Copy Detection Mechanisms for Digital Documents *

Sergey Brin, James Davis, Hector Garcia-Molina
Department of Computer Science
Stanford University
Stanford, CA 94305-2140
e-mail: sergey@cs.stanford.edu

October 31, 1994

Abstract

In a digital library system, documents are available in digital form and therefore are more easily copied and their copyrights are more easily violated. This is a very serious problem, as it discourages owners of valuable information from sharing it with authorized users. There are two main philosophies for addressing this problem: prevention and detection. The former actually makes unauthorized use of documents difficult or impossible while the latter makes it easier to discover such activity.

In this paper we propose a system for registering documents and then detecting copies, either complete copies or partial copies. We describe algorithms for such detection, and metrics required for evaluating detection mechanisms (covering accuracy, efficiency, and security). We also describe a working prototype, called COPS, describe implementation issues, and present experimental results that suggest the proper settings for copy detection parameters.

1 Introduction

Digital libraries are a concrete possibility today because of many technological advances in areas such as storage and processor technology, networks, database systems, scanning systems, and user interfaces. In many aspects, building a digital library today is just a matter of "doing it." However, there is a real danger that such a digital library will either have relatively few documents of interest, or will be a patchwork of isolated systems that provide very restricted access.

The reason for this danger is that the electronic medium makes it much easier to illegally copy and distribute information. If an information provider gives a document to a customer, the customer can easily distribute it on a large mailing list or can post it on a bulletin board. The danger of illegal copies is not new, of course; however, it is much more time consuming to reproduce and distribute paper, CDs or videotape copies than on-line documents.

Current technology does not strike a good balance between protecting the owners of intellectual property and giving access to those who need the information. At one extreme are the open sources on the Internet, where everything is free, but valuable information is frequently unavailable because of the dangers of unauthorized distribution. [1] At the other extreme are closed systems, such as the

*This research was sponsored by the Advanced Research Projects Agency (ARPA) of the Department of Defense under Grant No. MDA972-92-J-1029 with the Corporation for National Research Initiatives (CNRI). The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsement, either expressed or implied, of ARPA, the U. S. Government or CNRI.

[1] As just one example, Knight-Ridder Tribune recently (June 23, 1994) ceased publishing on ClariNet the Dave Barry and the Mike Royko columns because subscribers re-distributed the articles on large mailing lists.

one that the IEEE currently uses to distribute is papers in CD-ROM. This a completely stand-alone system where users can look for specific articles, view them, and print them, but cannot move any data in electronic form out of the system, and cannot add any of his or her data.

Clearly, one would like to have an infrastructure that gives users access to a wide variety of digital libraries and information sources, but that at the same time gives information providers good economic incentives for offering their information. In many ways, we believe this is *the* central issue for future digital information and library systems.

In this paper we present one component of the information infrastructure that addresses this issue. The key idea is quite simple: provide a *copy detection service* where original documents can be registered, and copies can be detected. The service will detect not just exact copies, but also documents that overlap is significant ways. The service can be used (see Section 2) in a variety of ways by information providers and communications agents to detect violations of intellectual property laws. Although the copy detection idea is simple, there are several challenging issues we address here involving performance, storage capacity, and accuracy that need to be resolved. Furthermore, copy detection is relevant to the "database community" since its central component is a large database of registered documents.

We stress that copy detection is not the complete solution by any means; it is simply a helpful tool. There are a number of other important "tools" that will also assist in safeguarding intellectual property. For example, good encryption and authorization mechanisms are needed in some cases. It is also important to have mechanisms for charging for access to information. The articles in [5, 7, 9] discuss a variety of other topics related to intellectual property. These other tools and topics will not be covered in this paper.

In the following section we will briefly discuss some of the options for safeguarding intellectual property, and will argue that copy detection is a very promising approach. In Section 3 we define the basic terms and evaluation metrics for copy detection. Then in Section 5 we describe our working prototype, COPS, and report on some initial experiments. A sampling technique that can reduce the storage space of registered documents or can speed up checking time is presented and analyzed in Section 6. Finally, some security considerations are discussed in Section 3.3.

2 Safeguarding intellectual property

How can we ensure that a document is only seen and used by a person who is authorized (e.g., has paid) to see it? Let us illustrate the possibilities and the problems by suggesting two particular techniques.

The first technique is based on the notion of a *secure printer*. Such a printer is sealed and cannot be opened by its owner. It contains a public key encryption device [11], where the private key is unique to this printer and is only known to the printer itself. The printer's public key and the name of the owner are registered in a database provided by the trusted printer manufacturer. When the owner requests a document from an information vendor, the vendor first ensures the owner is authorized (e.g., has paid), then it fetches the public key of the printer from the registry, it encrypts the document using the public key and sends the result. When the owner receives the data, he can send it to the printer which can then decrypt and print the document. However, the electronic data cannot be used for anything else, as only this one printer can decrypt it. The data can be resent to the printer to create another paper copy, so the document can be reproduced in this way. However, illegally reproduced paper copies is a "previously unsolved problem" that this scheme does not address.

The main problem with this scheme is that it is too restrictive. It is more of an "electronic paper delivery system" than anything else. Users cannot browse through documents before buying, and cannot use parts of the document in others, e.g., for quotes. Furthermore, it requires special purpose hardware. However, it may still be useful in conjunction with other schemes. For example, perhaps users can be allowed to browse through low-resolution copies of documents, or through documents that have key components missing. Once the user decides he wants to read the document, he can purchase the a high quality copy that can be delivered via the secure printer. The scheme can also be adapted for a "secure computer" instead of a printer.

The second technique we wish to illustrate is that of an *active document* (suggested in [6]). The idea is that an information vendor does not send out a documents; instead it sends out programs that can generate documents. When a user receives one of these programs, call it P, he can run it on his local machine. Embedded within P and its data structures is the encrypted document; as P runs, it displays the document. However, before or during display, P sends a message to the vendor, informing it that it is being run, and waits for a response. This way, the vendor can charge each time P runs, or can limit the number of times P runs.

This scheme also has its drawbacks. A user cannot read the document through his favorite viewer. The vendor must know the architecture of the user's machine in advance, to generate appropriate code. The user cannot see the document if the vendor's machine is unavailable on the network. Finally, the scheme is not bullet proof, since the user could run P in an software emulator of his machine that could record the characters of the document as they are displayed.

While we have only given two examples, we believe that they illustrate a common problem with document *protection* techniques: they are often cumbersome and usually get in the way of users. The alternative is to use *detection* techniques. That is, we assume most users are honest, allow them access to the documents, and focus on detecting those that violate the rules. Many software vendors have found this approach to be superior (protection mechanisms get in the way of honest users, and sales may actually decrease).

One possible direction is to incorporate a "watermark" into a document that identifies its origin [12, 3, 4, 2]. For example, if we think of the documents as images, we may encode the watermark into a small number of random bits throughout the image. The users would be unaware of where the watermark bits were, but the information vendor that originally provided the document could extract them to determine who the document was sold to originally. If the document is possessed by a different person or organization, then a violation is detected. The main weakness of approaches such as these is that users may destroy the watermark by processing the document. For instance, passing the (image) document through a noise filter or lossy compression algorithm could easily change enough bits (without really altering the image) to destroy the watermark.

A second approach, and one that we advocate in this paper (for text documents), is that of a *copy detection server* [1, 10]. The basic idea is as follows: When an author creates a new work, he or she registers it at the server. The server could also be the repository for a copyright recordation and registration system, as suggested in [8]. As documents are registered, they are broken into small units, for now say sentences. Each sentence is hashed and a pointer to it is stored in a large hash table.

Documents can be compared to existing documents in the repository, to check for plagiarism or other types of significant overlap. When a document is to be checked, it is also broken into sentences. For each sentence, we probe the hash table to see if that particular sentence has been seen before. If the document and a previously registered document share more than some *threshold* number of sentences, then a violation is flagged. The threshold can be set depending on the desired checks, smaller if we are looking for copied paragraphs, larger if we only want to check if documents share large portions. A human would then have to examine both documents to see if it was truly a violation.

Unlike the case with watermarks, it is not easy for a user to automatically subvert the system, i.e., to make an undetectable copy. For example, if the decomposition units are sentences, a user would have to change a large number of sentences in the document. This involves more than just adding a blank space between words (assuming that the hashing scheme ignores spaces). Of course, a determined user could change all sentences, but our goal is to make it hard to copy documents, not to make it impossible. This makes it hard to rapidly distribute copies of documents.

The copy detection server can be used in a variety of ways. For example, a publisher is legally liable for publishing materials the author does not have copyright on; thus, it may wish to check if a soon-to-be-published document is actually an original document. Similarly, bulletin-board software may automatically check new postings in this fashion. An electronic mail gateway may also check the messages that go through (checking for "transportation of stolen goods"). Program committee members may check if a submission overlaps too much with an author's previous paper. Lawyers may want to check subpoenaed documents to prove illegal behavior. (Copy detection can also be used for computer programs [10], but we only focus on text in this paper.) The are also applications that do not involve detection of undesirable behavior. For example, a user that is retrieving documents from an information retrieval system or who is reading electronic mail, may want to flag duplicate items (with a given overlap threshold). Here the "registered" documents are those that have been seen already; the "copies" represent messages that are retransmitted or forwarded many times, different editions or versions of the same work, and so on. Of course, potential duplicates should not be deleted automatically; it is up to the user to decide if he wants to view possible duplicates.

In summary, we think that detecting copies of text documents is a fundamental problem for distributed information or database systems. And there are many issues that need to be addressed. For instance, should the decomposition units be paragraphs or something else instead of sentences? Should we take into account order of the units (paragraphs or sentences), e.g., by hashing sequences of units? Is it feasible to only hash a fraction of the sentences of registered documents? This would make the hash table smaller, hopefully still making it very likely that we will catch major violations. If the hash table is relatively small, it can be cloned. Our mail gateway above could then perform its checks locally, instead of having to contact a remote copy detection server for each message. There are also implementation issues that need to be addressed. For example, how are sentences extracted from say Latex or Word documents? Can one extract them from Postscript documents, or from bit maps via OCR?

These and other questions will be addressed in the rest of this paper. We start in Sections 3 and 4 by defining the basic terms, evaluation metrics, and options for copy detection. Then in Section 5 we describe our working prototype, COPS, and report on some initial experiments. A sampling technique that can reduce the storage space of registered documents or can speed up checking time is presented and analyzed in Section 6.

3 General Concepts

In this section we define some of the basic concepts for copy detection and for evaluating mechanisms that implement it. (As far as we know, text copy detection has not been formally studied, so we start from basics.) The starting point is the concept of a *document*, a body of text from which some structural information (such as word and sentence boundaries) can be extracted. In an initial phase, formatting information and non-textual components are removed from documents (see Section 5). The resulting *canonical form* document consists of a string of ascii characters with whitespace separating words, punctuation separating sentences and possibly a standard method of marking the beginning of paragraphs.

A *violation* occurs when a document infringes upon another document in some way (e.g., by duplicating portions of text). There are a number of violation types which can occur including plagiarism of a few sentences, exact replication of the entire document, and many steps in between. The notion of checking for a particular type of violation between two documents is captured by a *violation test*. If t is a violation test and $t(d, r)$ holds, then document d violates document r according to the particular test. For example, $Plagiarism(d, r)$ is true if document d has plagiarized from document r. We also extend this notation to include checking against a set of documents:

$t(d, \mathcal{R})$ is true if and only if $t(d, r)$ holds for some document $r \in \mathcal{R}$.

Most of the violation tests we are interested in are not well defined and require a decision by a human being. For example, plagiarism is particularly difficult to test for. For instance, the sentence "The proof is as follows" may occur in many scientific papers and would not be considered plagiarism if it occurred in two documents, while this sentence most certainly would. If we consider a test *Subset* that detects if a document is essentially a subset of another one, we again need to consider if the smaller document makes any significant contributions. This again, requires human evaluation.

The goal of a copy detection system is to implement well defined algorithmic tests, termed *operating tests* (with the same notation as violation tests), that approximate the desired violation tests. For instance, consider the operating test $t_1(d, r)$ that holds if 90% of the sentences in d are contained in r. This test may be considered an approximation to the Subset test described above. If the system flags t_1 violations, then a human can check if they are indeed Subset violations.

3.1 Ordinary Operational Tests

In the rest of this paper we will focus on a specific class of operational tests, *ordinary operational tests* (OOTs), that can be implemented efficiently. We believe they can accurately approximate many violation tests of interest, such as Subset, Overlap, and Plagiarism.

Before we describe OOTs we need to define some primitives for specifying the level of detail at which we look at the documents. As mentioned in Section 3, documents contain some structural information. In particular, documents can be divided into well defined parts, consistent with the underlying structure such sections, paragraphs, sentences, words, or characters. We call each of these types of divisions a unit type and particular instances of these unit types are called *units*.

We define a *chunk* as a sequence of consecutive units in a document of a given unit type. A document may be divided into chunks in a number of ways since chunks can overlap, may be of different sizes, and need not completely cover the document. For example, let us assume we have a document ABCDEFG where the letters represent sentences or some other units. Then it can be organized into chunks as follows : A,B,C,D,E,F,G; or AB,CD,EF,G; or AB,BC,CD,DE,EF,FG; or ABC,CD,EFG; or A,D,G. A method of selecting chunks from a document divided into units is a *chunking strategy*. It is important to note that unlike units, chunks have no structural significance to the document and so chunking strategies cannot use structural information about the document.

An OOT, o, uses hashing to detect matching chunks and is implemented by the set of procedures in Figure 1. The code is intended to convey key concepts, not an efficient or complete implementation. (Section 5 describes our actual prototype system.) First there is the preprocessing operation, `PREPROCESS(R, H)`, that takes as input a set of registered documents `R` and creates the hash table, `H`. Second, there are procedures for on-the-fly adding documents to `H` (registering new documents) and for removing them from `H` (un-registering documents). Third is the function `EVALUATE(d, H)` that computes $o(d, \mathcal{R})$.

To insert documents in the hash table, procedure `INSERT` uses a function `INS-CHUNKS(r)` to break up a document `r` into its chunks. The function returns a set of tuples. Each `<t,l>` tuple represents one chunk in `r`, where `t` is the text in the chunk, and `l` is the location of the chunk, measured in some unit. An entry is stored in the hash table for every `<t,l>` chunk in the document.

Procedure `EVALUATE(d, H)` tests a given document `d` for violations. The procedure uses `EVAL-CHUNKS` function to break up `d`. The reason why we use a different chunking function at evaluation time will become apparent in Section 6. For now, we can assume that both `INS-CHUNKS` and `EVAL-CHUNKS` are identical and we use `CHUNKS` to refer to them.

After chunking, procedure `EVALUATE` then looks up the chunks in the hash table `H`, producing a set of tuples `MATCH`. Each `<s,ld,r,lr>` in MATCH represents a match: a chunk of size `s` at location `ld` in document `d` matches (has same hash key) as a chunk at location `lr` in registered document `r`. The `MATCH` set is then given to function `DECIDE(MATCH, SIZE)` (where `SIZE` is the number of

```
PREPROCESS(R,H)
    CREATETABLE(H)
    for each r in R INSERT(r,H)

INSERT(r,H)
    C = INS-CHUNKS(r) /* OOT dependent */
    for each <t, l> in C
        h = HASH(t)
        /* assume size of reg. doc. may be obtained from id */
        INSERTCHUNK(<h,r,l>, H) /* implementation unspecified */

DELETE(r,H)
    C = INS-CHUNKS(r)
    for each <t, l> in C
    h = HASH(c)
    DELETECHUNK(<h,r,l>, H) /* implementation unspecified */

EVALUATE(d,H)
    C = EVAL-CHUNKS(d)
    SIZE = |C|
    MATCHES = {} /* empty set */
    for each <t, ld> in C
        h = HASH(t)
        SS = LOOKUP(h, H) /* returns all <r, lr> with matching h */
        for each <r, lr> in SS
            MATCHES += <|t|, ld, r, lr>
    return DECIDE(MATCHES, SIZE) /* OOT dependent */
```

Figure 1: Pseudo-code for OOT

chunks in `d`) that returns the set of matching registered documents. If the set is non-empty, then there was a violation, i.e., $o(d, \mathcal{R})$ holds.

Note that an instance of an OOT is specified simply by its `INS-CHUNKS`, `EVAL-CHUNKS` and `DECIDE` functions. That is, this is the only way in which OOTs differ. In particular, in Section 5 we will start by considering an OOT where both its `CHUNKS` functions extract sentences, and its `DECIDE` function selects registered documents that exceed some threshold fraction ϕ of matching chunks. That is, let `COUNT(r, MATCH)` be the number of tuples of the form `<-,-,r,->` in `MATCH`. Then document `r` will be selected if `COUNT(r, MATCH)` is greater than ϕ`SIZE`. For example, if $\phi = 0.4$ and the document to check has 100 sentences, then registered documents with 41 or more matching sentences will be selected. We call this `DECIDE` function the *match_ratio* function.

In the code of Figure 1 we only store the ids of registered documents in `H`, not the full documents. That is, for a tuple `<h,r,l>` in `H`, `r` is simply the name or id of `r`. The copy detection system may also store separately the registered documents. (Our COPS prototype does this.) This can be useful for showing a user the matching documents and highlighting the matching chunks.

3.2 Measuring Accuracy

As described earlier, OOTs (and operational tests in general) are intended for approximating violation tests such as Plagiarism and Subset. It is therefore important to evaluate how well an OOT approximates some other test. It is also important to evaluate the security of OOTs, i.e., how hard it is to subvert the copy detection, as well as their efficiency, i.e., what computational resources they require. Accuracy and security are discussed in the rest of this section; efficiency is addressed in Section 6.

Assume a random registered document Y chosen from a distribution of registered documents R. That is, the probability that Y is a particular document r_1 out of a population of registered documents is $R(r_1)$. Similarly, assume a random test document X is selected from a distribution of test documents D. We can then define the following accuracy metrics, each implicitly parametrized by R and D.

Definition 3.1 *For a test t, we define $freq(t) = P(t(X,Y))$. ("P" stands for "probability.")*

Intuitively, *freq* measures how frequently a test is true. For example, suppose X is uniform over $\{x_1, x_2\}$ (either one of two test documents is just as likely to be tested) and Y is uniform over $\{y_1, y_2, y_3\}$ (all three of these documents are equally likely to be registered). Further, assume that only $t(x_1, y_2)$, $t(x_1, y_3)$, $t(x_2, y_3)$ hold (i.e., only these pairs of documents are t violations). Then $freq(t) = 3/6 = 1/2$ since $t(x, y)$ holds for 3 out of the 6 possible choices for (x, y).

If an operating test approximates a violation test well, then their *freq*'s should be close but the converse is not true since they can accept on disjoint sets. If the *freq* of the operating test is small compared to the violation test it is approximating, then it is being too conservative. If it is too large then the operating test is too liberal.

Suppose we have an operating test t_2 and a violation test t_1. Then we define the following measures for accuracy. (Note that these can also be applied between two operating tests and in general between any two tests).

Definition 3.2 *The Alpha metric corresponds to a measure of false negatives, i.e., $Alpha(t_1, t_2) = P(\neg t_2(X,Y) \mid t_1(X,Y))$. Note Alpha is not symmetric. A high $Alpha(t_1, t_2)$ value indicates that operating test t_2 is missing too many violations of t_1.*

Definition 3.3 *The Beta metric is analogous to Alpha except that it measures false positives, i.e., $Beta(t_1, t_2) = P(t_2(X,Y) \mid \neg t_1(X,Y))$. Beta is not symmetric either. A high $Beta(t_1, t_2)$ value indicates that t_2 is finding too many violations not in t_1.*

Definition 3.4 *The Error metric is the combination of Alpha and Beta in that it takes into account both false positives and false negatives and is defined as $Error(t_1, t_2) = P(t_1(X,Y) \neq t_2(X,Y))$. It is symmetric. A high Error value indicates that the two tests are dissimilar.*

3.3 Security

So far we have assumed that the author of a test document does not know how our copy detection system works and does not intend to sabotage it. However, another important measure for an OOT is how hard it is for a malicious user to break it. We measure this notion of *security* in terms of how many changes need to be made to a registered document so that it will not be identified by the OOT as a copy.

Definition 3.5 *The security of an OOT o (also applicable to any operating test) on a given document r, $SEC(o, r)$, is the minimum number of characters that must be inserted, deleted, or modified in r to produce a new document r' such that $o(r', r)$ is false. The higher a $SEC(o, r)$ value is, the more secure o is.*

We can use this notion to evaluate and compare OOTs. For example, consider an OOT o_1 that considers the entire document as a single chunk. Then $SEC(o_1, r) = 1$ for all r, because changing a single character makes the document not detectable as a copy. [2]

[2]This assumes a decision function which doesn't flag a violation if there are no matches (a reasonable condition). For instance, if $o_1(d, r)$ is always true, no matter if there are matches or not, then our statement does not hold.

As another example consider OOT o_2 that uses sentences as chunks and a *match_ratio* decision function. Then $SEC(o_2, r) = (1-\phi)\texttt{SIZE}$ where SIZE is the number of sentences in r. For instance, if $\phi = 0.6$ and our document has 100 sentences, we need to change at least 40 of them. As a third example, consider an OOT o_3 that uses pairs of overlapping sentences as chunks. For instance, if the document has sentences A, B, C, ... , o_3 considers chunks AB, BC, CD, ... Here we need to modify half as many sentences as before (roughly), since each modification can affect two chunks. Thus, $SEC(o_3, r)$ is approximately equal to $SEC(o_2, r)/2$, i.e., o_3 is approximately half as secure as o_2.

Note that our security definition is weak because it assumes the adversary knows all about our OOT. However, by keeping certain information about our OOT secret we can enhance security. We can model this by having a large class of OOTs, O, that vary only by some parameters, and then secretly choosing one OOT from O. We assume that the adversary does not know which OOT we have chosen and thus needs to subvert all of them. For this model we define $SEC(O, r)$ as the number of characters that must be inserted, deleted, or modified to make $o(r', r)$ false for all $o \in O$. For examples of using classes of OOT's see chunking strategy D of Section 4.2 and Section 6 (consider the seed for the random number generator as a parameter).

Finally, notice that the security measures we have presented here do not address "authorization" issues. For example, when a user registers a document, how does the system ensure the user is who he claims to be and that he actually "owns" the document? When a user checks for violations, can we show him the matching documents, or do we just inform him that there were violations? Should the owner of a document be notified that someone was checking a document that violates his? Should the owner be given the identity of the person submitting the test document? These are important administrative questions that we do not attempt to address in this paper.

4 Taxonomy of OOTs

The units selected, the chunking strategy, and the decision function can affect the accuracy and the security of an OOT. In this section we consider some of the options and the tradeoffs involved.

4.1 Units

To determine how documents are to be divided into chunks we must first choose the units. One key factor to consider is the number of characters in a unit. Larger units (all else being equal) will tend to generate fewer matches and hence will have a smaller *freq* and be more selective. This, of course, can be compensated by changing the chunk selection strategy or decision function.

Another important factor in the choice of a unit type is the ease of detecting the unit separators. For example, words that are separated by spaces and punctuation are easier to detect than paragraphs which can be distinguished in many ways.

Perhaps the most important factor in unit selection is the violation test of interest. For instance, if it is more meaningful that sequences of sentences were copied rather than sequences of words (e.g., sentence fragments), then sentences and not words should be used as units.

4.2 Chunks

There are a number of strategies for selecting chunks. To contrast them we can consider the number of units involved, the number of hash table entries that are required for a document, and an upper bound for the security $SEC(o, r)$. [3] See Table 1 for a summary of the four strategies we consider. (There are also many variations not covered here.) In the table, $|r|$ refers to the number of units in

[3]For our discussion we assume that documents do not have significant numbers of repeating units.

strat	*summary*	*example on ABCDEF* ($k = 3$)	*space*	*# units*	$SEC \leq$
A	1 unit	A,B,C,D,E,F	$\|r\|$	1	$\|r\|$
B	k units, 0 over	ABC,DEF	$\|r\|/k$	k	1
C	k units, k-1 over	ABC,BCD,CDE,DEF	$\|r\|$	k	$\|r\|/k$
D	hashed breakpoints	AB,CDEF	$\|r\|/k$	k	$\|r\|$

Table 1: Properties of Chunking Strategies

the document r being chunked, and k is a parameter of the strategies. The "space" column gives the number of hash table entries need for r, while "# units" gives the chunk size.

(A) One chunk equals one unit. Here every unit (e.g. every sentence) is a chunk. This yields the smallest chunks. As with units, small chunks tend to make the *freq* of an OOT smaller. The major weakness is the high storage cost; $|r|$ hash table entries are required for a document. However, it is the most secure scheme; $SEC(o, r)$ is bounded by $|r|$. That is, depending on the decision function, it may be necessary to alter up to $|r|$ characters (one per chunk) to subvert the OOT.

(B) One chunk equals k nonoverlapping units. In this strategy, we break the document up into sequences of k consecutive units and use these sequences as our chunks. It uses $(1/k)^{th}$ the space of Strategy A but is very insecure since altering a document by adding a single unit at the start will cause it to have no matches with the original. We call this effect "phase dependence". This effect also leads to high Alpha errors.

(C) One chunk equals k units, overlapping on $k-1$ units. Here, we take every sequence of k consecutive units in our document as our chunks. Therefore we do not suffer from phase dependence as in Strategy B but unfortunately the space cost is equivalent to Strategy A. Comparing an OOT o_A that uses Strategy A, and an OOT o_C that is the same except for its use of Strategy C, one can see that $Alpha(o, o_C) \geq Alpha(o, o_A)$ and $Beta(o, o_C) \leq Beta(o, o_A)$ for any violation test o. This is because $o_C(d, r)$ being true implies that $o_A(d, r)$ is true. Thus Strategy C is prone to higher Alpha errors (but lower Beta errors). Also, Strategy C is relatively insecure (though more secure than B) in that modifying every k^{th} unit of a registered document is sufficient to fool the system.

(D) Use nonoverlapping units, determining break points by hashing units. We start by hashing the first unit in the document. If the hash value is equal to some constant x modulo k, then the first chunk is simply the first unit. If not, we consider the second unit. If its hash value equals x modulo k, the the first chunk is the first two units. If not we consider the third unit, and so on until we find some unit that hashs to x modulo k, and this identifies the end of the first chunk. We then repeat the procedure to identify the following nonoverlapping chunks.

It can be shown that the expected number of units in each chunk will be k. Thus, Strategy D is similar to B in its hash table requirements. However, unlike B, it is not affected by phase dependence since similar text will have the same break points. Strategy D, like C, has higher Alpha (and lower Beta) errors as compared to A. Furthermore, all else being the same, *freq* should be only slightly less than that of C because significant portions of duplicated text will be caught just as in C.

The key advantage of Strategy D is that it is very secure. (It is really a family of strategies with a secret parameter; see Section 3.3.) Without knowing the hash function, one must change every unit of a test document to be sure it will get through the system without warnings.

4.3 Decision Functions

There are many options for choosing decision functions. The *match_ratio* function (Section 3.1) can be useful for approximating Subset and Overlap violation tests. Another simple decision function is *matches* (with parameter k) that simply tests if the number of matches between the test and the registered document is above a certain value k. This would be useful for detecting violations such as Plagiarism. One might also consider using *ordered_matches* which tests whether there are more than a certain number of matches occurring the same order in both documents. This would be useful if unordered matches are likely to be coincidental.

5 Prototype and Preliminary Results

We have built a working OOT prototype to test our ideas and to understand how to select good `CHUNKS` and `DECIDE` functions. The prototype is called *COPS* (COpy Protection System) and Figure 2 shows its major modules. Documents can be submitted via email in TEX (including LATEX), DVI, troff and ASCII formats. New documents can be either registered in the system or tested against the existing set of registered documents. If a new document is tested, a summary is returned listing the registered documents that it violates.

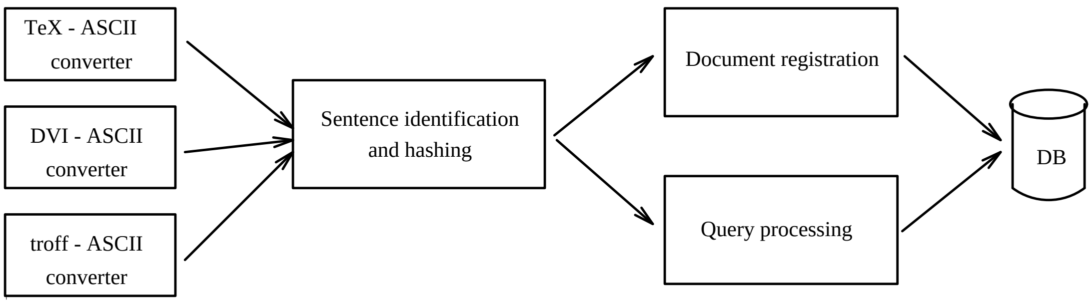

Figure 2: Modules in COPS implementation.

COPS allows modules to be easily replaced, permitting experimentation with different strategies (e.g., different `INS-CHUNKS`, `EVAL-CHUNKS` and `DECIDE` functions). We will begin our explanation with the simplest case (sentence chunking for both insertion and evaluation, and a *match_ratio* decision function) and later discuss possible improvements. A document that has been submitted to the system is given a unique document ID. This ID is used to index a table of document information such as title and author. To register the document, first it must be converted into the canonical form, i.e., plain ASCII text. The process by which this occurs is dependent upon the document format. A TEX document can be piped through the Unix utility *detex*, while a document with troff formatting commands can be converted with *nroff*. Similarly DVI and other document formats have filters to handle their conversion to plain ASCII text. After producing plain ASCII we are ready to determine and hash the document's individual sentences. Using periods, exclamation points, and question marks as sentence delimiters, we hash each sentence into a numeric key. The current document's unique ID is then stored in a permanent hash table, once for each sentence.

When we wish to check a new document against the existing set of registered documents, we use a very similar procedure. We generate the plain ASCII, determine sentences, and generate a list of hash keys, and look them up in the hash table (see Section 3.1). If more than ϕ`SIZE` sentences match with any given registered document we report a possible violation.

5.1 Conversion to ASCII

The procedure described above is the ideal case. In practice a number of interesting difficulties arise. Let us first consider some of the challenges associated with the conversion to ASCII text. The most important is that no exact objective method of reducing a formatted document to ASCII exists. Documents are formatted using TEX or troff precisely because there is some "value added" over plain text. This extra formatting cannot be represented in ASCII, and so will be lost. For example, embedded graphs have no ASCII equivalent. We can retain any text items or labels associated with the graph, but the primary structure is not translatable. Equations and tables are difficulties as well. In our implementation we discard graphs, equations, tables, pictures, and all other pieces of information that cannot be represented naturally in ASCII. We also choose to discard all text formatting commands that effect the presentation, but not the content, of the document. For example, command sequences to produce italic type and change font are removed and ignored.

The conversion process is not perfect. If the document input format is DVI, then it is sometimes impossible to distinguish "equations" from "plain text". Consider the sentence, "Let X+Y equal the answer." This sentence will be translated to ASCII exactly as it is shown. However, if we begin with TEX, then the equation will be discarded, leaving the sentence "Let equal the answer." Since the conversion to plain ASCII produced different sentences, our system would be unable to recognize that a sentence match occurred. Later in this section we will discuss some system enhancements that allows us to detect matching sentences, despite imperfect translations.

Another complication with DVI is that it gives directions for placing text on a page but it does not specify what text is part of the main body, and what is part of subsidiary structures like footnotes, page headers and bibliographies. Our DVI converter does not attempt to rearrange text; it simply considers the text in the order it appears on the page. However, one case it does handle is that of two column format. Instead of reading characters left to right, top to bottom (which would corrupt most sentences in a two column format), the converter detects the inter-column gap and reads down the left column and then the right one.

An input format COPS can not handle in general is Postscript. Since Postscript is actually a programming language, it is very difficult to convert its layout commands to plain ASCII text. Some Postscript generators such as *dvips, enscript,* and Microsoft Word produce relatively simple Postscript from which text can be extracted. However, others such as Interleaf produce Postscript code which would require the generation of page bit maps. These could be scanned with OCR (optical character recognition) to analyze and reconstruct the text. This process is difficult and error prone.

In summary, the approach we have taken with the COPS converters is to do a reasonable job converting to ASCII, but not necessarily perfect. Most matching sentences that are not translated identically, will still be found by the system, since enhancements discussed later attempt to negate the effects of common translation misinterpretations. Even if some matching sentences are missed, there should be enough other matches in overlapping documents so that COPS can still flag the violations. Later, we present experimental results that confirm this.

5.2 Sentence Identification and Hashing

Difficult problems also arise in the sentence identification and hashing module. In particular, even if we are given correctly translated plain ASCII, it is not always clear how to extract sentences. As a first approximation, we can identify a sentence by merely taking all words up to a period or question mark. However, sentences that contain "e.g." or other abbreviations will be broken into multiple parts because of the embedded periods. An extension to our simple model explicitly watches for and eliminates common abbreviations such as "e.g." and "i.e." so that sentences will not be broken in this way. Nevertheless, unexpected abbreviations will still cause difficulties. For example, given the actual sentences, "I am a U.S. citizen." and "The U.S. is large." our system will identify the following set of sentences. "I am a U.", "S.", "citizen.", "The U.", "S.", and "is large." Notice that the sentence "S." is identified twice. The system will flag this as a match, even though the actual sentences are not the same. To reduce this sort of error we can disregard sentences composed of a single word; however, other similar errors may still occur. For example, title and author names at the head of a document are also difficult to extract as sentences, since they rarely end with punctuation. We discuss later some further improvements to the simple algorithm we have described here. Note that paragraph detection, if it were needed, would involve similar issues. COPS currently does not detect paragraphs.

The units used by COPS' OOT are words and sentences (see Section 3.1). COPS first converts each word in the text to a hash key. The result is a sequence of hash keys with interspersed end-of-sentence markers. The chunking of this sequence is done by calling a procedure `COMBINE(N-UNITS, STEP, UNIT-TYPE)`, where `N-UNITS` is the number of units to be combined into the next chunk, `STEP` is the number of units to advance for the next chunk, and `UNIT-TYPE` indicates what should be considered a unit. For example, repreatedly calling `COMBINE(1, 1, WORD)` creates a chunk for each word in the input sequence. Calling `COMBINE(1, 1, SENTENCE)` creates a chunk for each sentence. Using `COMBINE(3, 3, WORD)` takes every three words as a chunk, while `COMBINE(3, 1, WORD)` produces overlapping three word chunks. `COMBINE(2, 1, SENTENCE)` would produce overlapping two sentence chunks. Thus, we can see that this scheme gives us great flexibility for experimenting with different `CHUNKS` functions. However, it should be noted that once a `CHUNKS` function is chosen, it must be used consistently for all documents. That is, the flexibility just described is useful only in an experimental setting.

5.3 Exploratory Tests

To evaluate the accuracy of the system, we conducted some exploratory experiments using a set of ninety two Latex, ASCII, and DVI technical documents (i.e., papers like this one). These experiments are not intended to be comprehensive; our goal is simply to understand how many matching chunks real documents might be expected to have, and how well our converters work.

The documents average approximately 7300 words and 450 sentences in length. Approximately half of these documents are grouped into nine topical sets (labeled A, B, ..., I in the tables). The two or three documents within each group are closely related, usually multiple revisions of a conference or journal paper describing the same work. The documents in separate topical groups are unrelated except for the author's affiliation with our research group at Stanford. The remaining half of the documents not in any topical group are drawn from outside Stanford and not related to any document in our collection.

All of these documents were registered in COPS, and then each was queried against the complete set. Our goal is to see if COPS can determine the closely related documents. Using the terminology of Section 3, we are considering a violation test Related(d, r) that evaluates to true if d and r are in the same group. This will be approximated by an OOT that computes the percentage of matching sentences in d and r. If the number if high, the documents will be assumed to be related.

Table 2 shows results from our exploration. Instead of reporting the number of violations that a particular *match_ratio* would yield, we show the percentage of matching sentences in each case. This gives us more information regarding the closeness of documents.

The first result column in Table 2 gives the precent matches of each document against itself. That is, for each document `d` in a group, we compute 100×`COUNT(d, MATCH)/SIZE` (see Section 3.1), average the values and report it in the row for that group. The fact that all values in the first column are 100% simply confirms that COPS is working properly.

The numbers in the second column are computed as follows. For each document `d` in a group, we compute 100×`COUNT(r, MATCH)/SIZE` for all other documents `r` in the group, and average the results. We refer to values in the second column as "affinity" values since they represent how

	Match self	Match Related Documents (Affinity)	Match Unrelated Documents (Noise)
Group A	100%	71.9%	0.6%
B	100%	N/A	0.9%
C	100%	38.6%	0.9%
D	100%	42.9%	0.3%
E	100%	38.4%	0.2%
F	100%	63.0%	0.8%
G	100%	66.0%	0.4%
H	100%	38.4%	0.4%
I	100%	93.3%	1.3%
TotalAvg	100%	52.9%±25.16%	0.6%±2.1%

Table 2: Average number of matching sentences.

close documents are. For the third column, we compare each `d` in a group against all `r` in others groups. We refer to number in this column as "noise" since they represent undesired matches. The numbers reported at the bottom of Table 2 are the averages over all document comparisons performed for that column. We also report the standard deviation between individual tests to illustrate the spread of values.

Ideally, one wants affinity values that are as high as possible, and noise values that are as low as possible. This makes it possible for a threshold value that is between the affinity and noise levels to distinguish between related and unrelated documents. Table 2 reports that related documents have on average 53% matching sentences, while unrelated ones have 0.6%. The reason why affinity is relatively low is that the notion of "Related" documents we have used here is very broad. For example, often the journal version and the conference version of the same work are quite different.

The noise level of 0.6%, equivalent to 2 or 3 sentences, is larger than what we expected. The discrepancy is caused by several things. A few sentences, such as, "This work partially supported by the NSF" are quite common in journal articles, so that even unrelated documents might both contain it. Other sentences may also be exact replicas by coincidence. Hash collisions may be another factor, especially when there are large numbers of registered documents, but are not an issue in our experiments. Also note the relatively large variance reported in the table. In particular, some unrelated documents had on the order of 20 matching sentences.

The process by which a document is translated to ASCII also has some effect on the noise level. For example, the translation we use to convert TeX documents produces somewhat less noise than does our translation from DVI. This is caused by differences in the inclusion of references. Many unrelated documents cite the same references, possibly generating matching sentences. Our TeX filter does not include references in its output (they are in separate "bib" files), so noise is reduced. The differences in noise generated by ASCII translation become less significant when the enhancements discussed later are added to our system.

The larger the noise level, the harder it is to detect plagiarism of small passages (e.g., a paragraph or two). If we set the threshold ϕ at say 5/`SIZE` sentences, the OOT would have a high Beta error rate (too many unrelated documents flagged as Plagiarism violations), while if we set it higher, say 10/`SIZE`, we would miss actual violations (high Alpha error). Thus, it is important to reduce the noise level as much as possible.

5.4 Enhancements

However, we need to decrease the noise without sacrificing affinity. If affinity is too low, it makes it hard to approximate the Related target test (again leading to high Alpha or Beta errors). With this goal in mind, we have considered a series of enhancements to the basic COPS algorithms. The results are summarized in Table 3. The first line represent the base case; each additional line of the table represents an independent enhancement. The reported values are averages over all document groups (i.e., equivalent to the last row of Table 2).

	Match self	Match Related (Affinity)	Match Unrelated (Noise)
Simple Method	100%	53.0%	0.61% ±2.08
No Common Chunks	100%	53.4%	0.06% ±0.30
Drop Numbers	100%	54.1%	0.47% ±1.34
No Short Sentences	100%	51.8%	0.04% ±0.21
No Short Words	100%	54.4%	0.36% ±0.90
All Enhancements	100%	53.6%	0.03% ±0.20

Table 3: COPS Enhancements.

In the "no common chunks" enhancement, chunks occurring in our hash table more than ten times are eliminated by the `LOOKUP` function (see Figure 1). This keeps legitimate common phrases and passages from causing a document violation. For example, the sentence "This work supported by the NSF," which is present in many documents, will not be reported as a match. The last three enhancements remove the indicated occurrence from the input stream. For "drop numbers," any word with a numeric digit is dropped; "short sentences" are arbitrarily defined to have three or fewer words; "short words" are defined to have three or fewer characters. These enhancements were motivated by our discovery that numbers, short sentences, and short words were sometimes involved in incorrect matches. (Recall the problem with abbreviations like "U.S." described in Section 5.2.)

The last row of Table 3 shows the effect of using all enhancements at once. One can see that the combined enhancements are quite effective at reducing the noise while keeping the affinity at roughly the same levels. We note that the parameter values we used for the enhancements (e.g., the number of occurrences that makes a chunk "common") worked well for our collection, but probably have to be adjusted for larger collections.

In Figure 3 we study the effect of increasing the number of overlapping sentences per chunk (without any of the enhancements of Table 3). The solid line shows the average noise as a function of the number of overlapping sentences in a chunk. As we see, the noise decreases dramatically as the number of overlapping sentences grows. This is beneficial since it decreases the minimum amount of plagiarism detectable. Figure 3 shows an "effective noise" curve that is the average noise plus three standard deviations. If we assume that noise is a normally distributed variable, we can interpret the effective noise curve as a lower bound for the threshold in order to eliminate 99% of the false positives due to noise. For example, if we use three sentence chunks and set our threshold at $\phi = 0.01$, then the Beta error will be less than 1%.

However, as described in Section 4.2, the Alpha error will increase as we combine sentences in chunks. This mean that, for instance, we will be unable to detect plagiarism of multiple, non-contiguous sentences. Also, the security of the system is reduced (Section 4.2): it takes fewer changes to a document to make it pass as a new one.

5.5 Effect of Converters

A final issue we investigate is the impact of different input converters. For example, say a Latex document is initially registered in COPS. Later, the DVI version of the same document (produced by running the original through the Latex processor) is submitted for testing. We would like to find that (a) the DVI copy clearly matches the registered Latex original, and (b) the DVI copy has

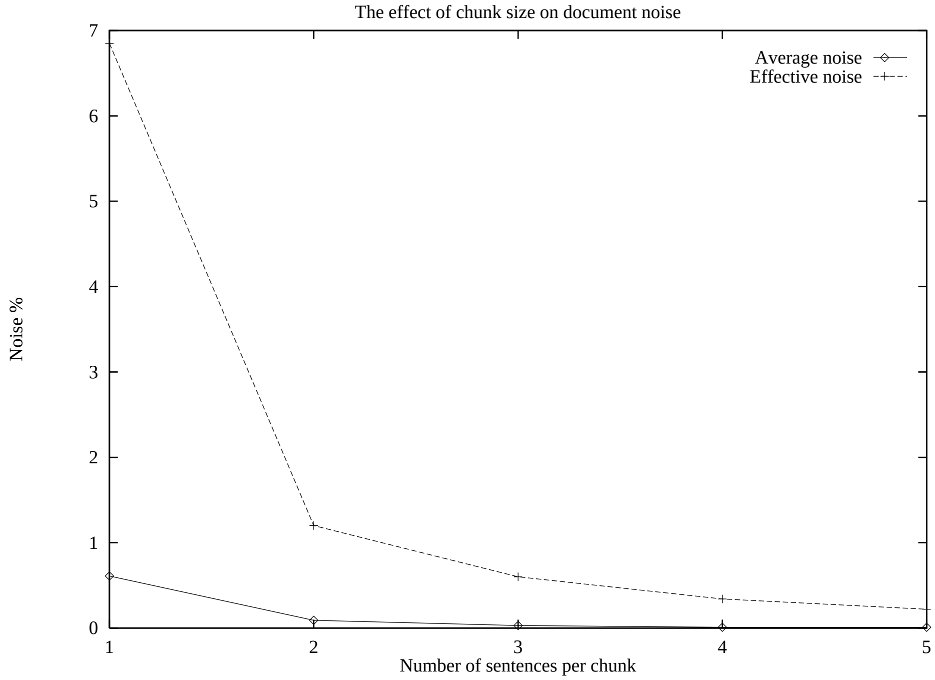

Figure 3: Noise as a function of number of overlapping sentences.

a similar number of matches with other documents as the original would have had.

Table 4 explores this issue. The first row is for the basic COPS algorithm; the second row is for the version that includes all the enhancements of Table 3. The first, third, and fifth columns are as before and are only included for reference. The "Altered Self" column reports the average precent of matching sentences when a DVI document is compared against its Latex original. The "Altered Related" column gives the average percent matching sentences when a DVI document is compared to all of the related Latex documents. Although the results are far from perfect, there seem to remain enough matches so that the DVI can be flagged as related to its original and to documents its original was related to.

	Match Self	Altered Self	Related Group	Altered Rel.	Unrelated
Simple	100%	60.9%	52.9%	36.0%	0.50%
Enhanced	100%	76.5%	53.6%	46.2%	0.03%

Table 4: Results for mechanically altered documents.

We believe that the results presented in this section, although not definitive, provide some insight into the selection of a good threshold value for COPS, at least for the Related target test. A threshold value of say $\phi = 0.05$ (25 out of 500 sentences) seems to identify the vast majority of related documents, while not triggering false violations due to noise. We also conclude that detecting plagiarism of about 10 or less sentences (roughly 2% of documents) will be quite hard, without either high Alpha or Beta errors.

6 Approximating OOTs

In this section we address the efficiency and scalability of OOTs. For copy detection to scale well, we require that it can operate with very large collections of registered documents, as well as the ability to quickly test many new documents. One effective way to achieve scalability is to use sampling.

To illustrate, say we have an OOT with a `DECIDE` function that tests whether more than 15 percent of the chunks of a document `d` match. Instead of checking all chunks in `d`, we could simply take say 20 random chunks and check whether more than 3 of them matched (15% of the 20 samples). We would expect that this new OOT based on sampling approximates the original OOT. If the average test document contains 1000 chunks, we will have reduced our evaluation time by a factor of 50. The cost, of course, is in the lost accuracy and that is analyzed in Section 6.1.

Another sampling option is to sample registered documents. The idea here is to only insert in our hash table a random sample of chunks for each registered document. For example, say that only 10% of the chunks are hashed. Next, suppose that we are checking all 100 chunks of a new document and find 2 matches with a registered document. Since the registered document was sampled, these 2 matches should be equivalent to 20 under the original OOT. Since 20/100 exceed the 15% threshold, the document would be flagged as a violation. In this case, the savings would be storage space: the hash table will have only 10% of the registered chunks. A smaller hash table also makes it possible to distribute it to other sites, so that copy detection can be done in a distributed fashion. Again, the cost is a loss of accuracy.

A third sampling option is to sample both at registration and at testing time. Due to space limitations, in this paper we only consider the first option, sampling for testing. However, note that the analysis for the sampling at registration time is almost identical to what we will present here, and the results are analogous.

We start by giving a more precise definition of the sampling at testing strategy. We are given an OOT o_1 with any chunking functions `INS-CHUNKS1` = `EVAL-CHUNKS1`, and the match_ratio `DECIDE1` function with threshold ϕ (Section 3.1). We define a second OOT, o_2, intended to approximate o_1. Its chunking function for evaluation, `EVAL-CHUNKS2` is simply

```
EVAL-CHUNKS2(r)
    C = EVAL-CHUNKS1(r)
    return  RANDOM-SELECT(N, C)
```

where `RANDOM-SELECT` picks `N` chunks at random. [4] The chunking function for insertions is not changed, i.e., `INS-CHUNKS2` = `INS-CHUNKS1`.

The `DECIDE1` function of o_1 selects documents `r` where the number of matching chunks `COUNT(r, MATCH)` is greater than ϕ`SIZE`. For o_2, only `N` chunks are tested (not `SIZE`), so the threshold number of chunks is ϕ`N`. Thus, `DECIDE2` selects documents `r` where the number of matching chunks `COUNT(r, MATCH)` is greater than ϕ`N`.

6.1 Accuracy of Randomized OOTs

Now we wish to determine how different o_2 is from o_1. As in Section 3.2, let D be our distribution of input documents and let R be the distribution of registered documents. Let X be a random document that follows D and Y be a random document that follows R. Let $m(X,Y)$ be the proportion of chunks (according to o_1's chunking function) in X which match chunks in Y. Then let $W(x)$ be the probability density function that $m(X,Y) = x$, i.e., $P(x_1 \leq m(X,Y) \leq x_2) = \int_{x_1}^{x_2} W(x)dx$. Using this we can compute $Alpha(o_1,o_2)$, $Beta(o_1,o_2)$, and $Error(o_1,o_2)$. The details of the computation are in Appendix A; the results are as follows:

$$
\begin{aligned}
Alpha(o_1,o_2) &= \frac{\int_\phi^1 W(x)Q(x)dx}{\int_\phi^1 W(x)dx} \\
Beta(o_1,o_2) &= \frac{\int_0^\phi W(x)(1-Q(x))dx}{\int_o^\phi W(x)dx}
\end{aligned}
$$

[4] This is not the most efficient way to sample. The code is just for explanation purposes.

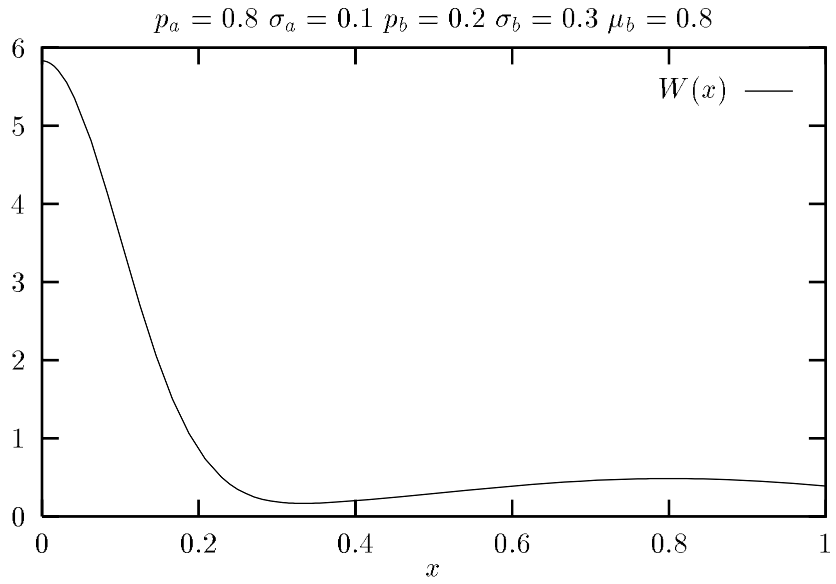

Figure 4: An Exaggerated W

$$\begin{aligned} Error(o_1, o_2) &= \int_{\phi}^{1} W(x)Q(x)dx + \int_{0}^{\phi} W(x)(1 - Q(x))dx \\ \text{where } Q(x) &= \sum_{j=0}^{\lfloor \phi \mathtt{N} \rfloor} \binom{\mathtt{N}}{j} x^j (1-x)^{\mathtt{N}-j} \end{aligned}$$

6.2 Results

Before we can evaluate our expressions, we need to know the $W(x)$ distribution. Recall that $W(x)$ tells us how likely it is to have a proportion of x matches between a test and a registered document. One option would be to measure $W(x)$ for a given body of documents, but then our results would be specific to that particular body. Instead, we use a parametrized function that lets us consider a variety of scenarios.

Using the observations of Section 5, we arrive at the following $W(x)$ function. With a very high probability p_a, the test document will be unrelated to the registered one. In this case, there can still be noise matches, which we model as normally distributed with mean 0 and standard deviation σ_a (which will probably be very small). With probability $p_b = 1 - p_a$ the test document is unrelated to the registered one. In this case we assume that the number of matching chunks is normally distributed with mean μ_b and standard deviation σ_b. We would expect σ_b to be large since, as we have seen, related documents tend to have widely varying numbers of matches. Thus, our $W(x)$ function is the weighted sum of two normal (truncated at 0 and 1) distributions, normalized to make $\int_0^1 W(x) = 1$.

Figure 4 shows a sample $W(x)$ function with exaggerated parameters to make its form more apparent. The area under the curve in the range $0 \leq x \leq 0.2$ represents the likelihood of noise matches, while the rest of the range represents mainly matches of related documents. In practice, of course, we would expect p_a to be much closer to 1 (most comparisons will be between unrelated documents) and σ_a to be much smaller.

Given a parametrized $W(x)$, we can present results that show how good an approximation o_2 is to o_1. An important first issue to study is the number of samples $\mathtt{N}$ required for accurate results. Figure 5 shows the $Alpha(o_1, o_2)$, $Beta(o_1, o_2)$, and $Error(o_1, o_2)$ values as a function of $\mathtt{N}$, for $\phi = 0.4$. Recall that the ϕ value of 0.4 means that o_1 is looking for registered documents whose

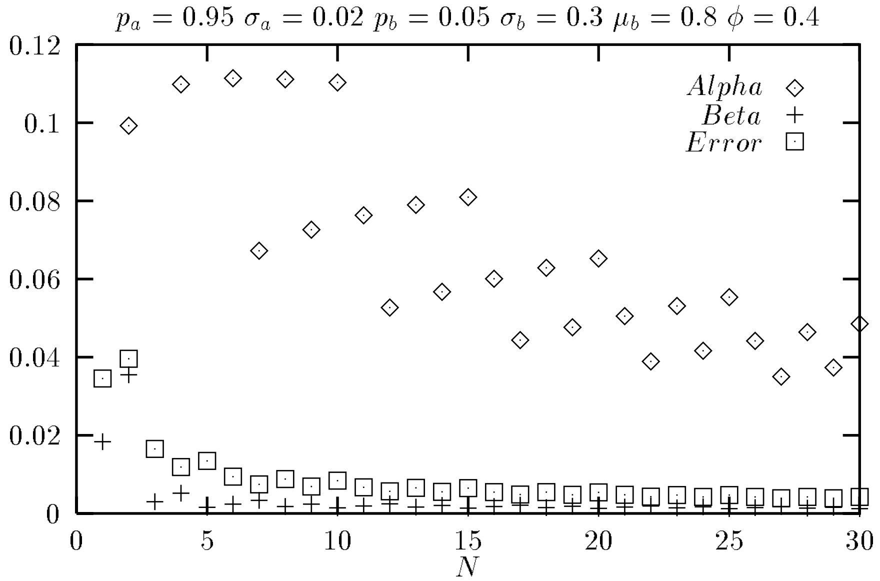

Figure 5: The Effect of the Number of Sample Points on Accuracy

chunks that match 40% of the chunks of the test document. This value may have been picked, say, because we are interested in a Subset target test. The parameters for $W(x)$ are given in the figure.

Note that the values in Figure 5 are not simply monotonically decreasing. For example, the Alpha and Error values *increase* as `N` goes from 9 to 10. Rounding error is the cause for this. For example, for `N` $= 9$, o_2 selects documents with `COUNT` (the number of matching chunks) greater than 3.6 ($=\phi$`N`), i.e., with 4 or more matches. For `N` $= 10$, documents with `COUNT` greater than 4 (i.e., 5 or more) are selected. Consider now a test document that matches with say 40% to 50% of the chunks of a registered document (hence is selected by o_1). It is more likely that o_2 with `N` $= 9$ will select it since it only has to get 4 hits. With `N` $= 10$, o_2 is less likely to select it because with only one extra sample, it has to get 5 hits. This effect leads to the higher Alpha error for `N` $= 10$.

In spite of the non-monotonicity, it is important to note how overall the Error decreases very rapidly as `N` increases. For `N` > 10, the Error stays well below 0.01. This shows that o_2 can approximate o_1 well with a relatively small number of sampled chunks.

Note, however, that the Alpha error does not decrease as rapidly, but this is not as serious. The Alpha error for `N` beyond say 20 is mainly caused by test documents whose match ratio is slightly higher than $\phi = 0.4$. (The area under the $W(x)$ curve in the vicinity to the right of 0.4 gives the probability of getting one of these documents.) In these cases, the sampling OOT may not muster enough hits to trigger a detection. However, in this case the original OOT o_1 may not very good at approximating the violation test of interest either. In other words, in the percent of matches is close to 40%, it may not be clear if the documents are related on not. Thus, the fact that o_1 detects a violation but o_2 does not is not as serious, we believe.

Our results are sensitive to the $W(x)$ parameters used, so in Figure 6 we demonstrate the effect of σ_a and in Figure 7 the effect of p_a. We can see from Figure 6 that the Error stays very low as long as σ_a is not near $\phi = 0.4$. If σ_a is close to ϕ, we get more documents in the region where o_2 has trouble identifying documents selected by o_1. Similarly, the error keeps very low in the high p_a range, which is where we expect to be in practice.

In summary, using sampling in OOTs seems to work very well under good conditions (when ϕ is far from the bulk of the match ratios). There is a large gain in efficiency with only a small loss of accuracy. As stated earlier, the sample at registration OOT can be analyzed almost identically to what we have done here, and can be shown to substantially reduce the storage costs.

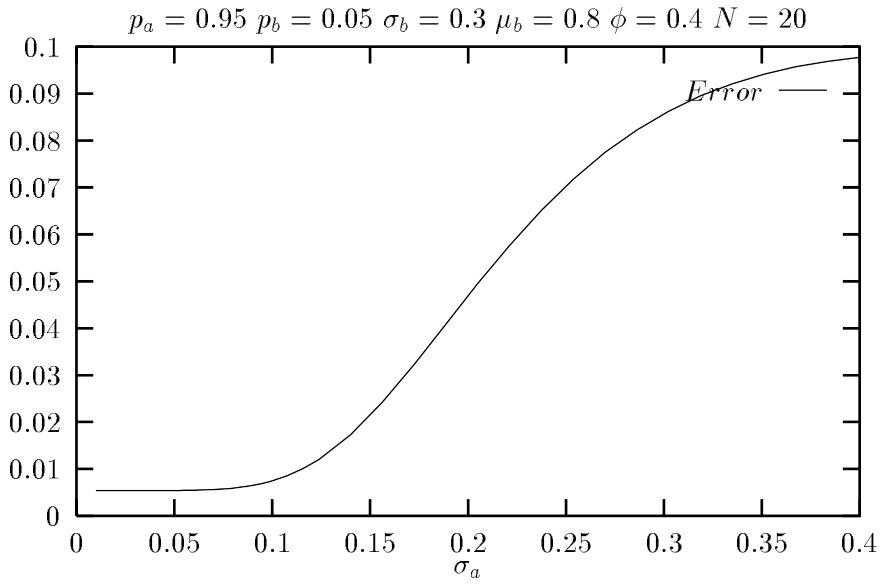

Figure 6: The Effect of σ_a on Error.

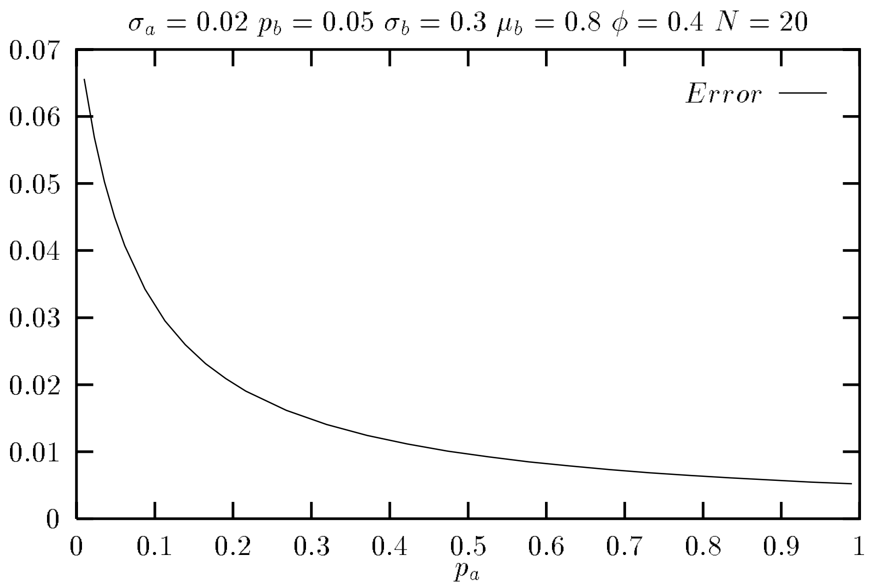

Figure 7: The Effect of p_a on Error.

7 Conclusions

In this paper we have proposed a copy detection service that can identify partial or complete overlap of documents. We described a prototype implementation of this service, COPS, and presented experimental results that suggest the service can indeed detect violations of interest. We also analyzed several important variations, including ones for breaking up a document into chunks, and for sampling chunks for detecting overlap.

It is important to note that while we have described copy detection as a centralized function, there are many ways to distribute it. For example, copies of the registered document hash table can be distributed to permit checking for duplicates at remote sites. If the table contains only samples (Section 6) it can be relatively small and distributable more easily. Also, document registration can also be performed at a set of distributed registration services. These services could periodically exchange information on new registered documents they have seen.

Perhaps the most important question regarding copy detection is whether authors can be convinced to register their documents: Without a substantial body of documents, the service will not be very useful. We believe they can, especially if one starts with the documents of a particular community (e.g., netnews users, or SIGMOD authors). But regardless of the success of COPS and copy detection, we believe it is essential to explore and understand solutions for safeguarding intellectual property in digital libraries. Their success hinges on finding at least one approach that works.

References

[1] C. Anderson. Robocops: Stewart and Feder's mechanized misconduct search. *Nature*, 350(6318):454–455, April 1991.

[2] J. Brassil, S. Low, N. Maxemchuk, and L.O'Gorman. Document marking and identification using both line and word shifting. Technical report, AT&T Bell Labratories, 1994. May be obtained from ftp://ftp.research.att.com/dist/brassil/docmark2.ps.

[3] J. Brassil, S. Low, N. Maxemchuk, and L.O'Gorman. Electronic marking and identification techniques to discourage document copying. Technical report, AT&T Bell Labratories, 1994.

[4] A. Choudhury, N. Maxemchuk, S. Paul, and H. Schulzrinne. Copyright protection for electronic publishing over computer networks. Technical report, AT&T Bell Labratories, 1994. Submitted to IEEE Network Magazine June 1994.

[5] J. R. Garrett and J. S. Alen. Toward a copyright management system for digital libraries. Technical report, Copyright Clearance Center, 1991.

[6] G. N. Griswold. A method for protecting copyright on networks. In *Joint Harvard MIT Workshop on Technology Strategies for Protecting Intellectual Property in the Networked Multimedia Environment*, April 1993.

[7] M. B. Jensen. Making copyright work in electronic publishing models. *Serials Review*, 18(1–2):62–66, 1992.

[8] R. E. Kahn. Deposit, registration and recordation in an electronic copyright management system. Technical report, Corporation for National Research Initiatives, Reston, Virginia, August 1992.

[9] P. A. Lyons. Knowledge-based systems and copyright. *Serials Review*, 18(1–2):88–91, 1992.

[10] A. Parker and J. O. Hamblen. Computer algorithms for plagiarism detection. *IEEE Trasnactions on Education*, 32(2):94–99, May 1989.

[11] G.J. Popek and C.S. Kline. Encryption and secure computer networks. *ACM Computing Surveys*, 11(4):331–356, December 1979.

[12] D. Wheeler. Computer networks are said to offer new opportunities for plagarists. *The Chronicle of Higher Education*, pages 17, 19, June 1993.

Appendix A: Error Terms for Sampling OOT

In this appendix we compute the Alpha, Beta, and Error terms for the sampling-for-testing OOT.

Recall that $m(X,Y)$ is the proportion of chunks (according to o_1's chunking function) in X which match chunks in Y, and that $W(x)$ is the probability density function that $m(X,Y) = x$, i.e., $Pr(x_1 \leq m(X,Y) \leq x_2) = \int_{x_1}^{x_2} W(x)dx$. Using this we can compute $Alpha(o_1, o_2)$ as follows:

$$\begin{aligned} Alpha(o_1, o_2) &= Pr(\neg o_2(X,Y) | o(X,Y)) & (1) \\ &= \frac{Pr(\neg o_2(X,Y) \wedge o(X,Y))}{Pr(o_1(X,Y)).} & (2) \end{aligned}$$

To compute the numerator and denominator of Equation 2 we consider every possible value of $m(X,Y)$ separately, i.e., we integrate over all possible values of $m(X,Y) = x$ and weight the integral by the relative probability of that value, $W(x)$. In the case of the denominator this is simple,

$$Pr(o_1(X,Y)) = \int_{\phi}^{1} W(x)dx. \quad (3)$$

However the numerator is somewhat more complicated:

$$\begin{aligned} Pr(\neg o_2(X,Y) \wedge o(X,Y)) &= \int_{\phi}^{1} W(x) Pr(\neg o_2(X,Y) | m(X,Y) = x) dx & (4) \\ &= \int_{\phi}^{1} W(x) Q(x) dx, & (5) \end{aligned}$$

where $Q(x) = Pr(\neg o_2(X,Y) | m(X,Y) = x)$. That is, $Q(x)$ is the probability that $\neg o_2(X,Y)$ given that an x proportion of X's chunks match in Y. So $Q(x)$ is the probability that of $\mathtt{N}$ chunks, each with probability of matching x, there will be $\leq \phi\mathtt{N}$ matches. Let each the event of chunk i matching be denoted B_i and be 1 if it matches and 0 if not. Then,

$$\begin{aligned} Q(x) &= Pr(\sum_{i=1}^{\mathtt{N}} B_i \leq \lfloor \phi\mathtt{N} \rfloor) \\ &= \sum_{j=0}^{\lfloor \phi\mathtt{N} \rfloor} Pr(\sum_{1}^{\mathtt{N}} B_i = j) \\ &= \sum_{j=0}^{\lfloor \phi\mathtt{N} \rfloor} \binom{\mathtt{N}}{j} x^j (1-x)^{\mathtt{N}-j} \end{aligned}$$

Therefore, $Q(x)$ is an evaluation of the cumulative probability mass function of a binomial distribution with parameters $\mathtt{N}$ and x evaluated at the floor of $\phi\mathtt{N}$. So,

$$Alpha(o_1, o_2) = \frac{\int_{\phi}^{1} W(x) Q(x) dx}{\int_{\phi}^{1} W(x) dx}$$

Similarly, we can derive

$$\begin{aligned} Beta(o_1, o_2) &= \frac{\int_{0}^{\phi} W(x)(1 - Q(x)) dx}{\int_{o}^{\phi} W(x) dx} \\ Error(o_1, o_2) &= \int_{\phi}^{1} W(x) Q(x) dx + \int_{0}^{\phi} W(x)(1 - Q(x)) dx. \end{aligned}$$