Tracking Web Spam with Hidden Style Similarity

Tanguy Urvoy, Thomas Lavergne, Pascal Filoche

France Telecom R&D*

{tanguy.urvoy,thomas.lavergne,pascal.filoche}@orange-ft.com

ABSTRACT

Automatically generated content is ubiquitous in the web: dynamic sites built using the three-tier paradigm are good examples (e.g. commercial sites, blogs and other sites powered by a web authoring software), as well as less legitimous spamdexing attempts (e.g. link farms, faked directories...).

Those pages built using the same generating method (template or script) share a common "look and feel" that is not easily detected by common text classification methods, but is more related to stylometry.

In this paper, we present a (hidden) style similarity measure based on extra-textual features in HTML source code. We also describe a method to clusterize a large collection of documents according to this measure. The clustering algorithm being based on fingerprints, we also give some recalls about fingerprinting.

By conveniently sorting the generated clusters, one can efficiently track back instances of a particular automatic content generation method among web pages collected using a crawler. This is particularly useful to detect pages across different sites sharing the same design — this is often a good hint of either spamdexing attempt or mirrored content.

1. INTRODUCTION

Automatically generated content is nowadays ubiquitous on the web, especially with the advent of professional web sites and popular three-tier architectures such as "LAMP" (Linux Apache Mysql Php). Generation of these pages using such architecture involves:

- a scripting component;
- a page template ("skeleton" of the site pages);
- content (e.g. product catalog, articles repository...), usually stored in databases.

When summoned, the scripting component combines the page template with information from the database to generate an HTML page, having no difference with a static HTML page from a robot crawler point of view (shall robots have point of view).

*Thomas Lavergne also ENST Paris

AIRWEB'06, August 10, 2006, Seattle, Washington, USA.

1.1 Spamdexing and Generated Content

By analogy with e-mail spam, the word *spamdexing* designates the techniques used to reach a web site to a higher-than-deserved rank in search engines response lists. For instance, one well known strategy to mislead search engines ranking algorithms consists of generating a maze of fake web pages called *link farm*.

Apart from the common dynamic web sites practice, the ability to automatically generate a large amount of web pages is also appealing to web spammers. Indeed [3] points out that *"the only way to effectively create a very large number of spam pages is to generate them automatically"*.

When those pages are all hosted under a few domains, the detection of those domains can be a sufficient countermeasure for a search engine, but this is not an option when the link farm spans hundreds or thousands of different hosts — for instance using word stuffed new domain names, or buying expired ones [6].

One would like to be able to detect all pages generated using the same method once a spam page is detected in a particular search engine response list. One direct application of such a process would be to enhance the efficiency of search engines blacklist databases by "spreading" detected spam information to find affiliate domains (following the philosophy of [7]).

1.2 Detecting Generated Pages

We see the problem of spam detection in a search engine back office process as two-fold:

- detecting new instances of already encountered spam (through editorial review or automatic methods);
- pinpointing dubious sets of pages in a large uncategorised corpus.

The first side of the problem relates to supervised classification and textual similarity, while the second is more of the unsupervised clustering kind.

1.2.1 Detecting Similarity With Known Spam

Text similarity detection usually involves word-based features, such as in e-mail Bayesian filtering. This is not always relevant in our case, because though those pages share the same generation method, they rarely share the same vocabulary [15] (apart from the web spam specifically involving adult content) — hence using common text filtering methods with this kind of web spam would miss a lot of positive instances. For example exiled presidents and energising sex drugs are recurrent topics in e-mail spam, but link farm

automatically generated pages tend to rather use large dictionary in order to span a lot of different possible requests [6].

To detect similarity based on pages generation method, one needs to use features more closely related to the internal structure of the HTML document. For instance, [13] proposed to use HTML specific features along with text and word statistics to build a classifier for genre of web documents.

1.2.2 Stylometry and HTML

In fact, what best describes the relationship between those pages generated using the same template or method seems to be more on a *style* ground than a *topical* one. This would relate our problem with the *stylometry* area. Up to now, stylometry was more generally associated with authorship identification, to deal with problems such as attributing plays to the right Shakespeare, or to detect computer software plagiarism [5]. Usual metrics in stylometry are mainly based on word counts [12], but also sometimes non-alphabetic features such as punctuation. In the area of web spam detection, [15] and [11] propose to use lexicometric features to classify the part of web spam that does not follow regular language metrics.

1.2.3 Overview of This Paper

We first give some recalls about similarity, fingerprints and clustering in section 2. We then detail the specificities of the *"Hidden Style Similarity"* algorithm in section 3. The experimental results are described in section 4.

2. SIMILARITY AND CLUSTERING

2.1 Similarity Measure

The first step before comparing documents is to extract their (interesting) content: this is what we call *preprocessing*. The second step is to transform this content into a model suitable for comparison (except for string edition based distances like Levenshtein and its derivatives where this intermediate model is not mandatory). For frequencies based distances the second step consists of splitting up the documents into multi-sets of parts (frequencies vectors). For set intersection based distances, the split is done into sets of parts. Depending on the granularity expected, these parts may be sequences of letters (n-grams), words, sequences of words, sentences or paragraphs. The parts may overlap or not.

There are many flavors of similarity measure [14, 16]. The most used measure in stylometry is the Jaccard similarity index. For two sets of parts D_1, D_2:

$$Jaccard(D_1, D_2) = \frac{|D_1 \cap D_2|}{|D_1 \cup D_2|} \ .$$

Variants may be used for the normalizing factor, such as in the Dice index:

$$Dice(D_1, D_2) = 2 \cdot \frac{|D_1 \cap D_2|}{|D_1| + |D_2|} \ .$$

Whatever kind of normalisation used for $|D_1 \cap D_2|$, the most important ingredients for the quality of comparison are the preprocessing step and the parts granularity.

The one-to-one calculus of similarities is interesting for fine comparison into a small set of documents, but the quadratic explosion induced by such a brute-force approach is unacceptable at the scale of a web search engine. To circumvent this explosion, more tricky methods are required. These methods are described in the next three sections.

2.2 Fingerprints

The technique of documents fingerprinting and its application for similarity clustering is a kind of *locality sensitive hashing* [9]. We mainly based our work on the papers [8], [2] and [1], where the pratical use of minsampling (instead of random sampling) and its leverage effect on similarity estimation is well described. We also noticed a phrase level use of fingerprints to track search engine spam in [4].

2.2.1 Minsampling

Each document is split up into parts. Let us call P the set of all possible parts. The main principle of *minsampling* over P is to fix at random a linear ordering on P (call it $\prec$) and represent each document $D \subseteq P$ by its m lowest elements according to $\prec$ (we denote this set $Min_{\prec,m}(D)$).

If $\prec$ is chosen at random over all permutations over P then for two random documents $D_1, D_2 \subseteq P$ and for m growing, it is shown in [1] that

$$\frac{|Min_{\prec,m}(D_1) \cap Min_{\prec,m}(D_2) \cap Min_{\prec,m}(D_1 \cup D_2)|}{|Min_{\prec,m}(D_1 \cup D_2)|}$$

is a non biased estimator of $Jaccard(D_1, D_2)$.

2.2.2 Fingerprints and Index Storage

With this fingerprinting method, the fingerprint of a document D is stored as a sorted list of m integers (which are hashing keys of the elements of $Min_{\prec,m}(D)$). The experiments of [8] show that a value around $m = 100$ is reasonable for the detection of similar documents in a large database. The drawback of this method it that it does not provide a real vector space structure to the fingerprints. A vector space structure is more convenient, for instance to build indexes in a database to later fetch near duplicates of a particular document.

2.2.3 Optimization of Minsampling

An improvement of the model is to use m independent linear orderings over P, let us call them $\prec_i$ for $i \in [m]$, and use these ordering to select one minimun element by ordering. The result is a real vector of m independant integers and the similarity measure becomes:

$$Sim_a(D_1, D_2) = \frac{\sum_{i=0}^{m} |Min_{\prec_i}(D_1) \cap Min_{\prec_i}(D_2)|}{m}$$

which is a correct estimator of the Dice similarity.

2.3 Clustering

When working on large volume of documents, one would like to group together the documents which are similar enough according to the chosen similarity. This is especially useful when no specific paragon to look for is known in advance.

If D is the set of documents, we want to compute a mapping $Cluster : D \to D$ associating to each element x its class representative $Cluster(x)$, with $Cluster(x) = Cluster(y)$ if and only if $sim(x, y)$ is lower than a given threshold.

2.3.1 Clustering with Fingerprints

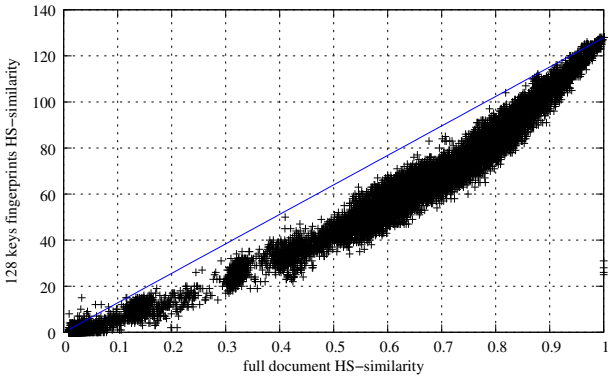

Figure 1: The rate of matched dimensions according to the full document HS-similarity (one-to-one comparison between 10000 HTML files).

The first benefit of using fingerprints is to reduce the size of documents representatives, allowing to perform all computation in memory. As shown in figure 1, this reduction by sampling is at the cost of a little loss of quality in the similarity estimation.

Another important benefit of fingerprints is to give a low dimension representation of documents. It becomes possible to compare only the documents that match at least on some dimensions. This is a way to build the sparse similarity matrix with some control over the quadratic explosion induced by the biggest clusters [2].

3. THE HSS ALGORITHM

To capture similarity based on pages generation method, we propose to use a specific document preprocessing excluding all alpha-numeric characters, and keeping into account the remaining characters through the use of n-grams. By analysing usually neglected features of HTML texts like extra-spaces, lines feed or tags, we are able to modelize the "style" of HTML documents. This model enables us to compare and group together documents sharing many hidden features.

We consider both the one-to-one full document HS-similarity and the global HS-clustering of several documents. These two aspects of the HSS algorithm are described in figure 2.

As a side effect, the algorithm is efficient to characterize HTML documents coming from the same web site without information about the host or URL, but the most interesting results are similarity classes containing pages across many differents domains yet with a high HS-similarity.

3.1 Preprocessing

The usual setup procedure to compare two documents is to first remove everything that do not reflect their content. In the case of HTML documents, this preprocessing step may include removing of tags, of extra spaces and of stop words. It may also include normalization of words by capitalization or stemming.

The originality of our approach is to do exactly the opposite: we keep only the "noisy" parts of HTML documents by removing any alphanumeric character. For example, applying such a preprocessing to a relatively human readable HTML code like:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xht...
```

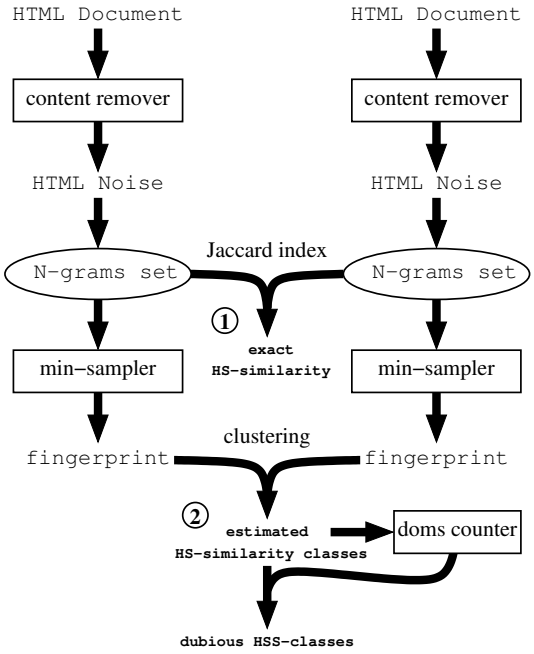

Figure 2: The HSS algorithm: step (1) describes one-to-one full document HS-similarity computation, step (2) describes large scale similarity classes calculus and link farm detection.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"><head><title>Th...

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<meta http-equiv="Keywords" content="GNU, FSF, Free Software Foundation, Linux, Em...
<meta http-equiv="Description" content="Since 1983, developing the free UNIX style...
<link rev="made" href="mailto:webmasters@gnu.org">
<link rel="stylesheet" type="text/css" href="gnu_fichiers/gnu.css">
```

gives an HTML noise such as:

```
<!   "-////  . //" "://..////-.">
< ="://..//" :="" =""><><>   </>

< -="-" ="/; =-">
< -="" =", ,   , , , , ,  , ,  , ,  ">
< -="" =" ,        ,              .">
< ="" =":.">
< ="" ="/" ="/.">
```

Using non-alphanumeric characters – in the case of standard text, these are punctuation signs – as features to classify text is not completely unusual ([10], [13]), though most use it only as a complementary hint. Since HTML syntax includes a lot of non-alphanumeric characters, they happen to be very relevant in our case. Because it is straightforward, this filtering of HTML text is also extremly fastly computed.

3.2 Similarity

For scalability reasons, we chose to use a set intersection based distance with overlaping n-grams on the preprocessing output as parts. A simple one-to-one style similarity measure can then be computed using formula from 2.1 But as said earlier, using this one-to-one similarity measure doesn't fit for large scale clustering. Using fingerprints on the preprocessing output is required to address the scalability issue.

3.3 Fingerprints

We chose to use minsampling with m independant orderings, with another (speedup) improvement which consists of using a *pre-hashing* function to select in advance which dimensions of the final vector are concerned by a given part

of the document. Formally, if $(C_0, \ldots, C_{m-1})$ is the partition of P induced by the pre-hashing function, we have the following similarity measure:

$$Sim_b(D_1, D_2) = \frac{\sum_{i=0}^{m} |Min_{\prec_i}(D_1 \cap C_i) \cap Min_{\prec_i}(D_2 \cap C_i)|}{m}$$

This pre-hashing avoids the heavy calculus of m linear orderings for each considered part, and also avoids to fill two dimensions of the vector with the same part of a document. The drawback is that small documents may not contain enough parts to fill all dimensions of their fingerprint vector.

We chose to ignore these empty dimensions in the counting of matched dimensions, thus lowering drastically the similarity estimation for small documents. This side effect is not critical, HS-similarity diagnostic being by essence unreliable for small documents. Figure 1, shows a comparison between exact Dice measure and Sim_b measure on fingerprints (with $m = 128$).

3.3.1 Implementation

Each document is preprocessed on the fly. The parts we used are overlapping n-grams hashed into 64 bits integers. To compute the m independent orderings $\prec_i$, we precompute m permutations $\sigma_i : [2^{64}] \rightarrow [2^{64}]$ and compare the permuted values:

$$x \prec_i y \Leftrightarrow \sigma_i(x) < \sigma_i(y)$$

To compute these permutations, we use a subfamily of permutations of the form $\sigma_i = \sigma_i^1 \circ \sigma_i^2$ where σ_i^1 is a bits shuffle and $\sigma_i^2(x)$ is an exclusive OR mask. After having initialized every dimension of the fingerprint vector $V \in \mathbb{N}^m$ to ∞, we evaluate each n-gram of the HTML noise with Procedure 1.

Procedure 1 Insert a string s by minsampling into a fingerprint $V \in \mathbb{N}^m$.

Require: $m > 0$ and V initialized
  $h := preHash(s)$
  $i := h \mod m$
  $h' := \sigma_i(h)$
  if $h' < V[i]$ **then**
    $V[i] := h'$
  end if

3.4 Clustering

We used a variant from the algorithm described in [2]. It does not build the entire *similarity graph* and uses a probing heuristic to find potentially similar pairs.

3.4.1 Using Quasi-Transitivity on Similarity Matrix

By thresholding the similarity matrix (cf. 2.3.1), we obtain a symetric relation (let us call it *similarity graph*):

$$S = \{(x, y) \in D \times D \mid sim(x, y) < threshold\}$$

Similarity graphs are characterized by their *quasi-transitivity* property: if xSy and ySz then there is a high probability that xSz. In other words, the connected components of these graphs are almost equivalence classes. This *quasi-transitivity* is helpful to accelerate the clustering process. If the relation is transitive enough, any element may be used as reference to decide if other elements are in the same class. In practice, though building the full similarity graph is too expensive, the noise induced by sampling raises some interest in using a bit of redundancy to improve the robustness of the process (cf. figure 3).

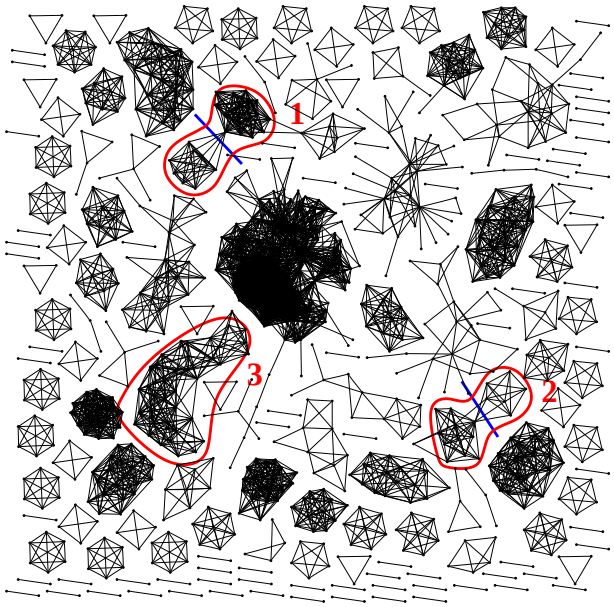

Figure 3: The *quasi-transitivity* of estimated HS-similarity relation is well illustrated by this full similarity graph (realized from 3000 HTML files). With a transitive relation, each connected component would be a clique. For "bunch of grapes" components like (1) and (2), a clear cut may be done but for "worms" components like (3) the similarity diagnostic is less clear.

To approximate the clustering map $Cluster$, we use an algorithm controlled by two parameters: a probe threshold p and a check threshold t. Depending of the aggressiveness of hashing, it may be useful to group fingerprint keys. This introduces a new parameter k to define the size of these groups. We first generate p random subsets of size k in $[m]$. By projecting and indexing fingerprint vectors p times according to these k-subsets, we probe for potential similar pairs which are then checked according to t (See Procedure 2).

Procedure 2 Search HS-similarity classes

Require: $0 < k, p, t \leq m$
  init similarity graph;
  for $i := 0$ to p **do**
    pick a k-subset $s \subseteq [m]$;
    for all pairs (x, y) of fingerprints matching according to s **do**
      if $Sim_b(x, y) > t$ **then**
        add edge (x, y) to similarity graph;
      end if
    end for
  end for
  compute $Clusters$ from connected components of the graph;

3.4.2 Setting Up of Parameters

A good way to choose parameters p, t and k is to make sure that the probability to miss a pair of similar documents during the probing step is under a certain value. With $k = 1$, for a check threshold t and a probe threshold p, the probability of missing a pair of similar documents is dominated by:

$$P_{miss} = \frac{\binom{m-p}{t}}{\binom{m}{t}} = \frac{(m-t-1)\dots(m-t-p+1)}{m\dots(m-p+1)}$$

The actual error rate is lower due to the quasi-transitivity of the similarity graph. With $m = 128$ and $k = 1$ the relation $p \cdot t \geq 512$ ensures an edge missing probability lower than 1%.

4. EXPERIMENTAL RESULTS

We used for experimentation a corpus of five million HTML pages crawled from the web. This corpus was built by combining tree crawl strategies:

- a deep internal crawl of 3 million documents from 1300 hosts of DMOZ directory;
- a flat crawl of 1 million documents from a french search engine blacklist (with many adult content);
- a deep breadth first crawl from 10 non adult spam urls (chosen in the blacklist) and 10 trustable urls (mostly from french universities).

At the end of the process, we estimated roughly that 2/5 of the collected documents where spam.

After fingerprinting, the initial volume of 130 GB data to analyse was reduced down to 390 Mo, so the next step could easily be made in memory.

4.1 One-to-All Similarity

The comparison between one HTML page and all other pages of our test base is a way to estimate the quality of HS-similarity. If the reference page comes from a known web site, we are able to judge the quality of HS-similarity as web site detector. In the example considered here: `franao.com` (a web directory with many links and many internal pages), a threshold of 20/128 gives a `franao` web site detector which is not based on URLS. On our corpus, this detector is 100% correct according to URLS prefixes.

By sorting all HTML documents by decreasing HS-similarity to one randomly chosen page of `franao` web site, we get a decreasing curve with different gaps (Figure 4). These gaps are interesting to consider in detail:

- The first 20,000 HTML pages are long lists of external links, all built from template 1 (Figure 5);
- Around 20,000 there is a smooth gap (1) between long list and short list pages, but up to 95,000, the template is the same;
- Around 95,000, there is a strong gap (2) which marks a new template (Figure 6): up to 180,000, all pages are internal `franao` links built according to template 2;
- Around 180,000 there is a strong gap (3) between `franao` pages and other web sites pages.

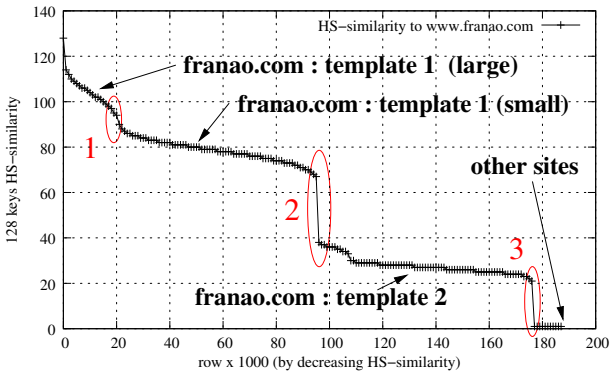

Figure 4: By sorting all HTML documents by decreasing HS-similarity with one reference page (here from `franao.com`) we get a curve with different gaps. The third gap (around 180,000) marks the end of `franao` web site.

Figure 5: `franao.com` template 1 (external links)

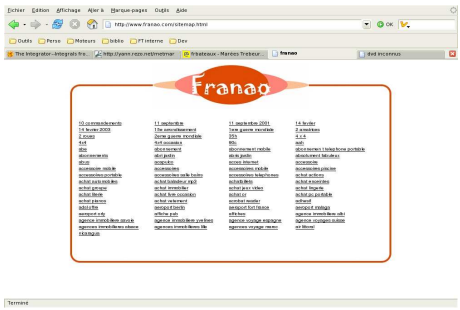

Figure 6: `franao.com` template 2 (internal links)

Table 1: Clusters with highest mean similarity and domain count

URLS	Domains	Mean similarity	Prototypical member (centroïd)	
268	231	1	www.9eleven.com/index.html	Copy/Paste
93148	313	0.58	www.les7laux.com/hiver/forum/phpBB2/membe...	Template (Forums)
3495	255	0.33	www.orpha.net/static/index.html	Template (Apache)
966	174	0.40	www.asliguruney.com/result.php?Keywords=m...	Link farm
122	91	0.74	anus.fistingfisting.com/index.htm	Copy/Paste
1148	173	0.38	www.basketmag.com/result.php?Keywords=gif...	Link farm
19834	164	0.40	www.series-tele.fr/index.html?mo=serie_t...	Template
122	55	0.91	www.ie.gnu.org/philosophy/index.html	Mirror
139	101	0.44	www.reha-care.net/home_buying.htm?r=p	Link farm
218	195	0.21	chat.porno-star.it/index.html	Copy/Paste
177	60	0.67	www.ie.gnu.org/home.html	Mirror
2288	44	0.90	www.cash4you.com/insuranceproviders/index...	Link farm
626900	70	0.52	animalworld.petparty.com/automotivecenter...	Link farm
168	96	0.32	www.google.ca/intl/en/index.html	Mirror
214	61	0.50	shortcuts.00go.com/shorcuts.html	Link farm
42314	112	0.26	forums.cosplay.com/index.html	Template
121	63	0.41	collection.galerie-yemaya.com/index.html	Copy/Paste
555	34	0.68	allmacintosh.digsys.bg/audiomac_rating.h...	Template
114	77	0.29	www.gfx-revolution.com/search/webarchiv.p...	Link farm
286	60	0.35	gnu.typhon.net/home.sv.html	Mirror

4.2 Global Clustering

To clusterize the whole corpus we need to raise the threshold to ensure a low level of false positive. In our experiments we chose $n = 32$, $m = 128$, $k = 1$, $t = 35$ and $p = 20$. With a similarity score of at least 35/128, the number of misclassified URLS seems negligible but some clusters are split into smaller ones.

We obtained 43,000 clusters with at least 2 elements. The table 1 shows the 20 clusters sorted by highest *mean similarity* × *domain count*. (For the sake of readability some mirror clusters have been removed from the list.)

In order to evaluate the quality of the clustering, the first 50 clusters, as well as 50 other randomly chosen ones, were manually checked, showing no misclassified URLS.

Most of the resulting clusters belong to one of these classes:

1. *Template* clusters groups HTML pages from dynamic web sites using the same skeleton. The cluster #2 of figure 1 is a perfect example, it groups all forum pages generated using the PhpBB open source project. The cluster #3 is also interesting: it is populated by Apache default directory listings;

2. *Link farm* clusters are also a special case of *Template*. They contain numerous computer generated pages, based on the same template and containing a lot of hyperlinks between each other;

3. *Mirrors* clusters contain sets of near duplicate pages hosted under different servers. Generally only minor changes were applied to copies like adding a link back to the server hosting the mirror;

4. *Copy/Paste* clusters contain pages that are not part of mirrors, but do share the same content: either a text (e.g. license, porn site legal warning...), a frameset scheme, or the same javascript code (often with few actual content).

Table 2: A sample template cluster

	URL
1	www.les7laux.com/hiver/forum/phpBB2/me...
0.68	ksosclan.free.fr/phpBB2/login.php
0.67	www.quartertothree.com/phpBB2/profile.ph...
0.65	www.lirone.com/forum/login.php?redirect=...
0.64	www.francemule.com/forum/login.php?redir...
0.63	ksosclan.free.fr/phpBB2/login.php?redire...
0.62	90plan.ovh.net/ lillofor/login.php?redir...
0.60	www.artisanatweb.com/phpBB/viewtopic.php...
0.57	www.artisanatweb.com/phpBB/viewforum.ph...
0.54	www.dualforum.com/profile.php?mode=viewp...
0.53	www.dualforum.com/viewforum.php?f=52
0.56	bande2floydiens.free.fr/forum/posting.ph...
0.33	forum.p2pfr.com/posting.php?mode=quote&a...
0.28	a.chavasse.free.fr/phpBB2/viewtopic.php?...
0.25	www.e-hotellerie.com/forum/index.html?c=...

The first two cluster classes are the most interesting benefit of the use of HSS clustering. They allow an easy classification of web pages by categorizing a few of them.

Table 2 shows sample URLS of a cluster with the associated similarity against the center of the cluster. This cluster gathers URLS from forum build with phpBB. Some of these could have been classified with simple methods like a search for the typical string *"phpBB2"* in the URL, but this would overlook some web sites that integrate phpBB with some changes in the display style. Using HSS algorithm allows to gather those forums in the same cluster.

Classical algorithms for similarity could be used to extract the last two cluster classes. Using HSS algorithm enables to quickly build a first clustering of the pages, and next use more expensive methods to refine this clustering, reducing the total computing time.

5. CONCLUSION

We presented in this paper a method to compute a distance based on HTML extra-textual features (*Hidden Style Similarity*). A computationally efficient method to cluster documents based on this similarity has been detailed, and some results on a test corpus have been commented.

This method finds several uses in a search engine back-office process: it makes it possible to cluster pages based on the template and writing style. It enables to find particular instances of well-known pages genre such as forum pages or Apache directory listings, so as to tag or skip them in a search engine response list. Given a set of already known undesirable pages, other pages sharing the same template can be sought after and tagged for deletion or editorial review.

Apart from the editorial detection of web spam, a complementary useful process is to point out large clusters of similar pages spanning several domains: this is often a good hint of either sites mirroring or automatic spam pages generation, both things being valuable information to be processed by a search engine back office.

6. REFERENCES

[1] A. Z. Broder. On the resemblance and containment of documents. In *Proceedings of Compression and Complexity of Sequences*, page 21, 1998.

[2] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. In *Selected papers from the sixth international conference on World Wide Web*, pages 1157–1166, Essex, UK, 1997. Elsevier Science Publishers Ltd.

[3] D. Fetterly, M. Manasse, and M. Najork. Spam, damn spam, and statistics: using statistical analysis to locate spam web pages. In *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases*, pages 1–6, New York, NY, USA, 2004. ACM Press.

[4] D. Fetterly, M. Manasse, and M. Najork. Detecting phrase-level duplication on the world wide web. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 170–177, New York, NY, USA, 2005. ACM Press.

[5] A. Gray, P. Sallis, and S. MacDonell. Software forensics: Extending authorship analysis techniques to computer programs. In *3rd Biannual Conference of International Association of Forensic Linguists (IAFL '97)*, pages 1–8, 1997.

[6] Z. Gyöngyi and H. Garcia-Molina. Web spam taxonomy. In *First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2005.

[7] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with trustrank. In *VLDB*, pages 576–587, 2004.

[8] N. Heintze. Scalable document fingerprinting. In *1996 USENIX Workshop on Electronic Commerce*, November 1996.

[9] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. of 30th STOC*, pages 604–613, 1998.

[10] B. Kessler, G. Nunberg, and H. Schütze. Automatic detection of text genre. In *Proceedings of the 35th Annual Meeting of the ACL and 8th Conference of the EACL*, pages 32–38, 1997.

[11] T. Lavergne. Unnatural language detection. In *Proceedings of RJCRI'06 : Young Scientists' conference on Information Retrieval*, 2006.

[12] T. McEnery and M. Oakes. Authorship identification and computational stylometry. In *Handbook of Natural Language Processing*. Marcel Dekker Inc., 2000.

[13] S. Meyer Zu Eissen and B. Stein. Genre classification of web pages. In S. Biundo, T. Frühwirth, and G. Palm, editors, *Proceedings of KI-04, 27th German Conference on Artificial Intelligence*, Ulm, DE, 2004. Published in LNCS 3238.

[14] C. J. Van Rijsbergen. *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow, 1979.

[15] A. Westbrook and R. Greene. Using semantic analysis to classify search engine spam. Technical report, Stanford University, 2002.

[16] J. Zobel and A. Moffat. Exploring the similarity space. *SIGIR Forum*, 32(1):18–34, 1998.