} \leftarrow \{\}$ `// S: the set of similar clusters`
[1] Construct *LinkSummary* table with schema
$\langle\ Ts.RID,\ Td.RID,\ a,\ b,\ s,\ d\ \rangle$ based on Figure 8
[2] For each entry in *LinkSummary*
[3] If $(a = s = d = b)$
[4] $\mathbf{S} \leftarrow \mathbf{S} \cup \{\langle Ts.RID,\ Td.RID\rangle\}$ `// Coalesce` $Ts.RID$ `and` $Td.RID$
[5] Return "UNION-FIND($\mathbf{S}$)" `// Find connected components`

Figure 9: Cluster growing algorithm.

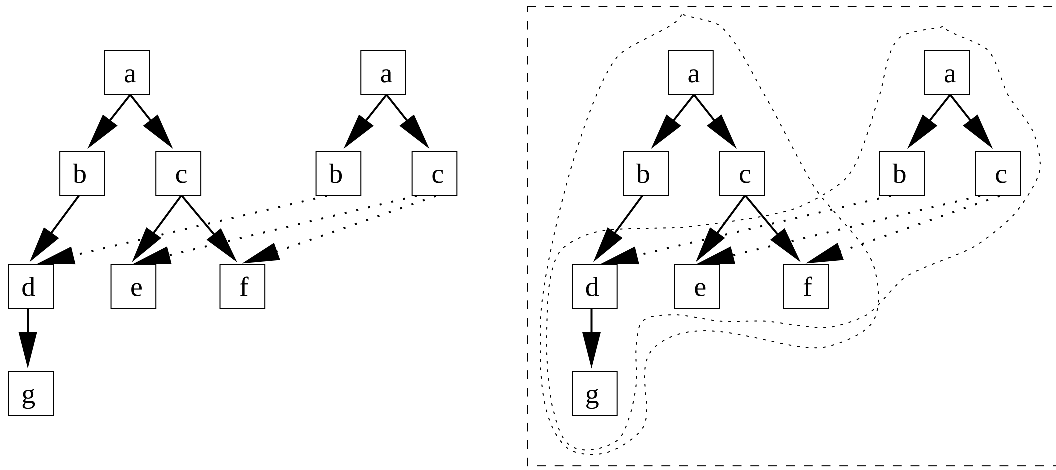
4 Quality of similarity measures

In defining our notion of similar clusters we made a number of choices, to arrive at an efficient algorithm for identifying clusters. In this section we describe experiments performed to *informally* evaluate the "quality" of the clusters found by our algorithm. We stress that since there is no single way to identify similar clusters, even if performance were not an issue, there is no definitive quality measure. Instead, we simply provide some examples and statistics to describe the characteristics of the clusters identified by our algorithm and by our definition of similarity.

We chose 25 widely replicated web collections (e.g., Perl, XML, JAVA manuals) from their primary sites (e.g., www.perl.org). We call these collections the *targets* to distinguish them from the potentially different collections that may be identified by our algorithm. The sizes of the targets varied between 50 and 1000 pages. For each target we automatically downloaded between five and ten mirrored versions from the web. The mirrored targets were different versions of the primary version, either older or slightly modified versions (e.g., additional links or inlined images). The total number of web pages so downloaded was approximately $35,000$. In addition, we added $15,000$ randomly crawled pages to this data set. We assume these pages were unrelated to each other and the targets we had already downloaded.

On this data set we computed similar clusters using our cluster growing algorithm. We then manually examined these clusters to see if they corresponded to the mirrored targets. Our algorithm identified 180 non-trivial collections. Of these, 149 collections formed 25 clusters, corresponding to our 25 targets. Each of these clusters had at least 2 collections.

The remaining 31 collections did not correspond to a target, and hence we call them "problem"



(a) Original graph with partial mirror

(b) Cluster found with extended merge condition

Figure 10: Example of partial mirrors.

collections. Upon examining the problem collections, we discovered that many were caused by what we call *partial mirrors*, as illustrated by Figure 10(a). The figure (left side) shows a collection [ii] that is a partial mirror of the [i] collection. That is, some of the [ii] pages point to the unmirrored portions of the first collection. In this case, our cluster growing algorithm would only identify a cluster with the collections $[i].\{a, b, c\}$ and $[ii].\{a, b, c\}$, leaving the other pages as "problem collections."

In some cases (e.g., for improving crawling), it is more useful if similar collections include the partially mirrored pages. In our example, it may be better to identify the two collections shown in Figure 10(b), where pages d, e, f and g appear in both collections.

Our cluster growing algorithm can be easily extended to identify partially mirrored collections, if so desired. This can be achieved by modifying the merge condition used by the algorithm: We now merge trivial clusters R_i and R_j whenever $|R_i| = s_{i,j} \geq d_{i,j} = |R_j|$. This weaker condition arises when pages in R_j have "virtual" links from pages in R_i, i.e., the R_j pages are not mirrored in as many sites as the R_i pages. We call the clusters produced by our modified merging condition as *extended clusters*.

Returning to our experiment, we ran the extended algorithm on our data set, and observed that only 8 collections were "problem collections." That is, 23 $(31 - 8)$ of the collections found by the original algorithm were caused by partial mirrors, and were now properly clustered by the extended condition. All the remaining collections $(180 - 8)$ correspond to the desired targets.

In summary, our algorithm (especially with the extension we have described) is very good at identifying what a human would call a replicated collection of web pages (i.e., our target collections).

	Measures / Signatures	Entire document	Four lines	Two lines
Space	Fingerprints	800 MBs	2.4 GBs	4.6 GBs
Time	Compute fingerprints	44 hrs	44 hrs	44 hrs
	Compute trivial similar clusters	97 mins	302 mins	630 mins

Table 1: Storage and time costs for computing trivial similar clusters.

However, it is important to note that in this experiment, the target collections were fully crawled. In practice, our algorithm may be run on data that includes partially crawled collections, and clearly it will not be possible to identify the complete target collection, no matter what clustering algorithm is used. In the next section we explore this more realistic scenario, and show that even with partial crawls, our algorithm can still yield significant benefits.

5 Exploiting similar clusters

To demonstrate the usefulness of similar clusters, we explored the use of our cluster growing algorithm in two applications, crawling and searching. Our results also demonstrate that the algorithm is efficient, since we had to identify clusters in hundreds of gigabytes of web data. For our work we used data crawled by the Stanford Google web crawler [BP99]. We ran our experiments on a SUN UltraSPARC with dual processors, 256 MBs of RAM and 1.4 GBs of swap space, running SunOS 5.5.1.

5.1 Improving crawling

For the first application, we performed a comprehensive performance analysis of web crawlers to identify how much redundant data they crawl. This allowed us to quantify the resources (such as network bandwidth) crawlers waste in visiting multiple copies of pages and collections. As we will see, our performance analysis helped us significantly improve the performance of the Google crawler.

We used the Google crawler since it was developed within our group and we had direct access to its data. We believe that analogous results could be obtained using the data from other crawlers. Indeed, in some recent work, other researchers report results similar to our own in the context of the AltaVista web crawler [BGM97, BB99]. The Google crawler fed us approximately 25 million web pages primarily from domains located in the United States of America. This dataset corresponds to about 150 gigabytes of textual information.

We experimented with three different chunking strategies for identifying similar pages: (1) one fingerprint for the entire document, (2) one fingerprint for every four lines of text, and (3) one fingerprint for every two lines of text. For each scheme we generated a *DocChunks* table, where

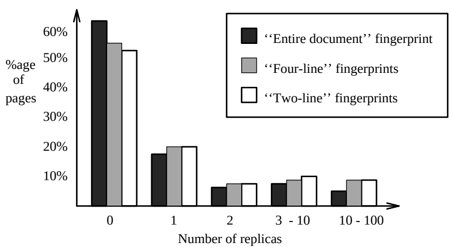


Figure 11: Document replication on 25 million web pages.

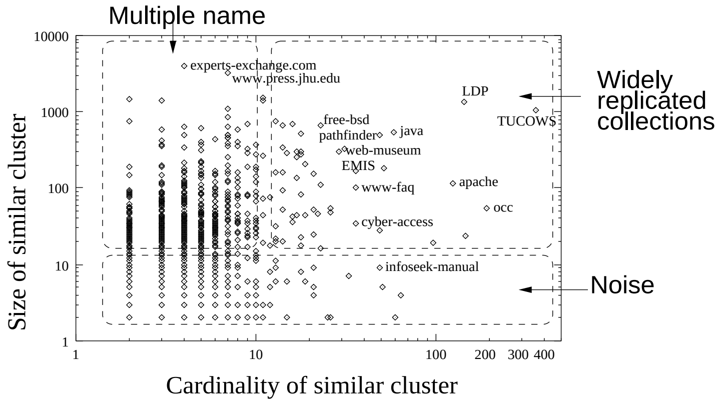


Figure 12: Distribution of maximal clusters.

each tuple gives a URL (represented as a 32-bit integer) and the fingerprint (32 bits) for one chunk in that document. By joining the *DocChunks* table with itself, we can generate the table of shared fingerprints that is the input to the cluster growing algorithm (Section 3.2). To give the reader an idea of the resources utilized, we report in Table 1 the storage cost for table *DocChunks*, as well as the time breakdown in computing the set of trivial similar clusters. The thresholds used were $T = 15$ and $T = 25$ for the "four line" and "two line" fingerprints respectively.

In Figure 11 we report the number of similar pages for each the three chunkings. For instance, let us consider the case when we compute one fingerprint on the entire document. About 64% of the 25 million web pages have no replicas (the left-most darkest bar) except itself: about 18% of pages have an additional exact copy – that is, there are about $\frac{1}{2} * \frac{18}{100} * 25 * 10^6$ distinct pages that have one exact copy among the other $\frac{1}{2} * \frac{18}{100} * 25 * 10^6$ pages in this category. Similarly, about 5% of pages have between 10 and 100 replicas. As expected, the percentage of pages with more than one similar page increases when we relax the notion of similarity from 36% (100 − 64%) for exact replicas, to about 48% (100 − 52%) for "two-line" chunks.

From the above experiments, it is clear that the Google crawler wastes significant resources crawling multiple copies of pages. Our next step was to identify the similar clusters, in order to help Google avoid crawling similar collections. We ran our extended cluster growing algorithm on the collected web graph, a process that took about 20 hours total. The algorithm was modified to consider only collections at a single web site.

In Figure 12, we visualize each cluster as a point in the graph, based on the cluster cardinality and size. For the reader's convenience, we annotate some of the points that correspond to document collections (we manually labeled these points after automatically computing extended clusters). For instance, the LDP point (near the top right corner) indicates that the LDP collection constitutes 1349 pages and has 143 replicas in our crawled data set. Based on our manual examination of the data, we *roughly* partitioned the clusters into the following categories, which are also shown in Figure 12.

Rank	Description	Similar cluster cardinality	Similar cluster size
1	TUCOWS WinSock utilities `http://www.tucows.com`	360	1052
2	LDP Linux Documentation Project `http://sunsite.unc.edu/LDP`	143	1359
3	Apache Web server `http://www.apache.org`	125	115
4	JAVA 1.0.2 API `http://java.sun.com`	59	552
5	Mars Pathfinder `http://mars.jpl.nasa.gov`	49	498

Table 2: Popular web collections.

1. **Widely replicated collections:** We found several extended clusters that corresponded to widely replicated collections. We list the five clusters with the largest cardinalities in Table 2. For instance, the TUCOWS WinSock utilities is replicated at 360 web sites across the world, constitutes about 1052 web pages and the principal copy is located at the listed URL.

2. **Multiple name servers:** In many cases, a set of machines in a single domain share content even if the machines have different IP addresses. Web sites such as `www.experts-exchange.com` fall into this category.

3. **Noise:** Most of the identified clusters in the area labeled "Noise" were not significant or very meaningful. For instance, may web sites have HTML versions of PowerPoint slides created by the "Save As HTML" feature in Microsoft PowerPoint. Readers may be aware that PowerPoint saves each slide as an image file (such as in .GIF format) and creates a slide-show of HTML pages. Each such HTML page has the same content and hyperlink structure but points to the inlined GIF rendering of the corresponding slide. Since our system computes page similarity based on textual content and not based on image similarity, PowerPoint slide-shows are placed into the same cluster. In a few cases, small clusters were induced by "break point" pages. Fortunately, the noise clusters can easily be identified by their small size (less than 10 pages in their collections), so it is easy to ignore them when using similar clusters to improve crawling.

Based on the above rough classification, we precoded information about the widely replicated collections and machines with multiple names into the Google web crawler. We then ran the crawler again and the crawler avoided the precoded collections and machines. This time, 35 million web pages corresponding to about 250 gigabytes of textual data were crawled. Notice that this is a larger crawl than the original one (which was for 25 million pages), so it included a significant amount of new data. Of course, since the web is very dynamic, a significant amount of the "old" data had also changed.

Nevertheless, the second crawl avoided many of the replicated collections very successfully. To confirm this, we again computed the set of similar pages as in the earlier experiments. We observed that the number of similar pages had dropped from a staggering 48% in the previous crawl to a more reasonable 13%. Similarly, the number of identical copies also fell to about 8%. The number of similar and exact pages did not drop to zero in the new crawl because new clusters were identified in the new data. These newly identified replicated collections could be added to the list of pages to avoid in future crawls.

In general, the replica avoiding process is a continuous one. Each crawl identifies new replicated collections that can be avoided in the future. In steady state, the number of newly identified replicas should be relatively small at each iteration. In addition, it is important to periodically revisit what are thought to be replicated collections to see if they have changed and need to be removed from the avoid list. We have not yet implemented such a general replica-avoidance scheme into our crawler, but we believe that the results we have shown here clearly illustrate the large potential savings. As the web continues to grow, and crawlers struggle to keep up [LG99], it will be even more important to avoid replicated pages!

5.2 Improving search engine result presentation

For some applications it makes sense for the crawler to continue gathering multiple copies of a collection. For example, in searching the web, one of the page replicas may be unavailable, so the user may wish to visit another copy. For some data mining, for instance, it may be better to analyze all copies, not just a representative collection. But even if the crawler collects all pages, it may still be very useful to filter out redundancy as information is presented to the user.

To illustrate this point, consider how a web search engine operates today. When we search for concepts such as "object-oriented programming" or "Linux" on the web, search engines return a list of pages ranked using some proprietary ranking function. The resulting display is often often distracting for the following two reasons:

- Multiple pages within a hyperlinked collection are displayed. For instance, in Figure 13 we show an example of how the Google search engine displays[5] results for the search for "XML databases." We see that several pages in `www.techweb.com` satisfy this query and links to all these pages are displayed.
- Links to replicated collections are displayed several times. For instance, in Figure 13, the TechWeb collection that is available at `www.techweb.com` and `techweb.cmp.com` (among other sites), is displayed multiple times.

[5]The screen shots are for the initial Google prototype, not the newer commercial version.

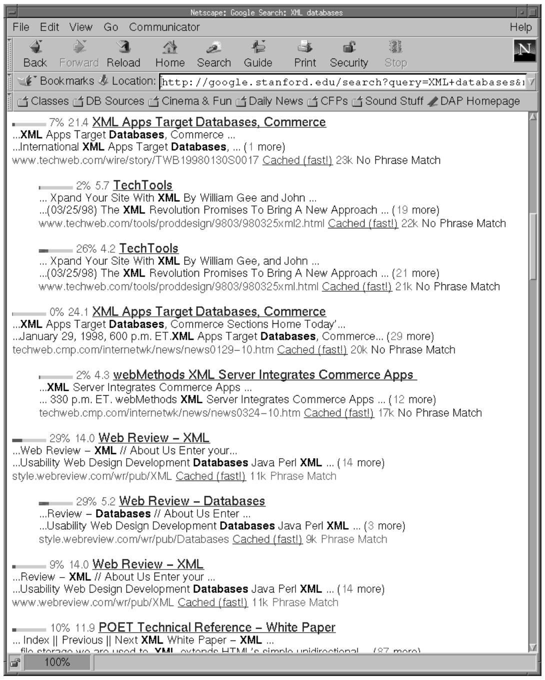


Figure 13: Search results for "XML databases" from Google search engine.

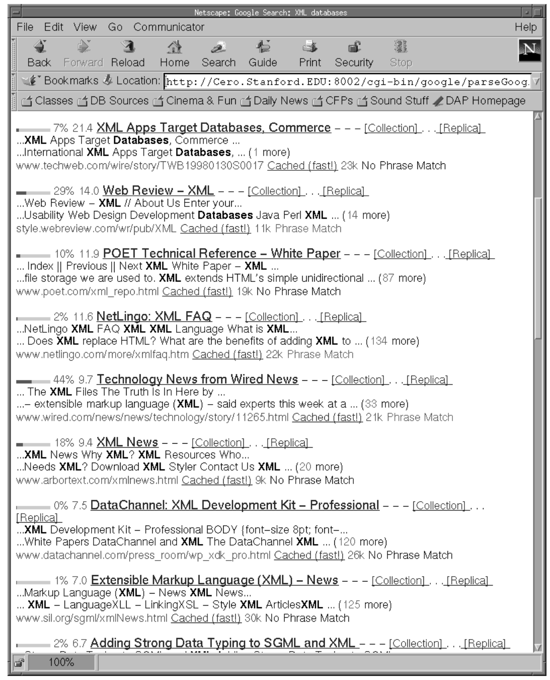


Figure 14: Rolled up search results for "XML databases".

We have implemented a prototype presentation filter that runs on top of the Google search engine to illustrate how search results can be better organized. The prototype computes similar clusters on the data collected by the crawler, and then uses the information to address the two problems identified above. In Figure 14 we show how our prototype displays the results for the user query "XML databases." As we can see, the prototype "rolls up" collections so that it only displays the link of one page in a collection, even if multiple pages within the collection satisfy the query. For example, there is a single page listed from `www.techweb.com`. Also the prototype automatically "rolls up" replicas of collections as well. In our example, the replica pages for `techweb.cmp.com` are not listed at all.

Notice that in our prototype display each result has two additional links marked **Collection** and **Replica**. When a user clicks on the **Collection** link, the collection is rolled down and all pages satisfying the user query are displayed. Similarly, when the user clicks on the **Replica** link, the prototype displays all the replicas of the collection. We believe such an interface is useful to a user, since the interface gives the user a "high-level" overview of the results rather than just a long list of URLs.

We evaluated the above search interface informally within our group. In general, users have found this prototype to be valuable especially for queries in technical areas. For instance, when our users tried queries in topics such as Latex, C++, Latex, JAVA, XML and GNU software, traditional search engines typically yield many redundant results. The same queries yielded much better results in our prototype because of the clustering. Hence we believe that by computing similar clusters and rolling up collections and clusters, we can improve current search engines significantly.

6 Conclusion

In this paper we have introduced a new definition for similarity of collections of web pages. We have also proposed a new algorithm for efficiently identifying similar collections that form what we call a similar cluster. In developing our definitions and algorithm, we have made tradeoffs between the generality of the similar cluster concept and the cost of identifying collections that meet the criteria. No definition of similarity will capture precisely what a human would consider a similar cluster (indeed, more than one human would probably not agree either). Nevertheless, by using our definition and cluster growing algorithm to improve crawling and result displaying, we have shown that our definition and algorithm can be very useful on very large web graphs: the work of a crawler can be reduced by 40%, and results can be much better organized when presented to a user.

There are various directions for future research in the area of replicated web collections. In particular, as mentioned in Section 2, we believe it is possible to post-process the output of our cluster growing algorithm to (a) join together clusters that are separated by small web segments, and (b) extend clusters by adding "nearby" pages. This would require an extended definition of similarity that would allow (a) some small number of pages in one collection not be be mapped, and (b) collections to be of slightly different sizes. An evaluation would be needed to see if the extra processing cost is worth the more general notion of similarity. Also, the concept of similar clusters may be applicable in other domains beyond the web. For instance, any XML or semi-structured database can be viewed as a graph, and hence may contain similar clusters. A clustering algorithm like ours could be used, say, to identify customers with similar interests or similar purchasing patterns.

References

[Ale99] Alexa Corporation. http://www.alexa.com, 1999.

[BB99] Krishna Bharat and Andrei Z. Broder. Mirror, Mirror, on the Web: A study of host pairs with replicated content. In *Proceedings of 8th International Conference on World Wide Web (WWW'99)*, May 1999.

[BGM97] Andrei Broder, Steve C. Glassman, and Mark S. Manasse. Syntactic clustering of the web. In *Sixth International World Wide Web Conference*, pages 391 – 404, April 1997.

[BP99] Sergey Brin and Lawrence Page. Google search engine. http://www.google.com, 1999.

[Bro97] Andrei Broder. On the resemblance and containment of documents. In *Compression and complexity of Sequences (SEQUENCES'97)*, pages 21 – 29, 1997.

[CLR91] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms*. The MIT Press, 1991.

[FSGM+98] Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, and Jeffrey D. Ullman. Computing iceberg queries effciently. In *Proceedings of International Conference on Very Large Databases (VLDB '98)*, pages 299 – 310, August 1998.

[LG99] Steve Lawrence and C. Lee Giles. Accessibility of information on the web. *Nature*, 400:107–109, 1999.

[PE98] M. Perkowitz and O. Etzioni. Adaptive web sites: Automatically synthesizing web pages. In *Fifteenth National Conference on Artificial Intelligence*, 1998.

[PP97] James Pitkow and Peter Pirolli. Life, death, and lawfulness on the electronic frontier. In *International conference on Computer and Human Interaction (CHI'97)*, 1997.

[Sal83] Gerard Salton. *Itroduction to modern information retrieval*. McGraw-Hill, New York, 1983.

[SGM95] Narayanan Shivakumar and Hector Garcia-Molina. SCAM:a copy detection mechanism for digital documents. In *Proceedings of 2nd International Conference in Theory and Practice of Digital Libraries (DL'95)*, Austin, Texas, June 1995.

[SGM96] Narayanan Shivakumar and Hector Garcia-Molina. Building a scalable and accurate copy detection mechanism. In *Proceedings of 1st ACM Conference on Digital Libraries (DL'96)*, Bethesda, Maryland, March 1996.