SPLAT: A SYSTEM FOR SELF-PLAGIARISM DETECTION

Christian Collberg, Stephen Kobourov, Joshua Louie, and Thomas Slattery
Department of Computer Science
University of Arizona
Tucson, AZ 85721
{collberg,kobourov,jdlouie,thomass}@cs.arizona.edu

ABSTRACT

This paper presents a system for self-plagiarism detection, SPLAT. The system uses a WebL web spider that crawls through the web sites of the top fifty Computer Science departments, downloading research papers and grouping them by author. Next a text-comparison algorithm is used to search for instances of textual reuse. Instances of potential self-plagiarism for each author are reported in an HTML document so that they can be considered in more detail, in order to determine if they are truly self-plagiarized papers. The system discovered a number of pairs of papers of questionable originality.

KEYWORDS

Self-plagiarism, web-spider, text-comparison.

1. INTRODUCTION

We are all too aware of the ravages of scientific misconduct in the academic community. Students submit assignments from the their friends who took the course the year before, on-line paper-mills allow students to browse for term-papers on popular topics, and occasionally researchers are found out when falsifying data or publishing the work of others as their own. Another, rarely discussed form of scientific misconduct is self-plagiarism. Self-plagiarism is the use of ones own previously published materials in the creation of a new published material without crediting the previous paper as a source. Sometimes the derived paper consists of bits and pieces from previous writings, and occasionally, the derived paper is identical to an earlier one, except for a different title and/or formatting.

While plagiarism, falsification of data, and other forms of misconduct have been the topic of many publications and research papers, self-plagiarism has received little or no attention. The legal and ethical implications of self-plagiarism have been the topic of a handful of papers [Samuelson, P., 1994; Loui, M., 2002; Bird, S., 2002]. However, self-plagiarism seems to be much more prevalent than other forms of scientific misconduct. The impact of self-plagiarism is more subtle but no less detrimental to the research community. Self-plagiarism allows for a bloated number of research papers to be produced without doing additional work to create new papers. As a result, fundamentally identical papers can be created and passed off to different publishing venues, all for the purpose of increasing the academic recognition of the researcher. However, such practices do not benefit the research society as a whole, in that many more papers are produced, with less new and exciting material to spur on new ideas. Instead the pool of papers becomes cluttered with papers on the same topics, but with different names. Moreover, self-plagiarism can prevent worthy and novel ideas from getting published, and give the general public the idea that their research dollars are spent on rehashing old results rather than on original research.

In this paper we describe an automated system for self-plagiarism detection, SPLAT. While our results do not represent a scientific study of self-plagiarism, they do confirm that textual reuse does occur even at the best academic institutions. An overview of our system is shown in Figure 1. The system consists of a web spider that traverses the top fifty computer science departments and finds the faculty homepages. For each

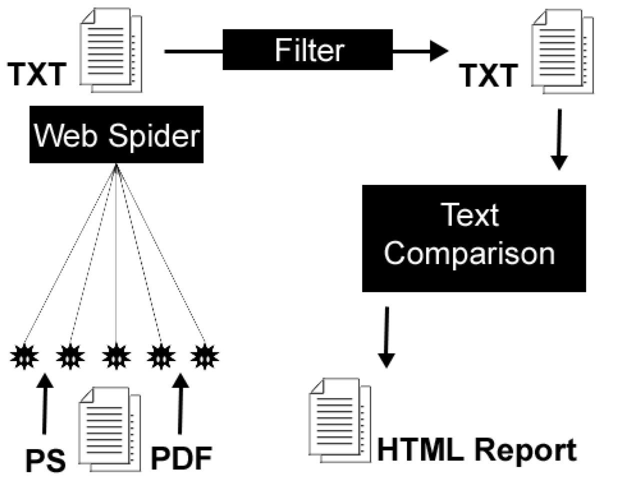


Figure 1: Overview of the SPALT system.

faculty member, we download all papers available on the homepage. After converting them to text, we run a text analysis program to check for self-plagiarism and report pairs of papers with high textual overlap. The results of the comparisons are summarized in a HTML report that color-codes co-occurring paragraphs. The final step consists of checking by hand whether a highly overlapping pair contains a self-plagiarized paper. The last step is necessary as many similar pairs of paper consist of a conference and journal version of the same paper, or technical report and a conference paper, etc. SPLAT can be downloaded from `splat.cs.arizona.edu.`

In Section 2 we consider related work. Details about our spider are in Section 3 and details about the text comparison algorithm are in Section 4. In Section 5 we summarize the type of textual reuse that was discovered by our system. In Section 6 we briefly describe future work on extending our system to a tool that can be used by conference and journal reviewers. In Section 7 we summarize the results presented in this paper.

2. RELATED WORK

CORA, a Computer Science Research Paper Search Engine [Rennie, J. et al, 1999], most closely resembles the type of spider that we are using. CORA makes use of smart spiders to crawl computer science web sites and record the papers located within. The major difference between that spider and the one we are using is that our spider is designed to dynamically search for one author's work at a time, and not crawl the entire site beforehand to get all the information first.

The Stanford Copy Analysis Mechanism (SCAM) [Shrivakumar, N. et al, 1995] is a comparison utility for detecting identical documents or documents with a high degree of overlap. SCAM uses a registration server to which original documents can be registered by their authors. Attempts to register illegal copies of already registered documents can be detected. Additionally, web crawlers can be used to search for documents and compare them against registered documents in a manner somewhat similar to our system.

Following up on an article about self-plagiarism in computer science [Samuelson, P., 1994], Collberg and Kobourov revisit the discussion with an effort to define different forms of textual reuse [Collberg, C. et al, 2003]. In this paper personal anecdotes and an informal survey of computer science academics are used to argue that while self-plagiarism is not uncommon, there seems to be little agreement on where the borderline is between acceptable forms of re-publication and self-plagiarism.

3. THE WEB SPIDER

The first component of the SPLAT system is the web spider that crawls the web collecting research papers. The programming language WebL [Kistler, T. et al, 1998] was used in the development of the web spider. WebL's is a language and system is designed for rapid prototyping of Web computations. It is well suited for the automation of tasks on the World Wide Web. It has modules to enable the programmer to quickly develop a web spider. However, a web spider that successfully locates authors' pages and downloads the papers has to be constrained heavily to avoid several major problems that arise when looking for papers:

- **Remaining on the author's page:** When examining a page, what is an acceptable page to go to and what is not?
- **Research papers only:** When preparing to download a file, it there anyway to know if it is a research paper or not?
- **Seemingly infinite link graph:** When traversing links on a single author's web site, when should the spider give up?
- **Slow downloads:** If downloading a 30Mb paper, with a transfer rate of less than 1Kb, should the spider bother downloading it?

Ideally, a web spider would start at the homepage of the author and traverse all links downloading all the research papers for that author. If seen as graph problem, this would be feasible if and only if the graph had one source (the author's home page) and no edges going outside of his page. In the real web, an author's homepage has many links that extend to other pages ranging from the classes he teaches to pages of interest all over the web. Attempting to try all links would result in the spider going far away from its target area and require a great deal of time to process one author. Along the way, the spider would download hoards of irrelevant papers.

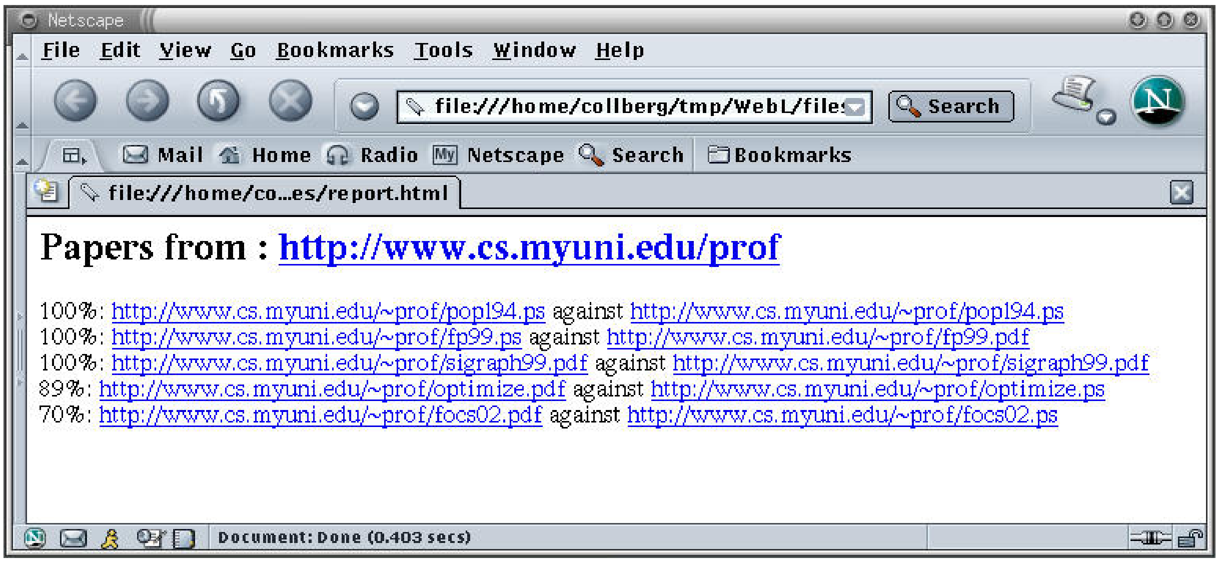
Several constraints were placed to limit where the spider could go. The first was to check for links containing the words *publication*, *paper*, or *research.* These links would most likely have the desired materials. If no links could be found with any of the key words, then searching would be done in a breadth-first search through all of the author's links. Another limitation put on the spider was that the only links that could be traversed were ones that existed on the same main site. That is, if the homepage were at `http://cs.myuni.edu/~author`, the spider would only check sites whose links contained the name `myuni`. Attempting to further constrain this to require the link to contain `cs.university.edu` or `~author` lead to many missed papers, since some authors work in multiple departments, and others do not place all their pages hierarchically under their initial home page.

Most of the papers located on the web are in standard formats such as, `.pdf`, `.ps`, and `.doc`. Instead of attempting to determine if a paper is a research paper or not, the spider simply downloads them all. After downloading all the files and converting them to text, a filter program is run to remove all files that did not convert properly and those that are not research papers. The rule used to determine whether a document is a research paper is that the document must contain an abstract, or introduction, and also references or bibliography sections. This process was used in CORA to find research papers, and experiments showed that it had roughly a 95% accuracy rate.

One author's website may contain many levels of pages, which results in hundreds and possibly thousands of checks and page loads in search of his papers. To deal with this problem, we use a timer that records when a particular site was first checked. Each time the spider gets ready to download a paper, search a new page or visit another page, it checks the time. If the time exceeds a predetermined limit, it stops downloading, instead finding new pages and visiting them. We attempt to deal with problem of downloading a large paper over a slow connection by using a collection of threads doing the crawl. Thus, with a few threads, if one gets stuck on a big file being downloaded on a slow connection, the others can continue searching for additional papers.

4. TEXT COMPARISON

The text comparison utility, which was implemented in Java, is an independent module from the web spider. After the web spider has finished downloading and converting to text all of the papers for a particular author, the text comparison utility is executed on the directory containing those papers. The utility performs pair-wise comparison of all papers in the directory. When finished, it produces an HTML report file in the same directory. The report file lists in descending order all pairs of files with their respective percentages (above an adjustable threshold) of detected similarity. See Figure 2.


Netscape
File Edit View Go Bookmarks Tools Window Help
file:///home/collberg/tmp/WebL/files
Search
Mail Home Radio Netscape Search Bookmarks
file:///home/co...es/report.html

Papers from : http://www.cs.myuni.edu/prof

100%: http://www.cs.myuni.edu/~prof/popl94.ps against http://www.cs.myuni.edu/~prof/popl94.ps
100%: http://www.cs.myuni.edu/~prof/fp99.ps against http://www.cs.myuni.edu/~prof/fp99.pdf
100%: http://www.cs.myuni.edu/~prof/sigraph99.pdf against http://www.cs.myuni.edu/~prof/sigraph99.pdf
89%: http://www.cs.myuni.edu/~prof/optimize.pdf against http://www.cs.myuni.edu/~prof/optimize.ps
70%: http://www.cs.myuni.edu/~prof/focs02.pdf against http://www.cs.myuni.edu/~prof/focs02.ps

Document: Done (0.403 secs)

Figure 2: Reports are presented in a standard browser.

Formalizing self-plagiarism is a non-trivial task. We decided to focus on a particular form of self-plagiarism that is easy to detect: reusing large blocks of text from previously published work. With this in mind, we would like the comparison algorithm to deal appropriately with the following problems:

- **Cosmetic changes:** Minor cosmetic changes to text, such as the addition or removal of punctuation or spacing should not affect the comparison.
- **Reordered text:** Paragraphs or sentences from paper A can be copied but placed in a different order in paper A'. If the bulk of the content is the same, however, this should still be detected.
- **Reworded text:** Rewording of text without significant change to the meaning, such as the substitution of a few words for synonyms or swapping clauses of a sentence, should be caught. While not nearly as severe as directly copying text, a significant amount of this should still register as plagiarism.

In addition to worrying about what sorts of similarities amount to plagiarism, there are also an almost endless number of criteria upon which two texts can be compared to detect it. The primary goal of our system is accuracy in detecting instances of plagiarism, but a certain degree of efficiency is necessary as well. Papers in plaintext format in excess of 200kb are not uncommon, but even the pairwise comparison of 50+ average-sized documents ($\approx$50kb) is a very lengthy $O(n^2)$ operation. More complex algorithms, such as attempting to align the documents, were considered. However, the highly likely possibility that the text will be reordered renders alignment a relatively poor choice for a primary algorithm. Since plagiarized text from one document could appear anywhere and in any order in a second document, a brute-force comparison algorithm seemed to be the best choice. A great deal of optimization is possible to improve the speed of the algorithm, but a certain degree of brute force is necessary to find all occurrences of textual reuse. The final algorithm consists of three main parts:

1. parsing the text documents into paragraphs and sentences in a canonical form;
2. performing a highly optimized, brute-force, pairwise comparison of the parsed documents;
3. producing an HTML report of the results.

4. 1 Parsing into Canonical Form

In the first phase, each text file is parsed into a Document object with a list of Paragraph objects, each of which then has a list of Sentence objects. The text file is expected to contain the URL of the original file on the first line, which is extracted first, followed by a separate paragraph on each line. Each line after the first then becomes a Paragraph. Sentences are considered to be anything delimited by the characters `! . ? ;`. Paragraphs with less than four sentences are discarded, since they are most likely just section headings, parts of formulas, or other random and insignificant text. If a Paragraph has at least four sentences, the resulting text for the Sentences is converted to lowercase and then parsed into words. Words are considered to be anything delimited by whitespace or any of the following characters: `! . ? " \ < > : ; [ ] { } ( ) /`.

Any whitespace or delimiting characters are removed and only the list of words is retained. Words from a short, pre-defined list deemed insignificant to the meaning of a sentence (such as a, an, the, this, and that) are discarded. Similar to Paragraphs, any Sentence with less than four words (after removing insignificant ones) is discarded. Most “sentences” with less than four meaningful words are not actual sentences–those that happen to be real sentences are too short to hold much content and thus are not of much interest for comparison. Rather than storing a large numbers of small strings, the words are represented as integers. A global hash of words to their corresponding integer values is maintained across the program. As a sentence is parsed, each word is looked up in the table. If it already exists, that value is used. Otherwise, the word is inserted into the table with a new, unique value. Each Sentence then maintains a list of its unique words (as integers in sorted order); the sum of the integer values of all its words; and the original sentence, as a string in canonical form with a space between each word.

4. 2 Comparison Algorithm

After all documents have been parsed, all pairs of documents are “scored" for the level of similarities found between them. All pairs of paragraphs in the two documents are scored against each other by a pairwise comparison of their respective sentences. The results then filter up, with paragraphs earning points based upon the number of similar sentences and their levels of similarity and the document earning points based upon the amount of detected plagiarism in each paragraph with a total score above a certain threshold.

Sentences are compared for similarity on two levels. Sentences that are identical earn the maximum score possible; sentences that are highly similar to one another earn a score somewhere between 50-100% of the possible score, depending on the amount of similarity. Comparing sentences for equality is easy and highly optimized. Before even looking at the words in a sentence, the "sums" of the two sentences–calculated while parsing–are compared. If they are not the same, it is known immediately that the sentences are not identical. While occasional overlap of the values of these sums does occur between sentences that are different, it is rare enough to eliminate almost all unnecessary comparisons. Only if the sums and word counts of two sentences are identical are the actual strings compared.

Sentences are considered similar if the intersection of their sets of unique words is the same size or only slightly smaller than the sets themselves. Since the lists of unique words are maintained in sorted order, binary searching can be used to efficiently calculate the size of the intersection of the sets. This comparison is performed only when significant similarities exist. Sentences that have a major discrepancy in the sizes of their unique word sets are ignored, as are any sentences that have very small sets of unique words.

4. 3 Reporting Results

After pairs of papers have been scored against each other, they are given a percentage to represent the approximate level of plagiarism found between the two. The score for a document represents roughly the number of sentences worth of textual reuse found in the document. Once all pairs of papers have been compared, the list of papers and their respective percentages is sorted in descending order and the results are written out to an HTML file in the directory. The top of the file gives the URL for the root page from which the papers were downloaded. Following that is the list of percentages with links to the corresponding papers in their original format. In addition, an HTML page with color-coded matches between the two papers under consideration is created; see Figure 3.

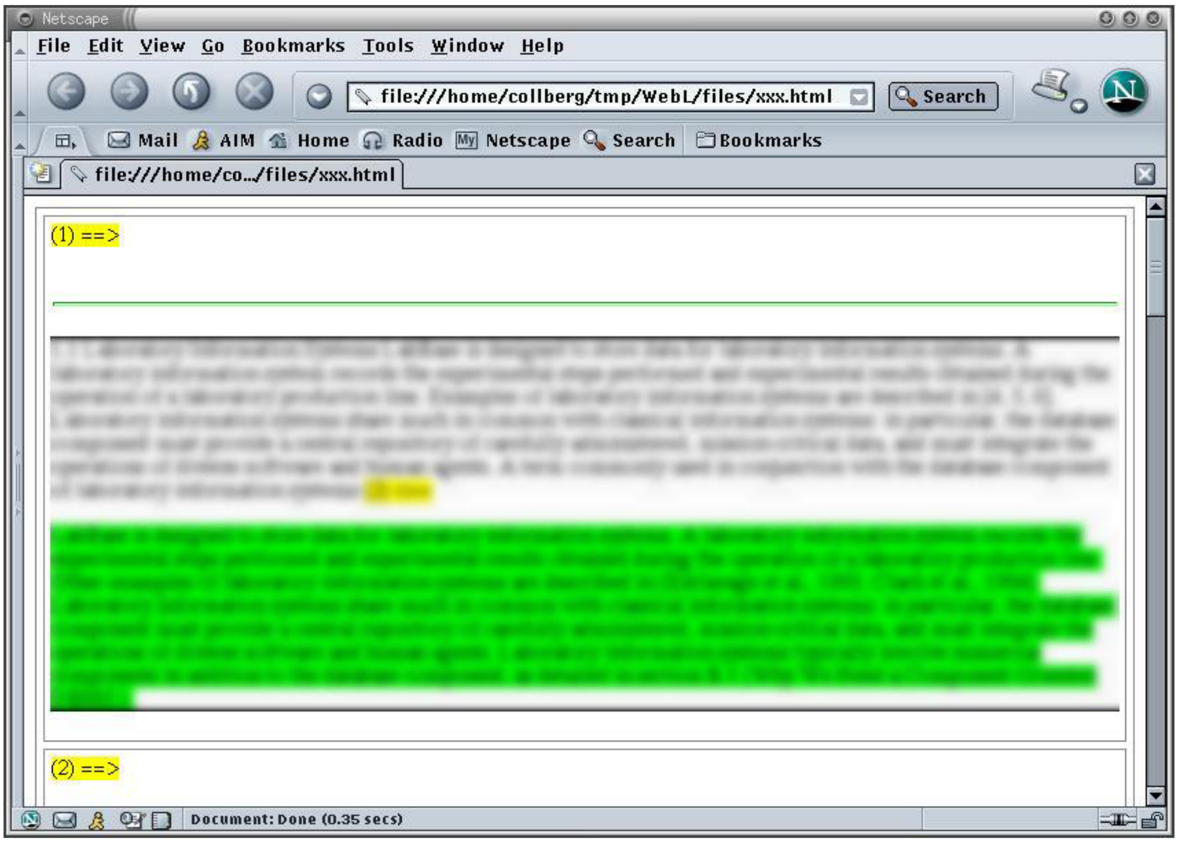

Netscape
File Edit View Go Bookmarks Tools Window Help
file:///home/collberg/tmp/WebL/files/xxx.html
Search
Mail AIM Home Radio Netscape Search Bookmarks
file:///home/co.../files/xxx.html
(1) ==>
(2) ==>
Document: Done (0.35 secs)

Figure 3: The system allows for two papers to be examined in more detail. Similar paragraphs are color coded and presented in a standard browser.

5. RESULTS OF THE STUDY

It is difficult to know how common self-plagiarism is. Anecdotally, we hear of colleagues who publish the same result with minor modifications over and over again, and occasionally we come across a paper whose content is too close to previous publications. Nonetheless, our study did reveal numerous instances of textual reuse. Some highly correlated pairs of papers represented what is generally thought of as acceptable forms of re-publication: technical reports published in conferences, conference articles recast as journal papers, etc. These were all weeded out manually. However, we found a number of instances of papers with questionable originality. In particular, we ran across cases such as:

- pairs of conference publications with common introduction and/or related work sections that do not reference each other;
- pairs of conference publications with over 50% common text that do not reference each other;
- pairs of nearly identical conference and journal versions of the same paper, where the journal version does not reference the conference version;
- pairs of nearly identical conference and journal versions of the same paper, with different titles, where the journal version does not reference the conference version.

This is in many ways an unsatisfactory study. We only looked at papers downloaded from the web pages of the authors and we did not attempt to collect all papers by a given author (for example, by using services such as citeseer.nj.nec.com). Authors who deliberately engage in self-plagiarism are unlikely to put both the original and the derived work on their website. However, our study confirms that textual reuse does occur. A discussion about what may or may not constitute self-plagiarism can be found in a related paper [Collberg, C. et al, 2003].

6. FUTURE WORK

The spider will be more accurate if the entire computer science site is crawled. All papers would be downloaded and converted. Those would be filtered, parsed and sorted into the appropriate author's directories. The spider would implement a better schema such as reinforcement learning, with fewer time and traversal constraints. At its current state, a number of valid papers are missed as well as a number of invalid papers gathered.

There is a great deal that could still be added to the comparison utility. It currently offers a good balance of accuracy and efficiency, although more optimizations could be made to increase the speed. The greatest improvement, though, would be to allow the criteria for comparison to be adjusted, so that the user could make the decision between speed and an even higher level of accuracy. There are an almost limitless number of statistics that could be examined to more accurately–or quickly–spot possible plagiarism. Some interesting criteria could include:

- the number of words that occur very infrequently in each document, yet occur in both, and
- the percentage of overlap between the unique word sets for both complete documents.

Another improvement to the comparison utility would be to allow more thorough comparison and scoring across all documents, rather than simple pairwise comparison. It is probably of greater value, for instance, to see the total amount of plagiarism in a document from five others than to see five separate comparisons.

Finally, we are planning to extend our system so that it acts as a reviewer's workbench. The program would compare a paper under review (for a conference or journal) to a record of the author's previously published articles extracted from their web site and online article repositories (such as the ACM and IEEE digital libraries and CiteSeer's database).

7. CONCLUSION

We have presented SPLAT, a system for detecting self-plagiarism. We described the web spider and the text comparison utility currently used in the system and we also outlined the potential of our system as a reviewer's workbench.

ACKNOWLEDGEMENTS

Christian Collberg was partially supported by the NSF under grant CCR-0073483. Stephen G. Kobourov was partially supported by the NSF under grant ACR-0222920. Joshua Louie and Thomas Slattery are in the undergraduate program in the Department of Computer Science at the University of Arizona.

REFERENCES

Bird, S., 2002. Self-plagiarism and dual and redundant publications: What is the problem? Commentary on "Seven ways to plagiarize: Handling real allegations of research misconduct", *Science and Engineering Ethics*, vol. 8, n. 4.

Collberg, C. and Kobourov, S., 2003. Self-plagiarism in computer science. Techincal Report TR03-03, University of Arizona, February 2003.

Kistler, T. et al, 1998. WebL – A programming language for the web. In *Proceedings of WWW7*, pages 259–270. Elsevier.

Loui, M., 2002. Seven ways to plagiarize: Handling real allegations of research misconduct, *Science and Engineering Ethics*, vol. 8, n. 4.

Rennie, J. et al, 1999. Using reinforcement learning to spider the Web efficiently. In *Proceedings of ICML-99, 16th International Conference on Machine Learning*, pages 335–343, Bled, SL, 1999. Morgan Kaufmann Publishers, San Francisco, US.

Samuelson, P., 1994. Self-Plagiarism or fair use. *Communications of the ACM*, vol. 37, n. 2, p. 21-25.

Shivakumar N. et al, 1995. SCAM: A copy detection mechanism for digital documents. In *Proceedings of the Second Annual Conference on the Theory and Practice of Digital Libraries*.