Scalable Techniques for Clustering the Web

Extended Abstract

Taher H. Haveliwala
Stanford University
taherh@db.stanford.edu

Aristides Gionis
Stanford University
gionis@db.stanford.edu

Piotr Indyk
Stanford University
indyk@db.stanford.edu

ABSTRACT

Clustering is one of the most crucial techniques for dealing with the massive amount of information present on the web. Clustering can either be performed once offline, independent of search queries, or performed online on the results of search queries. Our offline approach aims to efficiently cluster similar pages on the web, using the technique of Locality-Sensitive Hashing (LSH), in which web pages are hashed in such a way that similar pages have a much higher probability of collision than dissimilar pages. Our preliminary experiments on the Stanford WebBase have shown that the hash-based scheme can be scaled to millions of urls.

1. INTRODUCTION

Clustering, or finding sets of related pages, is currently one of the crucial web-related information-retrieval problems. Various forms of clustering are required in a wide range of applications, including finding mirrored web pages, detecting copyright violations, and reporting search results in a structured way. With an estimated 1 billion pages currently accessible on the web [19], the design of highly scalable clustering algorithms is required.

Recently, there has been considerable work on web clustering. The approaches can be roughly divided into two categories[1]

- Offline clustering, in which the entire web crawl data is used to precompute sets of pages that are related according to some metric. Published work on very-large-scale offline clustering has dealt mainly with a metric that provides a *syntactic* notion of similarity (initiated by Broder et al. [5], see also [9]), where the goal is to find pairs or clusters of web pages which are nearly identical.

- Online clustering, in which clustering is done on the results of search queries, according to topic. Recent work has included both link-based (initiated by Dean and Henzinger [8]), and text-based (see Zamir and Etzioni [18]) methods.

Although the syntactic approach for finding duplicates has been tried offline on a large portion of the web, it cannot be used when the form of documents is not an issue (e.g, when two pages, one devoted to "automobiles" and the other focused on "cars," are considered similar). The approaches taken by [5, 9, 17] can not scale to the case where we are looking for similar, as opposed to almost identical, documents. Computing the document-document similarity matrix essentially requires processing the self-join of the relation $DOCS(doc, word)$ on the $word$ attribute, and counting the number of words each pair of documents has in common. The syntactic clustering algorithms [5, 9], on the other hand, use *shingles* or sentences rather than words to reduce the size of the self-join; this of course only allows for copy detection type applications. Although clever algorithms exist which do not require the self-join to be represented explicitly [9], their running time is still proportional to the size of the self-join, which could contain as many as 0.4×10^{13} tuples if we are looking for similar, rather than identical, documents[2]. Assuming the processing power of, say, 0.5×10^6 pairs per second[3], the running time of those algorithms could easily exceed 90 days.

Online methods based on link structure and text have been applied successfully to finding pages on related topics. Unfortunately, the text-based methods are not in general scalable to an offline clustering of the whole web. The link-based methods, on the other hand, suffer from the usual drawbacks of collaborative filtering techniques:

- At least a few pages pointing to two pages are necessary in order to provide evidence of similarity between the two. This prevents search engines from finding the relations early in the page's life, e.g., when a page is first crawled. Pages are discovered to be similar only when a sufficient number of people cocite them. By that time, the web pages are likely to already have been included in one of the popular web directories (e.g., Yahoo! or Open Directory), making the discovery less attractive.

- Link-based methods are sensitive to specific choices made by the authors of web pages; for example, some

[1] As far as the published results are concerned. Several major search engines, including AOL, Excite, Google and Infoseek, offer the "find related web pages" option, but the details of their algorithms are not publicly available.

[2] Our estimation using the sketching approach of [5], see Section 3 for more details of that technique. Unfortunately, the relation is too large to compute its exact size.

[3] Estimation based on the experiment in [17], page 45

people use (and point to) CNN weather information, while others prefer MSNBC, in which case there might be no "bridge" between these two pages.

We describe an ongoing project at Stanford whose goal is to build a scalable, offline clustering tool that overcomes the limitations of the above approaches, allowing topical clustering of the entire web. Our approach uses the text information from the pages themselves and from the pages pointing to them. In the simplest scenario, we might want to find web pages that share many similar words.

We use algorithms based on *Locality-Sensitive Hashing (LSH)*, introduced by Indyk and Motwani [14][4]. The basic idea is to hash the web pages in a way such that the pages which are similar, according to metrics we will discuss later, have a much higher probability of collision than pages which are very different. We show that using LSH allows us to circumvent the "self-join bottleneck" and make web clustering possible in a matter of a few days on modest hardware, rather than months if the aforementioned techniques were used.

In Section 2 we describe our representation of documents. In Section 3 we discuss our similarity measure and provide evidence of its validity. In Section 4 we show how we use LSH techniques to efficiently find pairs of similar documents, and in Section 5 we use the set of similar pairs to generate clusters. We present our initial timing results in Section 6 and finally end with future work in Section 7.

2. BAG GENERATION

We now describe our representation of a given document. The most common representation used in the IR community is based on the vector space model, in which each document is treated as an n-dimensional vector, where dimension i represents the frequency of $term_i$ [16]. Because similarity metric, described further in Section 3, is based on the intersection and union of multisets, we will use an equivalent characterization in which each document doc^u is represented by a bag $B^u = \{(w_1^u, f_1^u), \ldots, (w_k^u, f_k^u)\}$ where w_i^u are the words present in the bag and f_i^u are the corresponding frequencies. We consider two options for choosing which terms are assigned to a document's bag. In the first strategy, we take the obvious approach, and say that B^u for doc^u is given by the multiset of words appearing in doc^u. Our second approach is to define B^u to be the union of all *anchor-windows* referring to doc^u. We will define anchor-windows in detail in Section 2.2, but briefly, it means that $term_j$ appears in B^u for each occurrence of $term_j$ occurring near a hyperlink to doc^u. We chose to drop both low frequency and high frequency words from all bags. For both the content-based and anchor-based approach, we have the option of applying one of the commonly used variants of TFIDF scaling; we scale each word's frequency according to $tfidf_i^u = \sqrt{f_i^u} \times log(\frac{N}{df_i})$, where N is the number of documents, df_i is the overall document frequency of word i, and f_i^u is as before [16]. Finally we normalize the frequencies within each bag so all frequencies sum to a fixed number (in our implementation 100).

[4]A similar algorithm has been independently discovered by Broder [3]

2.1 Content-Based Bags

The generation of content based bags is straightforward. We scan through the web repository, outputting normalized word-occurrence frequencies for each document in turn. The following three heuristics are used to improve the quality of the word bags generated:

- All HTML comments, Javascript code, and non-alphabetic characters are removed. All HTML tags are removed, although image 'alt' text is preserved.
- A custom stopword list containing roughly 750 terms is used. Roughly 300 terms were taken from a popular indexing engine. We then inspected the 500 highest frequency terms from a sample of the repository and included all words except for names of companies and other terms we subjectively decided could be meaningful.
- The well known Porter's stemming algorithm is used to remove word endings [15].

Note that the content-based bags never need to be stored to disk; rather, the stream of bags is piped directly to the min-hash generator described in 3.

2.2 Anchor-Based Bags

The use of page content for clustering is problematic for several reasons. Often times the top level pages in a site contain mostly navigational links and image maps, and may not contain content useful for clustering [1]. Different pages use different styles of writing, leading to the well known linguistic problems of polysemy and synonymy [16].

One way to alleviate these problems is to define the bag representing a document to be the multiset of occurrences of words near a hyperlink to the page. When pages are linked to, the anchor text, as well as the text surrounding the link, henceforth referred to collectively as *anchor-windows*, are often succinct descriptions of the page [1, 6]. The detrimental effects of synonymy in particular is reduced, since the union of all anchor-windows will likely contain most variations of words strongly indicative of the target page's content.

Also note that with the anchor-based approach, more documents can be clustered: because it currently is not feasible to crawl the entire web, for any web-crawl repository of size n, references to more than n pages will be made in anchors.

We now discuss the generation of anchor-based bags. We sequentially process each document in the web repository using the same heuristics given in Section 2.1, except that instead of outputting a bag of words for the current document, we output bag fragments for each url to which the document links. Each bag fragment consists of the anchor text of the url's link, as well as a window of words immediately preceding and immediately following the link. The issue of what window size yields the best results is still being investigated; initial experiments led us to use a window of size 8 (before and after the anchortext) for the results presented in this paper. In addition to anchor-windows, we generate an additional bag fragment for doc^u consisting solely of the words in the title of doc^u.

As they are generated, we write the bag fragments to one of M on-disk buckets, based on a hash of the url, where M is chosen such that a single bucket can fit in main memory. In our case, $M = 256$. After all bag fragments are generated, we sort (in memory) each bucket, collapse the bag fragments for a given url, apply TFIDF scaling as discussed in Section 2, and finally normalize the frequencies to sum to 100.

The remainder of our discussion will be limited to the use of anchor-based bags for representing documents.

3. SIMILARITY MEASURE

The key idea of our approach is to create a small signature for each url, to ensure that similar urls have similar signatures. Recall that each url doc^u is represented as a bag $B^u = \{(w_1^u, f_1^u), \ldots, (w_k^u, f_k^u)\}$. For each pair of urls u and v, we define their similarity as $sim(u, v) = \frac{|B^u \cap B^v|}{|B^u \cup B^v|}$. The extension of the operation of intersection (union, resp.) from sets to bags is defined by taking as the resulting frequency of a word w the minimum (maximum, resp.) of the frequencies of w in the two bags to be intersected (merged, resp.).

Before discussing how we can efficiently find similar documents, we provide evidence suggesting that the above similarity metric applied to anchor-based bags as defined in Section 2.2 provides intuitive and useful results.

For all of our experiments, we used the first 12 million pages of the Stanford WebBase repository, on a crawl performed in January 1999 [11]. The 12 million pages led to the generation of anchor-based bags for 35 million urls.

We tested our approach to defining document-document similarity as follows. We gathered all urls contained at the second level of the Yahoo! hierarchy. We randomly chose 20 of the Yahoo! urls, and found the 10 nearest-neighbors for each among our collection of 35 million urls, using the similarity measure defined above. To find the neighbors for each of the 20 urls, we simply scan through our bags and keep track of the 10 nearest-neighbors for each. Of course this brute force method will not work when we wish to discover pairwise similarities among all 35 million urls in our collection; we will discuss in detail in Section 4 how to use LSH to do this efficiently.

Note that we never utilize Yahoo's classifications; we simply use Yahoo! as a source of query urls. By inspecting the sets of neighbors for each of the Yahoo! urls, we can qualitatively judge how well our measure of document-document similarity is performing. Substantial work remains in both measuring and improving the quality of our similarity measure; a quantitative comparison of how quality is affected based on our parameters (i.e., adjusting anchor-window sizes, using other IDF variants, using page content, etc...) is beyond the scope of our current presentation, but is an important part of our ongoing work. Our initial results suggest that using anchor-windows is a valid technique for judging the similarity of documents. We list seven of the nearest-neighbor sets below. The basic topics of the sets are, respectively: (1) English language studies (2) Dow Jones Index (3) roller-coasters (4) food (5) French national institutes (6) headline news, and (7) pets. In each set, the query url from Yahoo! is first, followed by its 10 nearest neighbors.

```
----------------------
1:
eserver.org
www.links2go.com/go/humanitas.ucsb.edu
eng.hss.cmu.edu
www.rci.rutgers.edu/~wcd/engweb1.htm
www.mala.bc.ca/~mcneil/template.htx
www.links2go.com/more/humanitas.ucsb.edu
www.teleport.com/~mgroves
www.ualberta.ca/~englishd/litlinks.htm
www.links2go.com/add/humanitas.ucsb.edu
english-www.hss.cmu.edu/cultronix
sunsite.unc.edu/ibic/guide.html
----------------------
2:
www.dowjones.com
bis.dowjones.com
bd.dowjones.com
businessdirectory.dowjones.com
www.djinteractive.com/cgi-bin/NewsRetrieval
www.dow.com
www.motherjones.com
www.yahoo.com/Business
rave.ohiolink.edu/databases/login/abig
www.bankhere.com/personal/service/cssurvey/1,1695,,00.html
www.gamelan.com/workbench/y2k/y2k_052998.html
----------------------
3:
www.casinopier-waterworks.com
www.cite-espace.com
www.rollercoaster.com/census/blands_park
world.std.com/~fun/clp.html
www2.storylandnh.com/storyland
www.storylandnh.com
www.rollercoaster.com/census/funtown_pier.html
www.wwtravelsource.com/newhampshire.htm
www.rollercoaster.com/census/casino_pier.html
www.dinosaurbeach.com
www.usatoday.com/life/travel/leisure/1998/t1228tw.htm
----------------------
4:
www.foodchannel.com
www.epicurious.com/a_home/a00_home/home.html
www.gourmetworld.com
www.foodwine.com
www.cuisinenet.com
www.kitchenlink.com
www.yumyum.com
www.menusonline.com
www.snap.com/directory/category/0,16,-324,00.html
www.ichef.com
www.home-canning.com
```

```
----------------------
5:
www.insee.fr
www.ined.fr
www.statistik-bund.de/e_home.htm
www.ineris.fr
cri.ensmp.fr/dp
www.ping.at/patent/index.htm
www.inist.fr
www.inrp.fr
www.industrie.gouv.fr
www.inpi.fr
www.adit.fr
----------------------
6:
www.nando.net/nt/world
www.cnn.com/WORLD/index.html
www.oneworld.org/news/index.html
www.iht.com
www2.nando.net/nt/world
www.rferl.org
www.politicsnow.com
www.cfn.cs.dal.ca/Media/TodaysNews/TodaysNews.html
www.csmonitor.com
www.herald.com
www.pathfinder.com/time/daily
----------------------
7:
www.petnewsinc.com
www.petchannel.com/petindustry/print/vpn/main.htm
www.pettribune.com
www.nwf.org/rrick
www.petchannel.com/reptiles
www.petsandvets.com.au
www.moorshead.com/pets
www.ecola.com/news/magazine/animals
www.thevivarium.com
www.petlifeweb.com
www.menagerie.on.ca
```

4. LOCALITY SENSITIVE HASHING

To describe our algorithms, let us assume for a moment that B^u, as defined in Section 2, is a *set* instead of a *bag*. For this case, it is known that there exists a family $\mathcal{H}$ of hash functions (see [5]) such that for each pair of pages u, v we have $Pr[mh(u) = mh(v)] = sim(u, v)$, where the hash function mh is chosen at random from the family $\mathcal{H}$. The family $\mathcal{H}$ is defined by imposing a random order on the set of all words and then representing each url u by the smallest (according to that random order) element from B^u. In practice, it is quite inefficient to generate fully random permutation of all words. Therefore, Broder et al [5] use a family of random linear functions of the form $h(x) = ax + b \bmod P$; we use the same approach (see Broder et al [4] and Indyk [13] for theoretical background of this technique).

A simple observation is that the notion of a min-wise independent family of hash functions can be extended naturally from sets to bags. This is done by replacing each bag $B = \{(w_1, f_1), \ldots, (w_k, f_k)\}$ by the set $S = \{w_1 1, \ldots, w_1 f_1, \ldots, w_k 1, \ldots w_k f_k\}$, where by $w_i j$ we denote the concatenation of the word w_i with the number j. It is easy to see that for any two bags B^u and B^v we have $|B^u \cap B^v| = |S^u \cap S^v|$ and $|B^u \cup B^v| = |S^u \cup S^v|$.

After flattening each bag B^u to the set S^u, a *Min Hash signature (MH-signature)* can be computed as $min_w\{h(w)|w \in S^u\}$, where $h(\cdot)$ is a random linear function as described above. Such an MH-signature has the desired property that the same value indicates similar urls. However, the method is probabilistic and therefore both false positives and false negatives are likely to occur. In order to reduce these inaccuracies, we apply the *Locality Sensitive Hashing (LSH)* technique introduced by Indyk and Motwani [14]. According to the LSH scheme, we generate m MH-signatures for each url, and compute an *LSH-signature* by concatenating k of these MH-signatures. Since unrelated pages are unlikely to agree on all k MH-signatures, using an LSH-signature decreases the number of false positives, but as a side effect, increases the number of false negatives. In order to reduce the latter effect, l different LSH-signatures are extracted for each url. In that way, it is likely that two related urls agree on at least one of their LSH-signatures[5].

The above discussion motivates our algorithm:

- In the first step, url bags are scanned and m MH-signatures are extracted from each url. This is very easy to implement with one pass over the url bags. This is the only information about urls used by the rest of the algorithm.
- In the second step, the algorithm generates LSH-signatures and outputs similar pairs of urls according these LSH-signatures.

This second step is done as follows:

```
Algorithm: ExtractSimilarPairs
Do l times
  Generate k distinct random indices,
    each from the interval {1 ... m}
  For each url u
    Create an LSH-signature for u, by concatenating
      the MH-signatures pointed by the k indices
  Sort all urls by their LSH-signatures
  For each run of urls with matching LSH-signatures
    Output all pairs
```

The output pairs are written to the disk.

To enhance the quality of our results and reduce false positives, we perform a post-filtering stage on the pairs produced by the ExtractSimilarPairs algorithm. During this stage, each pair (u, v) is validated by checking whether the urls u and v agree on a fraction of their MH-signatures which is at least as large as the desired similarity level (say 20%). If the condition does not hold, the pair is discarded.

[5]For a more formal analysis of the LSH technique see [14, 10, 7].

The implementation of the filtering stage requires a linear scan over the pairs, assuming that all m MH-signatures for all urls fit in the main memory. If this is not the case, more passes over the pair file might be needed. Notice that this step is the most main memory intensive part of our algorithm. In our actual implementation we used two additional techniques to reduce the memory requirements. The first is to keep in memory only one byte from each MH-signature. The second is to validate the pairs on less than m MH-signatures. Both techniques introduce statistical error.
Implementation choices: We chose to represent each MH-signature with $w = 3$ bytes. For each url we extract $m = 80$ MH-signatures, which leads to a space requirement of 240 bytes per url. By picking $k = 3$ the probability that two unrelated urls end up having the same LSH-signature is low; e.g., the probability that two urls with *disjoint* bags collide is at most $1/2^{8*w*k} = 1/2^{48}$, which guarantees a very small number of false positives (for about 20 million urls). On the other hand, since we look for pairs with similarity at least 20%, a *fixed* pair of urls with similarity 20% gets the same MH-signature with probability 2/10, and the same LSH-signature with probability $(2/10)^k = 1/125$. In order to ensure that the pair would finally be discovered, that is, to ensure a small probability of false negatives, we have to take about 125 different LSH-signatures ($l = 125$).

5. CLUSTERING

The set of similar document pairs S, generated by the algorithm discussed above, must then be sorted. Note that each pair appears twice, both as (u, v) and (v, u). Sorting the pairs data, which is close to 300 GB for 20 million urls, is the most expensive step of our procedure. After the sort step, we can efficiently build an index over the pairs so that we can respond to "What's Related" type queries: given a query for document u we can return the set $\{v|(u, v) \in S\}$.

We can proceed further by using the set of similar pairs, which represents the document-document similarity matrix, to group pages into flat clusters. The clustering step allows a more compact final representation than document pairs, and would be a necessary for creating hierarchies[6].

To form flat clusters, we use a variant of the *C-LINK* algorithm due to Hochbaum and Shmoys [12], which we call *CENTER*. The idea of the algorithm is as follows. We can think of the similar pairs generated earlier as edges in a graph (the nodes correspond to urls). Our algorithm partitions the graph in such a way that in each cluster there is a *center* node and all other nodes in the cluster are "close enough" to the center. For our purposes, "close enough" means that there is an edge in the graph; that it, there is a pair found in the previous phase that contains the node and its center.

CENTER can be implemented very efficiently. The algorithm performs a sequential scan over the sorted pairs. The first time that node u appears in the scan, it is marked as a cluster center. All subsequent nodes v that appear in pairs of the form (u, v) are marked as belonging to the cluster of u and are not considered again.

[6]We have not yet explored generating hierarchies

6. RESULTS

We discuss only the anchor-window approach here, although the content based approach requires a similar running time. As discussed in Section 3, our dataset consists of 35 million urls whose anchor-bags were generated from 12 million web pages. For the timing results presented here, we applied our LSH-based clustering technique to a subset of 20 million urls.

The timing results of the various stages are given in Table 1. We ran the first four steps of the experiment on dual Pentium II 300 MHz, with 512 MB of memory. The last two steps were performed on dual Pentium II 450 MHz, with 1 GB of memory. The timings for the last two steps are estimated from the time needed to generate the first 10,000 clusters, which is as many as we can inspect manually.

Algorithm step	No. CPUs	Time
bag generation	2	23 hours
bag sorting	2	4.7 hours
MH-signature generation	1	26 hours
pair generation	1	16 hours
filtering	1	83 hours
sorting	1	107 hours
CENTER	1	18 hours

Table 1: Timing results

Developing an effective way to measure cluster quality when the dataset consists of tens of millions of urls is an extremely challenging problem. As discussed more formally in [14, 10, 7], the LSH technique has probabilistic guarantees as to how well nearest-neighbors are approximated. Thus the initial results for the quality of exact nearest-neighbors that we described in Section 3 are indicative of our clustering quality. We are currently investigating techniques to analyze more thoroughly the overall clustering quality given the scale of our input.

7. FUTURE WORK

We are actively developing the techniques we have introduced in this paper. We plan to integrate our clustering mechanism with the Stanford WebBase to facilitate user feedback on our cluster quality, allowing us to both measure quality and make further enhancements. We also plan to experiment with a hybrid approach to clustering, by first using standard supervised classification algorithms to preclassify our set of urls into several hundred classes, and then applying LSH-based clustering to the urls of each of the resulting classes. This would help desensitize our algorithm from word ambiguity, while still allowing for the generation of fine-grained clusters in a scalable fashion.

8. REFERENCES

[1] E. Amitay, "Using common hypertext links to identify the best phrasal description of target web documents ", SIGIR'98 (Workshop on Hypertext Information Retrieval for the Web).

[2] A. Broder, "On the resemblance and containment of documents", SEQUENCES'98, p. 21-29.

[3] A. Broder, "Filtering near-duplicate documents", FUN'98.

[4] A. Broder, M. Charikar, A. Frieze, M. Mitzenmacher, "Min-wise independent permutations", STOC'98.

[5] A. Broder, S. Glassman, M. Manasse, G. Zweig, "Syntactic clustering of the Web", WWW6, p. 391-404, 1997.

[6] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam and S. Slattery, "Learning to Extract Symbolic Knowledge from the World Wide Web", AAAI'98

[7] E. Cohen, M. Datar, S. Fujiware, A. Gionis, P. Indyk, R. Motwani, J. Ullman, and C. Yang, "Finding interesting associations without support pruning", ICDE'2000.

[8] J. Dean, M. Henzinger, "Finding related web pages in the world wide web", WWW8, p., 389-401, 1999.

[9] M. Fang, H. Garcia-Molina, R. Motwani, N. Shivakumar and J. Ullman , "Computing iceberg queries efficiently", VLDB'98.

[10] A. Gionis, P. Indyk, R. Motwani, "Similarity search in high dimensions via hashing", VLDB'99.

[11] J. Hirai, S. Raghavan, H. Garcia-Molina, A. Paepcke, "WebBase: A repository of web pages", WWW9

[12] D. Hochbaum, D. Shmoys, "A best possible heuristic for the k-center problem", Mathematics of Operations Research, 10(2):180-184, 1985.

[13] P. Indyk, "A small minwise independent family of hash functions", SODA'99.

[14] P. Indyk, R. Motwani, "Approximate nearest neighbor: Towards removing the curse of dimensionality", STOC'98.

[15] M. Porter "An algorithm for suffix stripping", Program 14(3):130-137, 1980.

[16] G. Salton, M. J. McGill "Introduction to Modern Information Retrieval", McGraw-Hill Publishing Company, New York, NY, 1983.

[17] N. Shivakumar, "Detecting Digital Copyright Violations on the Internet" Ph.D. thesis, Stanford University, 1999.

[18] O. Zamir, O. Etzioni "Web document clustering: A feasibility demonstration", SIGIR'98.

[19] "Inktomi WebMap", http://www.inktomi.com/webmap/