Web Document Duplicate Detection Using Fuzzy Hashing

Carlos G. Figuerola, Raquel Gómez Díaz, José L. Alonso Berrocal, and Angel F. Zazo Rodríguez

Abstract. The web is the largest repository of documents available and, for retrieval for various purposes, we must use crawlers to navigate autonomously, to select documents and processing them according to the objectives pursued. However, we can see, even intuitively, that are obtained more or less abundant replications of a significant number of documents. The detection of these duplicates is important because it allows to lighten databases and improve the efficiency of information retrieval engines, but also improve the precision of cybermetric analysis, web mining studies, etc. Hash standard techniques used to detect these duplicates only detect exact duplicates, at the bit level. However, many of the duplicates found in the real world are not exactly alike. For example, we can find web pages with the same content, but with different headers or meta tags, or viewed with style sheets different. A frequent case is that of the same document but in different formats; in these cases we will have completely different documents at binary level. The obvious solution is to compare plain text conversions of all these formats, but these conversions are never identical, because of the different treatments of the converters on various formatting elements (treatment of textual characters, diacritics, spacing, paragraphs ...). In this work we introduce the possibility of using what is known as fuzzy-hashing. The idea is to produce fingerprints of files (or documents, etc..). This way, a comparison between two fingerprints could give us an estimate of the closeness or distance between two files, documents, etc. Based on the concept of ”rolling hash”, the fuzzy hashing has been used successfully in computer security tasks, such as identifying malware, spam, virus scanning, etc. We have added capabilities of fuzzy hashing to a slight crawler and have made several tests in a heterogeneous network domain, consisting of multiple servers with different software, static and dynamic pages, etc.. These tests allowed us to measure similarity thresholds and to obtain useful data about the quantity and distribution of duplicate documents on web servers.

Carlos G. Figuerola · Raquel Gómez Díaz · José L. Alonso Berrocal · Angel F. Zazo Rodríguez
University of Salamanca, C/ Francisco de Vitoria 6-16, 37008 Salamanca, Spain
e-mail: {figue,rgomez,berrocal,zazo}@usal.es

J.M. Corchado et al. (Eds.): Trends in PAAMS, AISC 90, pp. 117–125.
springerlink.com

Keywords: Web Crawling, Fuzzy Hashing, Document Duplicate Detection, Information Retrieval.

1 Introduction

The web is the largest repository of documents available. Even in a purely intuitive way is easy to find abundant information replicated in any tour of the web. Bharat and Broder [2] cite studies of the second half of the 90 in which it is shown that over 30% of web pages explored by crawlers are duplicates. Although in their study identified only 9.4% as duplicate pages, it is also true that they were focused only in duplicates by mirroring. In any case, these are important quantities which raises suspicions of high level of replication of web content for various reasons. Duplication of documents hinders the work of the crawlers and slows down the indexing of documents, is an added difficulty for users who need to review the documents retrieved after a search. Chowdhury and others have noted, moreover, the problem of duplication of documents and their impact not only on the speed of retrieval, but also in the calculation of weights of terms based on their frequencies [4]. In many cases, however, is not exact duplicates, but near duplicates: documents that have virtually the same content, but differ on some details. Yerra and Ng [19] quote some cases, by way of example, of documents in different versions, small documents assembled in a large document, and the reverse: split large documents into smaller parts, different views of the same content ... To this we can add a few more causes: specific headers added by web servers or web applications to identical documents, visitors counters, documents that include the current date, their own URL, relative paths, etc.

Because that, it is important to have techniques and tools to detect duplicate and near duplicate documents, this need arises in various contexts, and which concerns us: that of information retrieval on the web. This article shows some preliminary results in implementing in this type of tasks of techniques known as *fuzzy hashing*. This paper is organized as follows: Section 2 gives a brief review of the most common techniques for measuring the resemblance or similarity between documents, and its application in detecting near duplicate documents. Section 3 describes the *fuzzy hashing* or *Context Triggered Hasshing Piece (CTPH)*. Section 4 describes the experimental tests carried out and discuss its results. Finally we present some conclusions and future work lines.

2 Measuring Similarity between Documents

There are different systems to estimate the resemblance or similarity between two documents. Several of them stem from the calculation of similarity between vectors, apply where the documents are represented by vectors: cosine Dice, Jaccard et al. [16] But they are not useful in this case, as they were designed to estimate semantic similarity, the vector model itself which supporting its use part of the independence

between terms, without taking into account the position relative to each other. And in any case, their application has processing requirements inapplicable in this context [3].

Other systems derived from what is known as edit distance, ie the number of operations required to transform a string in another [14]. This includes measures of similarity between sequences of characters such as Hamming [8], Lenvensthein [12], [15] or Damerau [5], among several others. Their main problem is that they are designed for short sequences of characters, one of its most common uses, for example, is the correction of typographical errors by suggesting words close. Applied to very long strings, such as full papers, are inapplicable because they need too long time.

The most used solution to detect duplicate documents is *hashing*, ie obtaining a criptographyc fingerprint wich, because the algorithms used, is small, but fairly reliably ensures that there are not two equal hashes for documents are different. The hashing has diverse applications, from facilitating quick searches os data structures wich use a hash as a key or reference of the stored data, to use encryption, for example, digital signature applications or as a guarantee of no alteration of a document. Some of the most popular are MD5 or SHA, MD5 is often used in free software downloads for checking the integrity of the original files. It is easy to find utilities in the operating systems and programming languages that allow to obtain hashes by these algorithms.

However, conventional hashing, which works very well for the purposes for which it was conceived, has disadvantages in the detection of duplicate doumentos. In fact, the smallest difference between two documents, even at the bit level, produces hashes completely different, being this difference not proportional to the real difference between such documents. Thus, an algorithm as such MD5 certifies pretty sure that two documents are identical, but is not valid to detect documents that, although containing small differences, could be considered equal in practice.

3 Fuzzy Hashing

The *Fuzzy hashing* o *Context Triggered Piece Hashing (CTPH)* is based in works by A. Tridgell [18], who, after using hash techniques in the development of *rsync* (a popular GNU application for the incremental transfer of files, `http://rsync.samba.org/`) applied *fuzzy hash* in another application (*spamsum*) dedicated to detecting spam e-mail messages.

The *fuzzy hash* is based on what is known as *rolling hash*. Suppose a window of *n* characters in size that moves along a document, each time a character, for each position a conventional hash is calculated, which is recalculated very quickly to the next position of the window until at the end of the document. *Rolling hashing* is applied in the search for substrings, among other things. The *fuzzy hasing* applies a *rolling hash* which advancing along a document until it reachs certain points, known as *trigger points*. When it reaches a *trigger point*, the *rolling hash* computed until

then is stored and restarts, rolling to the next point, and so on until the end. Both the size *n* of the hash rolling window as the *trigger points* are obtained in various ways depending on the size of the document.

At the end of the process we have a number of hashes corresponding to as many segments of the original document, and we may get a short signature from each hash and concatenate them into a sequence of characters [17]. If we change the original document, one or more of the segments (corresponding to the amendment) will produce different *rolling hashes*, but not the rest. Thus, the final sequence of characters obtained will be different in the corresponding segments affected by the amendment, but not the rest. Thus, the number of different characters in the final *fuzzy hash* is proportional to the magnitude of the changes in the document [10].

The *fuzzy hashing* is used in tasks related to computer security (forensics, detection of polymorphic viruses, malware detection, etc.) [9].

4 Tests Carried Out

We conducted several tests designed to evaluate the usefulness of *fuzzy hashing* to detect duplicate documents on the web. The context of these tests is the Information Retrieval on the Web, in this context, a crawler traverses the network by collecting and indexing documents. Basically, the crawler scans a web page, indexes and extracts the links it contains, which are stored. Once indexed a page, the crawler gets from the storage a new URL to explore. Naturally, the crawler is careful not to scan the same URL, ie, does not store addresses that have been visited or already stored previously. Of course, we can also to limit the tours of the crawler to URLs belonging to certains domains or having certain characteristics, as is also possible to confine the indexing to documents in certain formats, etc. [6].

In the tests we used our own crawler, designed to be part of our search engine VII [7]. We have added the ability to detect exact duplicates, using a MD5 hash, and also to detect similar documents using *fuzzy hash*, using *ssdeep* python module [13], a port for this language based on the *ssdeep* library's Kornblum [11]. In all cases, both *fuzzy hash* and MD5 hash are obtained not from the original page or document, but from its conversion to plain text. In this way we ignore differences arising simply the format. On the other hand, as mentioned before, the crawler only visits different URLs, ie, we refer to duplicate or near duplicate documents with different network addresses.

Since these are preliminary tests, we have carried out crawling inside certain web servers or certain network subdomains. Duplicates found, therefore, are always on the same server or in any case, within the same subdomain. After several tests, we established a similarity threshold of 0.9 for the *fuzzy hash*, so that documents with a score above this threshold are considered duplicates. Some results are shown in the accompanying graphs.

The first thing to note is the abundance of duplicate documents in all cases. Many of them are exact duplicates (conversion to plain text), detected with MD5.

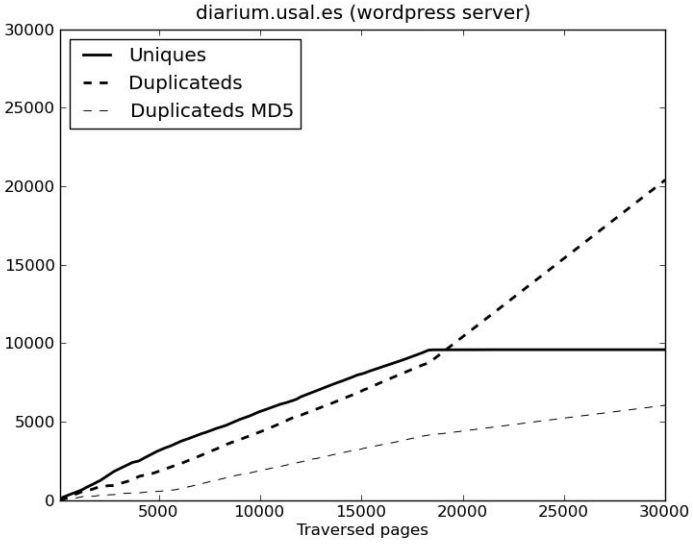

Fig. 1 `diarium.usal.es`. *WordPress* Server

But, in different ways, as the case, there is a significant amount of documents not identified as duplicates by MD5, but thanks to the *fuzzy hashing* can be considered as duplicates for practical purposes. In many cases, the amount of duplicates is even higher than that of unique documents. In some graphs is shown clearly how, as he goes the route of the crawler through a server, the number of unique pages or documents not duplicates grows to a ceiling which is not exceeded or exceeded with little slope. The number of duplicate and near duplicates is also growing as the crawler advances in the exploration of the site, but does not stop and overcomes, sooner or later, to the unique documents.

In other words, the crawler gets all unique documents but does not stop, then continue to get URLs that, being new and different from those already explored, lead to pages or documents that are duplicates of others already visited. This means that quite closely, we may suspend crawler travel long before having explored all the addresses collected, confident that after a certain point, little or no new content page will be covered.

On the other hand, tests have been made also with a full subdomain, the subdomain is heterogeneous in the sense that it consists of multiple servers (about 200) of various sizes, some with static and other with dynamic pages, managed by a variety of *CMS*s and / or ad-hoc Web applications. The size, number of pages, in that subdomain is difficult to determine, Google gives a figure of 1,500,000 pages, although it is known that the figures provided by the search engines must be observed with caution [1]. What is striking is that, after exploring the first nearly 73.000 pages the

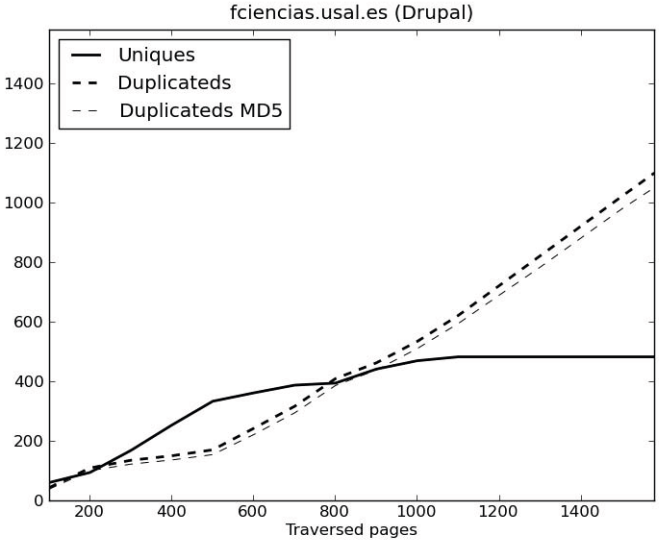

Fig. 2 `fciencias.usal.es`. *Drupal*

crawler don't find unique documents and, of these, only about 46,000 are uniques. The amount of exact duplicate documents (detected by MD5) is remarkable, but also stabilizes after a certain point. Which increases from that point the amount of duplicate documents are near duplicates detected by *fuzzy hash*.

At this point several questions may arise. First, the causes of the extensive duplication of documents. Taking into account that the crawler only follows unique URLs, some of these duplications could come from redirects and aliases in the server names. But, waiting to confirm this hypothesis with hard data, it appears that this cause could only explain part of exact duplication. Another part may stem from how CMSs and web applications generate links to their components, which could take different forms in regard to the URL, but point to the same document. In this regard it is interesting to ask if there are different behaviors depending on the particular software used in each site. We can assume that, perhaps, a CMS generates more duplicates than others. It is also likely that certain web applications generate a greater number of replications, either exact or either generating versions of the same document. This may occur with social-oriented web sites (web 2.0) and is also likely that this class of sites have a greater relationship with certains CMSs, aimed precisely at the Web 2.0.

The way the crawler traverses the site has to do with finding more or less duplicates? Can we find paths to allow us explore before the unique documents contained in a web site or domain?

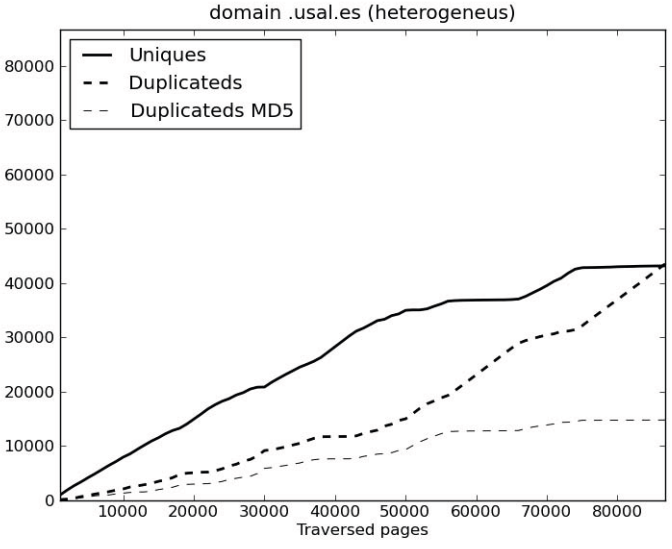

Fig. 3 Domain `usal.es`. Several servers and heterogeneus web applications

5 Conclussions and Future Work

We have shown a technique known as *fuzzy hash* and its use in detecting near duplicates in the web when a crawler navigates it. *Fuzzy hashing* is a real hash, it is computed quickly and the risk of collisions (two different documents produce the same hash) is very low. However, the difference between two *fuzzy hashes* is proportional to the difference between the documents from they came. This feature allow us to detect near dupplicated documents easily. Preliminary tests carried out show that there is a significative amount of duplicates or near duplicates documents. The results suggest that, for a given level of crossed pages within a website, there comes a point where they no longer reach new pages, which have not been explored previously.

For future work, various questions must be resolved. First, it is necessary to analyze the causes of the large number of duplicate documents. Also to consider whether certain CMS produce more duplicate than others. Also, find ways of improving crawling strategies, so that the useful pages can be explored before.

References

1. Bar-Ilan, J.: Expectations versus reality – search engine features needed for web research at mid 2005. Cybermetrics 9(1) (2005), `http://www.cindoc.csic.es/cybermetrics/articles/v9i1p2.html`

2. Bharat, K., Broder, A.: Mirror, mirror on the web: A study of host pairs with replicated content. Computer Networks 31(11-16), 1579–1590 (1999), http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.90.1488&rep=rep1&type=pdf
3. Chowdhury, A.: Duplicate data detection (2004), retrieved from http://ir.iit.edu/~abdur/Research/Duplicate.html, http://gogamza.mireene.co.kr/wp-content/uploads/1/XbsrPeUgh6.pdf
4. Chowdhury, A., Frieder, O., Grossman, D., McCabe, M.: Collection statistics for fast duplicate document detection. ACM Transactions on Information Systems (TOIS) 20(2), 171–191 (2002), http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.5.3673&rep=rep1&type=pdf
5. Damerau, F.: A technique for computer detection and correction of spelling errors. Communications of the ACM 7(3), 171–176 (1964), http://www.cis.uni-muenchen.de/ heller/SuchMasch/apcadg/literatur/data/damerau_distance.pdf
6. Figuerola, C.G., Alonso Berrocal, J.L., Zazo Rodríguez, Á.F.: Rodríguez Vázquez de Aldana, E.: Diseño de spiders. Tech. Rep. DPTOIA-IT-2006-002 (2006)
7. Figuerola, C.G., Gómez Díaz, R., Alonso Berrocal, J.L., Zazo Rodríguez, A.F.: Proyecto 7: un motor de recuperación web colaborativo. Scire. Representación y Organización del Conocimiento 16, 53–60 (2010)
8. Hamming, R.: Error detecting and error correcting codes. Bell System Technical Journal 29(2), 147–160 (1950), http://www.lee.eng.uerj.br/~gil/redesII/hamming.pdf
9. Kornblum, J.: Identifying almost identical files using context triggered piecewise hashing. digital investigation 3, 91–97 (2006), https://www.dfrws.org/2006/proceedings/12-Kornblum.pdf
10. Kornblum, J.: Beyond fuzzy hash. In: US Digital Forensic and Incident Response Summit (2010), http://computer-forensics.sans.org/community/summits/2010/files/19-beyond-fuzzy-hashing-kornblum.pdf
11. Kornblum, J.: Fuzzy hashing and sseep (2010), http://ssdeep.sourceforge.net/
12. Levenshtein, V.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10(8), 707–710 (1966)
13. Milenko, D.: ssdeep 2.5. python wrapper for ssdeep library (2010), http://pypi.python.org/pypi/ssdeep
14. Navarro, G.: A guided tour to approximate string matching. ACM computing surveys (CSUR) 33(1), 31–88 (2001), http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.7225&rep=rep1&type=pdf
15. Soukoreff, R., MacKenzie, I.: Measuring errors in text entry tasks: an application of the levenshtein string distance statistic. In: CHI 2001 Extended Abstracts on Human Factors in Computing Systems, pp. 319–320. ACM, New York (2001), http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.22.757&rep=rep1&type=pdf

16. Tan, P., Steinbach, M., Kumar, V., et al.: Introduction to data mining. Pearson Addison Wesley, Boston (2006),
http://www.pphust.cn/uploadfiles/200912/20091204204805761.pdf
17. Tridgell, A.: Spamsum overview and code (2002),
http://samba.org/ftp/unpacked/junkcode/spamsum
18. Tridgell, A., Mackerras, P.: The rsync algorithm (2004),
http://dspace-prod1.anu.edu.au/bitstream/1885/40765/2/TR-CS-96-05.pdf
19. Yerra, R., Ng, Y.: Detecting similar html documents using a fuzzy set information retrieval approach. In: 2005 IEEE International Conference on Granular Computing, vol. 2, pp. 693–699. IEEE, Los Alamitos (2005),
http://faculty.cs.byu.edu/~dennis/papers/ieee-grc.ps