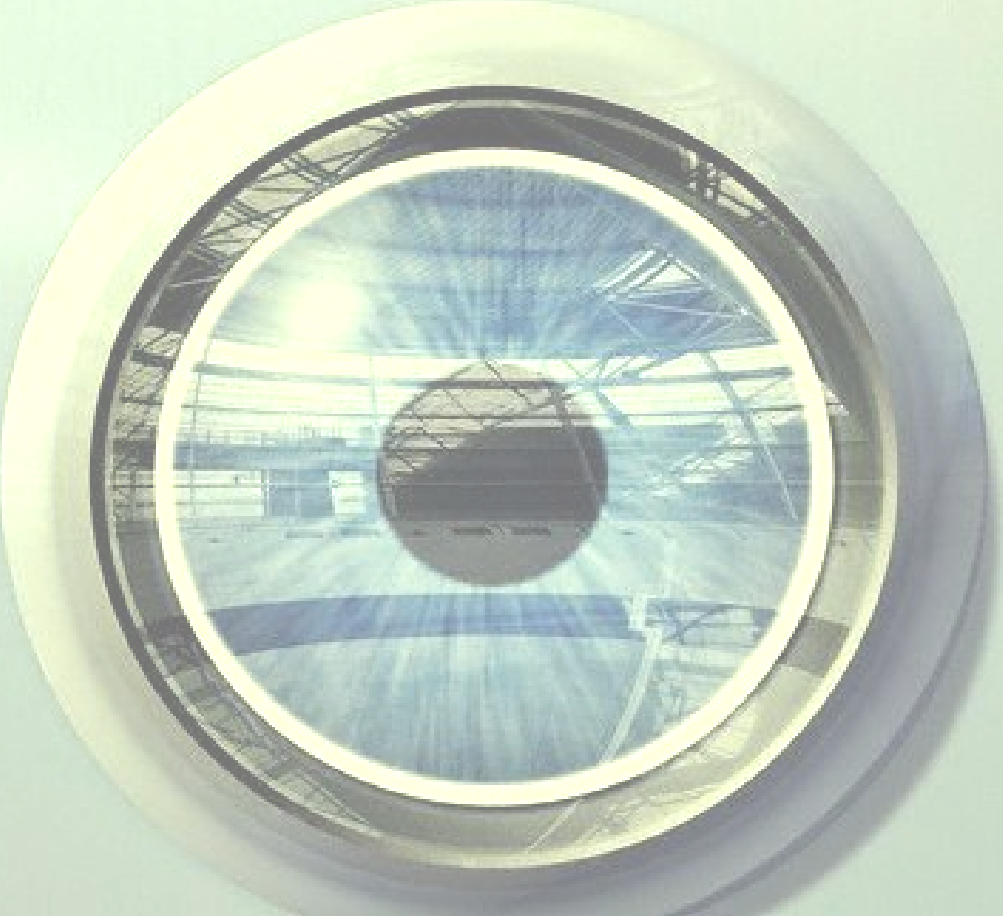
A report on

UT-Plag ™ 1.0

University of Tehran's Plagiarism Detection Software



Naeem Esfahani
Misagh Bagherian

August, 2005

Abstract

Nowadays it's the age of digital content and as you know digital content is for copying. Quotation, revision, plagiarism, and file sharing all create copies. Easy access to the web has led to increased potential for students cheating on assignments by plagiarizing others' work. Overlap detection tools are easy to use and accurate in plagiarism detection, so they can be an excellent deterrent to plagiarism. Consider a large collection of documents submitted to an online submission system in a conference or a course management system. Because of the nature of online submission systems, certainly there're some plagiarized documents which some are foolishly plagiarized thoroughly from another document and some are shrewdly plagiarized from different parts of different documents. We want to design and implement UT-Plag as a plagiarism detection software that in future will be a part of UTCM[1]. Now our system is similar to MOSS[2], a widely used plagiarism detection software at Berkeley. The results are included at the end of this report. There you can visit our future plan to enhance UT-Plag.

1 Introduction

1-1 What's plagiarism?

This is the first question that naturally comes to mind. What's plagiarism? What do we want to detect? Some important definitions that are acceptable in our work come in the following. Plagiarism refers to:

- The use of another's ideas, information, language, or writing, when done without proper acknowledgment of the original source [1].
- Buying a paper from a research service or term paper mill [2].
- Turning in another student's work without that student's knowledge [2].
- Copying a paper from a source text without proper acknowledgment [2].
- Copying materials from a source text, supplying proper documentation, but leaving out quotation marks [2].
- Paraphrasing materials from a source text without appropriate documentation [2].

In our domain specific problem in UTCM, plagiarism means submission of others' work into this course management system. As we said before this plagiarized version could be a foolishly thorough copy or a shrewdly reordered and changed one.

---

[1] University of Tehran Course Management system

[2] Measure Of Software Similarity

1-2 Plagiarism Detection

Now we want to investigate the reasons that led us to design and implement such a system. Being teacher assistants for about two years, we found that plagiarism is growing rapidly among students. Now they are so skilled in plagiarism. Usually they form colonies in which each one performs one assignment and all of them hand in others work. The most important concern is that they occasionally hand in full copies. They usually glue different parts of their friends works and internet articles, then shuffle the paragraphs and at last insert a little of their self opinions. In large collection of documents it's so hard to detect such kinds of plagiarized documents manually without aid of any software tool. Thus, students plagiarize and most of plagiarized works won't be detected. This is a dangerous issue that causes growth in digital plagiarism. We don't have any statistics about plagiarism in University of Tehran, or even in Iran. But here we mention some statistics from the CAI[3] which is till June, 2005 [3].

- On most campuses, 70% of students admit to some cheating. Close to one-quarter of the participating students admitted to serious test cheating in the past year and half admitted to one or more instances of serious cheating on written assignments.
- Academic honor codes effectively reduce cheating. Surveys conducted in 1990, 1995, and 1999, involving over 12,000 students on 48 different campuses, demonstrate the impact of honor codes and student involvement in the control of academic dishonesty. Serious test cheating on campuses with honor codes is typically 1/3 to 1/2 lower than the level on campuses that do not have honor codes. The level of serious cheating on written assignments is 1/4 to 1/3 lower.
- Internet plagiarism is a growing concern on all campuses as students struggle to understand what constitutes acceptable use of the Internet. In the absence of clear direction from faculty, most students have concluded that 'cut & paste' plagiarism - using a sentence or two (or more) from different sources on the Internet and weaving this information together into a paper without appropriate citation - is not a serious issue. While 10% of students admitted to engaging in such behavior in 1999, almost 40% admit to doing so in the Assessment Project surveys. A majority of students (77%) believe such cheating is not a very serious issue.
- Faculties are reluctant to take action against suspected cheaters. In Assessment project surveys involving almost 10,000 faculty in the last three years, 44% of those who were aware of student cheating in their course in the last three years, have never reported a student for cheating to the appropriate campus authority. Students suggest that cheating is higher in courses where it is well known that faculty members are likely to ignore cheating.
- Longitudinal comparisons show significant increases in serious test/examination cheating and unpermitted student collaboration. For example, the number of students self-reporting instances of unpermitted collaboration at nine medium to large state universities increased from 11% in a 1963 survey to 49% in 1993. This trend seems to be

---

[3] Center for Academic Integrity

continuing: between 1990 and 1995, instances of unpermitted collaboration at 31 small to medium schools increased from 30% to 38%.

- Studies of 18,000 students at 61 schools, conducted in the last four years, suggest cheating is also a significant problem in high school – over 70% of respondents at public and parochial schools admitted to one or more instances of serious test cheating and over 60% admitted to some form of plagiarism. Slightly less than half of the respondents from private schools admitted similar behaviors. About half of all students admitted they had engaged in some level of plagiarism using the Internet.

So according to these statistics we should think about a deterrent solution. Our solution is UT-Plag. This version (1.0) of UT-Plag does not support detection of plagiarism from the World Wide Web. But it will be feasible in future versions. It's a crucial concern. If you don't know the reason visit following tools that students use to plagiarize their work from the World Wide Web:

- http://fastpapers.com/
- http://schoolsucks.com/
- http://www.collegetermpapers.com/
- http://research-essays.com/
- http://antiessays.com/
- http://123helpme.com/
- etc.

So to treat all students consistently, investigate their work fairly, and enable convenient and fair electronic submission systems like UTCM we should detect plagiarism precisely by some powerful software like UT-Plag.

1-3 Main Problems

In the process of detecting plagiarism there're some important problems. The first problem is that we want to detect partial copies. We don't want to just find the exact match. As we said above shuffling is a popular approach for shrewd students to escape from detection of their plagiarized work.

Another problem is the accuracy of our method. Our software should be able to detect plagiarized documents accurately. False positiveness and false negativeness should be as minimized as possible. As you know it has a destructive effect if our software detects an innocent student's document as a plagiarized version of others' ones.

There is one more problem and that's the efficiency of our software. We should design and implement software with reasonable efficiency. It's so awful to wait for a long time to find out that there are some plagiarisms in the submitted collection or not.

As we mentioned before including web data is another difficult issue. We plan to add this feature to our system in future releases.

2 UT-Plag in Details

As we said before, UT-Plag 1.0 is mostly based on MOSS [4], a widely used plagiarism detection software. Here in this section we'll report some basic features of our software.

2-1 Desirable Properties

As a global rule every plagiarism detection method should meet these three properties [5]:

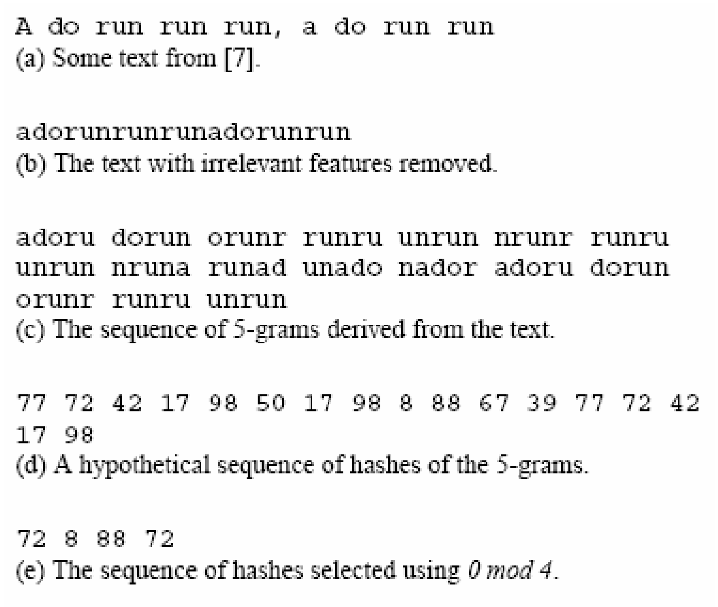
1. *Whitespace insensitivity*
   In matching text files, matches should be unaffected by such things as extra whitespace, capitalization, punctuation, etc. In other domains the notion of what strings should be equal is different—for example, in matching software text it is desirable to make matching insensitive to variable names.

2. *Noise suppression*
   Discovering short matches is uninteresting. Any match must be large enough to imply that the material has been copied and is not simply a common word or idiom of the language in which documents are written.

3. *Position independence*
   Coarse-grained permutation of the contents of a document (e.g., scrambling the order of paragraphs) should not affect the set of discovered matches. Adding to a document should not affect the set of matches in the original portion of the new document. Removing part of a document should not affect the set of matches in the portion that remains.

2-2 *k*-grams

A k-gram is a contiguous substring of length k. Divide a document into k-grams, where k is a parameter chosen by the user. For example, Figure 1(c) contains all the 5-grams of the string of characters in Figure 1(b). Note that there are almost as many k-grams as there are characters in the document, as every position in the document (except for the last $k - 1$ positions) marks the beginning of a k-gram. Now hash each k-gram and select some subset of these hashes to be the document's fingerprints. In all practical approaches, the set of fingerprints is a small subset of the set of all k-gram hashes. A fingerprint also contains positional information, which we do not show, describing the document and the location within that document that the fingerprint came from. If the hash function is chosen so that the probability of collisions is very small, then whenever two documents share one or more fingerprints, it is extremely likely that they share a k-gram as well. For efficiency, only a subset of the hashes should be retained as the document's fingerprints. In next sections we'll introduce our method of selecting fingerprints and compare it to some other methods. [5]



```
A do run run run, a do run run
```

(a) Some text from [7].

```
adorunrunrunadorunrun
```

(b) The text with irrelevant features removed.

```
adoru dorun orunr runru unrun nrunr runru
unrun nruna runad unado nador adoru dorun
orunr runru unrun
```

(c) The sequence of 5-grams derived from the text.

```
77 72 42 17 98 50 17 98 8 88 67 39 77 72 42
17 98
```

(d) A hypothetical sequence of hashes of the 5-grams.

```
72 8 88 72
```

(e) The sequence of hashes selected using *0 mod 4*.

Figure 1: Finger printing some sample text [5]

Here you can see that using *k*-grams provides "Noise suppression". When you choose *k* as the size of the atomic view of a document it means that each string match with size less than k would be ignored and could not be considered as a fingerprint. UT-Plag uses this approach and thus it will support "Noise suppression ". "Whitespace insensitivity" and "Position independence" both have been considered in UT-Plag. Later we'll discuss the algorithm in detail.

2-3 Karp-Robin String Matching [5]

Karp and Rabin's algorithm is one of the earliest version of substring matching techniques based on k-grams. Precisely we can say that the main goal of this algorithm is to find occurrences of a particular string *s* of length *k* within a much longer string like a document. The idea is to compare hashes of all *k*-grams in the long string with the hash of *s*. But to accomplish this we need to propose a "rolling" hash function. The question that may be popped to mind by now is this: What's a rolling hash function? If the hash for the i+1st *k*-gram is computed quickly (O (1)) from the hash of the ith k-gram, we say that the hash function is a rolling hash function. In our problem we treat a k-gram as a *k*-digit number in base *b*. In UT-Plag 1.0 we've used Java's hash function that has a rolling behavior as Karp-Rabin's algorithm suggests. Having $C_1 \ldots C_k$ as a *k*-gram (*k*-digit number in base *b*), H($C_1 \ldots C_k$) is its hash value that is computed according to following hash function:

$$c_1 * b^{k-1} + c_2 * b^{k-2} * \ldots + c_{k-1} * b + c_k$$

But as we mentioned above, according to Karp-Rabin's algorithm we need a hash function with rolling behavior to compute hash values as quickly as possible. Following shows the rolling behavior of suggested hash function:

$$H(c_2 \ldots c_{k+1}) = (H(c_1 \ldots c_k) - c_1 * b^{k-1}) * b + c_{k+1}$$

Since b^{k-1} is a constant, this allows each subsequent hash to be computed from the previous one with only two additions and two multiplications. Further, this identity holds when addition and multiplication are modulo some value (e.g., the size of the largest representable integer), so this method works well with standard machine arithmetic. As an aside, this rolling hash function has a weakness. Because the values of the C_i are relatively small integers, doing the addition last means that the last character only affects a few of the low-order bits of the hash. A better hash function would have each character C_i potentially affect all of the hash's bits. As noted in [5], it is easy to fix this by multiplying the entire hash of the first k-gram by an additional *b* and then switching the order of the multiply and add in the incremental step:

$$H'(c_2 \ldots c_{k+1}) = ((H'(c_1 \ldots c_k) - c_1 * b^k) + c_{k+1}) * b$$

Therefore, now we wish to compare all pairs of k-grams in the collection of documents. All-to-all nature of this comparison is the major difficulty. For example consider the problem of all-to-all matching on ASCII text. Using 4-bit integer hash values for each *k*-gram will eventuate to an index infrastructure much longer than the original document. A well known solution to this problem is "Fingerprinting".

3 Fingerprints

Fingerprinting means that we reduce the set of hash values of a document in order to have a smaller logical representation of that document. In our terminology we refer to these selected hash values as fingerprints of the document. But a new problem has been arisen: How to select document fingerprints?
Here we'll briefly investigate some different approaches to select document fingerprints.

- Select every i^{th} hash value of a document
  A simple but incorrect strategy is to select every i^{th} hash of a document, but this is not robust against reordering, insertions and deletions. In fact, prepending one character to a file shifts the positions of all k-grams by one, which means the modified file shares none of its fingerprints with the original. Thus, any effective algorithm for choosing the fingerprints to represent a document cannot rely on the position of the fingerprints within the document.

- Select all hashes of a document that are *0 mod p* [6]
  In this way fingerprints are chosen independent of their position, and if two documents share a hash value that is 0 mod p it is selected in both documents. Some commercial systems use this technique, but it needs some improvements to work well. However this approach is a popular method of document fingerprinting.

- Select n smallest hashes of all k-grams of a document
  By fixing the number of hashes per document, the system would be more scalable as large documents have the same number of fingerprints as small documents. This idea was later used to show that it was possible to cluster documents on the Web by similarity. The price for a fixed-size fingerprint set is that only near-copies of entire documents could be detected. Documents of vastly different size could not be meaningfully compared; for example, the fingerprints of a paragraph would probably contain no fingerprints of the book that the paragraph came from. Choosing hashes 0 mod p, on the other hand, generates variable size sets of fingerprints for documents but guarantees that all representative fingerprints for a paragraph would also be selected for the book. These two different approaches could be classified to fingerprinting as being able to detect only "resemblance" between documents or also being able to detect "containment" between documents.

- Winnowing [5]
  In each window select the minimum hash value. If there is more than one hash with the minimum value, select the rightmost occurrence. Now save all selected hashes as the fingerprints of the document.
  We've used Winnowing in UT-Plag 1.0. So we'll discuss this method in details in the next section.

4 Winnowing

Now, we'll discuss about details of winnowing as a fingerprinting algorithm. From [5] we mention an upper bound on the performance of winnowing, expressed as a trade-off between the number of fingerprints that must be selected and the shortest match that we are guaranteed to detect. Winnowing guarantees to find substring matches that satisfy two properties:

- Greater than *t* (guarantee threshold)
- Not smaller than *k* (noise threshold)

The constants *t* and $k \leq t$ are chosen by the user. We avoid matching strings below the noise threshold by considering only hashes of *k*-grams. The larger *k* is, the more confident we can be that matches between documents are not coincidental. On the other hand, larger values of k also limit the sensitivity to reordering of document contents, as we cannot detect the relocation of any substring of length less than k. Thus, it is important to choose k to be the minimum value that eliminates coincidental matches. Figures 3(a)-(d) are reproduced from Figure 1 for convenience and show a sequence of hashes of 5-grams derived from some sample text.

Given a sequence of hashes $h_1 \ldots h_n$, if $n > t - k$, then at least one of the h_i must be chosen to guarantee detection of all matches of length at least t. This suggests the following simple approach. Let the window size be $w = t - k + 1$. Consider the sequence of hashes $h_1 h_2 \ldots h_n$ that represents a document. Each position $1 \le i \le n - w + 1$ in this sequence defines a window of hashes $h_i \ldots hi + w - 1$. To maintain the guarantee it is necessary to select one hash value from every window to be a fingerprint of the document (Figure 2). (This is also sufficient [5]) In UT-Plag 1.0 we have used the following strategy as works well in MOSS, a practical and widely used plagiarism detection software:

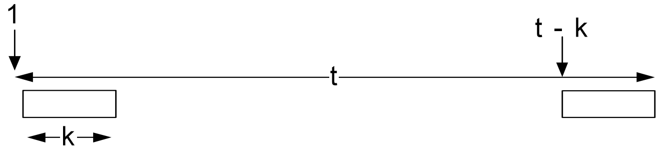


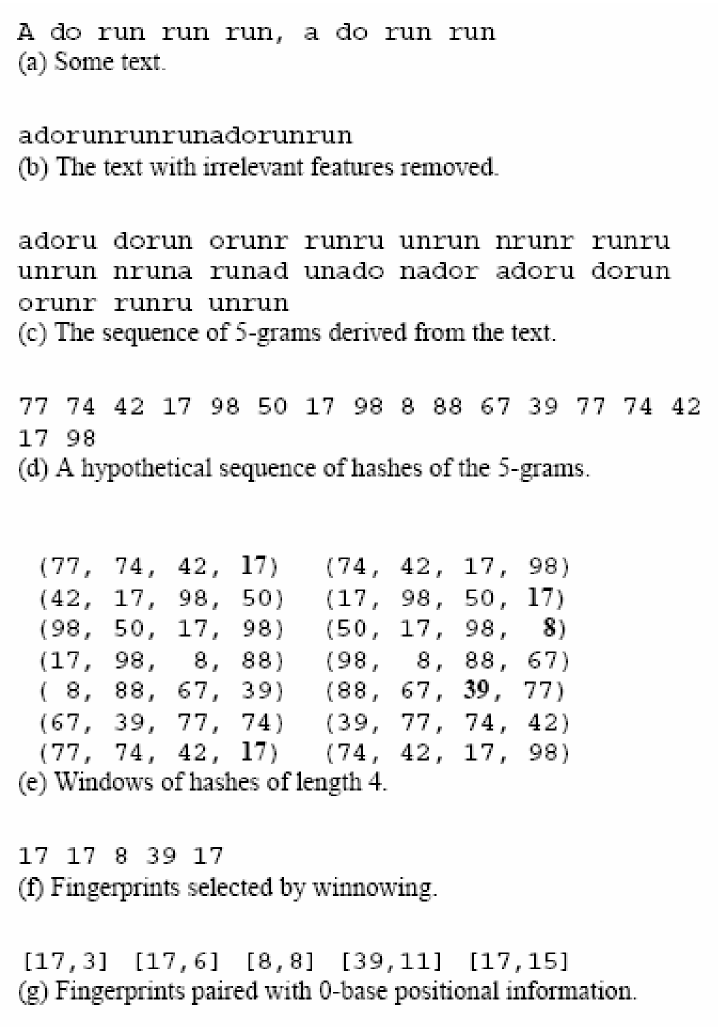
Figure 2: Window size to maintain the guarantee

Definition 1: Winnowing

In each window select the minimum hash value. If there is more than one hash with the minimum value, select the rightmost occurrence. Now save all selected hashes as the fingerprints of the document.

Figure 3(e) gives the windows of length four for the sequence of hashes in Figure 3(d). Each hash that is selected is shown in boldface (but only once, in the window that first selects that hash). The intuition behind choosing the minimum hash is that the minimum hash in one window is very likely to remain the minimum hash in adjacent windows, since the odds are that the minimum of w random numbers is smaller than one additional random number. Thus, many overlapping windows select the same hash, and the number of fingerprints selected is far smaller than the number of windows while still maintaining the guarantee. Figure 3(f) shows the set of fingerprints selected by winnowing in the example.

In many applications it is useful to record not only the fingerprints of a document, but also the position of the fingerprints in the document. For example, we need positional information to show the matching substrings in a user interface. In version 1.0 of UT-Plag we use an integer number to show the offset of found match in the document. An efficient implementation of winnowing also needs to retain the position of the most recently selected fingerprint. Figure 3(f) shows the set of [fingerprint, position] pairs for this example (in this figure the first position is numbered 0, but in UT-Plag 1.0 we used 1 to indicate the first offset).



```
A do run run run, a do run run
```
(a) Some text.

```
adorunrunrunadorunrun
```
(b) The text with irrelevant features removed.

```
adoru dorun orunr runru unrun nrunr runru
unrun nruna runad unado nador adoru dorun
orunr runru unrun
```
(c) The sequence of 5-grams derived from the text.

```
77 74 42 17 98 50 17 98 8 88 67 39 77 74 42
17 98
```
(d) A hypothetical sequence of hashes of the 5-grams.

```
(77, 74, 42, 17)  (74, 42, 17, 98)
(42, 17, 98, 50)  (17, 98, 50, 17)
(98, 50, 17, 98)  (50, 17, 98,  8)
(17, 98,  8, 88)  (98,  8, 88, 67)
( 8, 88, 67, 39)  (88, 67, 39, 77)
(67, 39, 77, 74)  (39, 77, 74, 42)
(77, 74, 42, 17)  (74, 42, 17, 98)
```
(e) Windows of hashes of length 4.

```
17 17 8 39 17
```
(f) Fingerprints selected by winnowing.

```
[17,3] [17,6] [8,8] [39,11] [17,15]
```
(g) Fingerprints paired with 0-base positional information.

Figure 3: Winnowing sample text

To avoid the notational complexity of indexing all hashes with their position in the global sequence of hashes of *k*-grams of a document, we suppress most explicit references to the position of *k*-grams in documents in our presentation. But now in UT-Plag 1.0 there's no graphical user interface to highlight the matches.

4-1 Expected Density

Definition 2: Expected Density

The density of a fingerprinting algorithm is the expected fraction of fingerprints selected from among all the hash values computed, given random input.

Now (from [5]) we mention the analysis about the density of winnowing, which gives the trade-off between the guarantee threshold and the number of fingerprints required. Consider the function C that maps the position of each selected fingerprint to the position of the first (leftmost) window that selected it in the sequence of all windows for a document. We say we are charging the cost of saving the fingerprint to the indicated window. The charge function is monotonic increasing that is, if p and q are the positions of two selected fingerprints and $p < q$, then $C(p) < C(q)$.

To proceed further recall that the sequence of hashes we are winnowing is random. We assume that the space of hash values is very large so that we can safely ignore the possibility that there is a tie for the minimum value for any small window size. Consider an indicator random variable Xi that is one iff the i^{th} window Wi is charged. Consider the adjacent window to the left W_{i-1}. The two intervals overlap except at the leftmost and rightmost positions. Their union is an interval of length $w+1$. Consider the position p containing the smallest hash in that union interval. Any window that includes p selects h_p as a fingerprint. There are three cases [5]:

1. If $p = i - 1$, the leftmost position in the union, then W_{i-1} selects it. Since $p \in W_i$, we know W_i must select a hash in another position, q. This hash is charged to W_i since W_i selected it, W_{i-1} did not select it, and the charge function is monotonic increasing. Thus in this case, $Xi = 1$.

2. If $p = i + w - 1$, the rightmost position in the union interval, then W_i selects it. W_i must be charged for it, as Wi is also the very leftmost interval to contain p. Again, $Xi = 1$.

3. If p is in any other position in the union interval, both W_{i-1} and W_i select it. No matter who is charged for it, it won't be W_i, since W_{i-1} is further left and also selected it. Thus in this case, $Xi = 0$.

The first two cases happen with probability $1 / (w + 1)$, and so the expected value of Xi is $2 / (w + 1)$. Recall that the sum of the expected values is the expected value of the sum, even if the random variables are not independent. The total expected number of intervals charged, and therefore the total number of fingerprints selected, is just this value times the document length. Thus the density is **$d = 2 / (w + 1)$**.

Now we compare the $0 \bmod p$ algorithm and Winnowing at the same density. That is, we take $p = 1 / d = (w + 1) / 2$. For a string of length $t = w + k - 1$ consider the event that the $0 \bmod p$ algorithm fails to select any fingerprint at all within it. (Recall that winnowing would never fail to do so.) We now compute the probability of this event for one given string.

Please note that for two overlapping such strings these events are not independent. Thus the probability we compute is not a good estimate for the fraction of all such substrings of a text that do not have a fingerprint selected using the *0 mod p* algorithm. Again we assume

independent uniformly distributed hash values. Also we assume large w. Thus, the probability that the guarantee fails in a given sequence of text of length *t*, i.e. that no hash in a given sequence of w hashes is *0 mod p*, is

$$(1-d)^w = \left(1 - \frac{2}{w+1}\right)^w \approx e^{\frac{-2w}{w+1}} = e^{-2+\frac{2}{w+1}} \sim 13.5\%.$$

4-2 Robust Winnowing

Winnowing, however, has a different problem. In low-entropy strings there are many equal hash values, and thus many ties for the minimum hash in a given window. To be truly local and independent of global position, it is necessary to take, say, the rightmost such hash in the winnowing window. But in the extreme case, say a long string of 0's with only one *k*-gram, nearly every single hash is selected, because there is only a single *k*-gram filling the entire winnowing window and at each step of the algorithm we must choose the rightmost copy—which is a new copy in every window. There is, however, an easy fix for this problem. In UT-Plag we've used the refined version of winnowing as follows [5]:

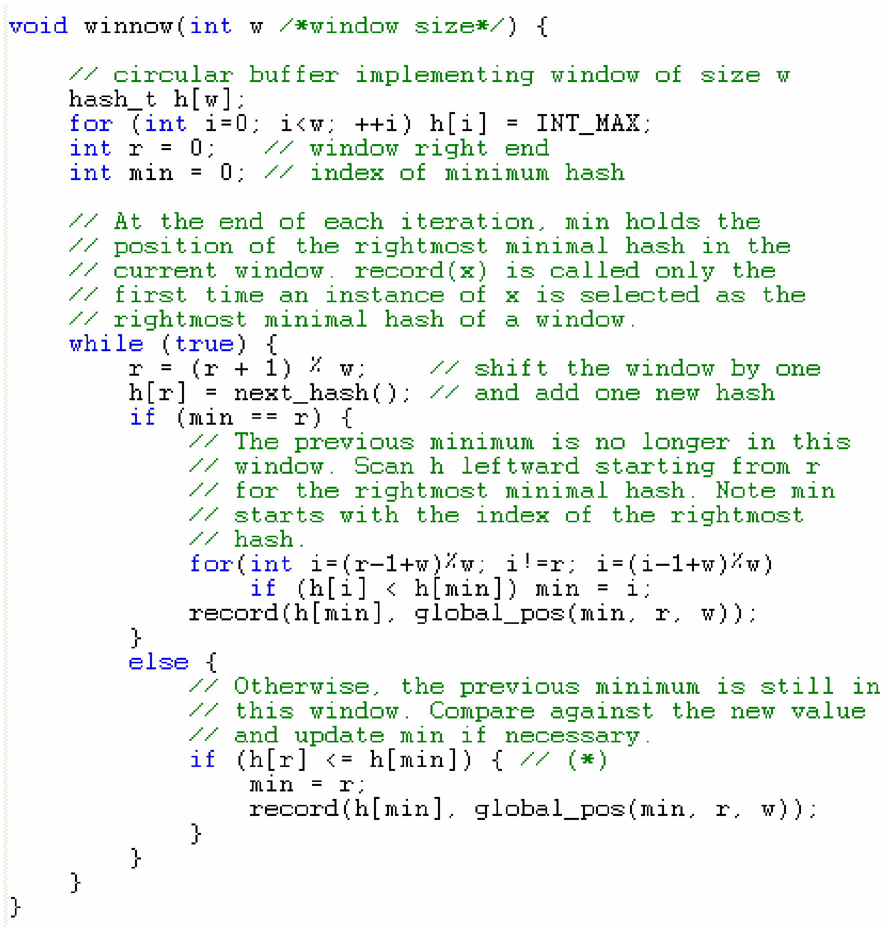
Definition 3: Robust Winnowing

In each window select the minimum hash value. If possible break ties by selecting the same hash as the window one position to the left. If not, select the rightmost minimal hash. Save all selected hashes as the fingerprints of the document.

Robust winnowing attempts to break ties by preferring a hash that has already been chosen by a previous window. This is no longer a local algorithm, but one easily observes that for any two matching substrings of length $t = w + k - 1$ we guarantee to select the same hash value and so the match is still found; we simply no longer guarantee that these fingerprints are in the same relative position in the substrings. However, the two fingerprints are close, within distance w − 1. This technique reduces the density on a string such as "0000 . . . " from asymptotically 1 to just 1/w, one fingerprint selected per window-length.

4-3 Engine Basic Source Code

Figure 4 shows the pseudocode of MOSS and UT-Plag 1.0 engine. We've implemented this engine pseudocode. We don't prefer to explain more about this pseudocode here. If you want to learn more about it read this pseudocode and its comments exactly.



```
void winnow(int w /*window size*/) {

    // circular buffer implementing window of size w
    hash_t h[w];
    for (int i=0; i<w; ++i) h[i] = INT_MAX;
    int r = 0;    // window right end
    int min = 0; // index of minimum hash

    // At the end of each iteration, min holds the
    // position of the rightmost minimal hash in the
    // current window. record(x) is called only the
    // first time an instance of x is selected as the
    // rightmost minimal hash of a window.
    while (true) {
        r = (r + 1) % w;    // shift the window by one
        h[r] = next_hash(); // and add one new hash
        if (min == r) {
            // The previous minimum is no longer in this
            // window. Scan h leftward starting from r
            // for the rightmost minimal hash. Note min
            // starts with the index of the rightmost
            // hash.
            for(int i=(r-1+w)%w; i!=r; i=(i-1+w)%w)
                if (h[i] < h[min]) min = i;
            record(h[min], global_pos(min, r, w));
        }
        else {
            // Otherwise, the previous minimum is still in
            // this window. Compare against the new value
            // and update min if necessary.
            if (h[r] <= h[min]) { // (*)
                min = r;
                record(h[min], global_pos(min, r, w));
            }
        }
    }
}
```

Figure 4: Pseudocode of UT-Plag 1.0 engine

4-4 Clustering the Results

We've one more feature that it's not yet supported in MOSS and most other well known systems. We cluster the results. We show the results in some distinct groups in which there's at least one document. If there're more than one document in a output cluster in means that probably these documents are plagiarized version of each other. This was a very nice suggestion from Dr. Farhad Oroumchian.

Our main goal was to understand the results easier. We start with original documents as individual clusters. Then we merge the two most similar clusters into one cluster and repeat this till the highest similarity between clusters drop below a threshold like *S.* We used vector space (ltc-ltc) to compute the similarity between the centroids of two clusters.

5 Future Works

UT-Plag is in its early stage. It's so young. We prepared version 1.0 of UT-Plag for final project of Intelligent Information Retrieval course in University of Tehran. But for future we plan to enhance it and add some helpful features. Followings are some of these future works:

- Design a graphical user interface that lets the user submit a collection of documents into system and view its results properly.
- Design some front end units to support some other widely used file types rather than plain text.
- Design some front end units to support some widely used programming languages files like Java, C/C++, Verilog HDL, Prolog, etc.
- Use some other information retrieval techniques to enhance UT-Plag, like signature extraction using 0 mod p techniques.

6 Conclusion

We have presented winnowing, a document fingerprinting algorithm that is both efficient and guarantees that matches of a certain length are detected. We have also presented the specific features of UT-Plag 1.0 and finally discussed our extra feature rather than MOSS and our future plan to enhance UT-Plag. Our engines pseudocode is also included in above pages.

7 Acknowledgements

Here we wish to thank Dr. Oroumchian for familiarizing us with information retrieval topics and his good suggestions to develop such a system. We should also thank MOSS designers for publishing most important topics about their widely used plagiarism detection software.

8 References

[1] http://en.wikipedia.org/wiki/Plagiarism, visited on 6/24/2005.

[2] Wilhoit.S, "*Helping students avoid plagiarism*", College Teaching, 1994, 42(4), 161-165.

[3] "*The Center For Academic Integrity, CAI Research*", Retrieved on 6/24/2005 from: http://www.academicintegrity.org/cai_research.asp

[4] http://www.cs.berkeley.edu/~aiken/moss.html, visited on 8/2/2005

[5] S.Schlemier et. al., "*Winnowing: Local Algorithms for Document Fingerprinting*", Retrieved on 6/24/2005 from: http://theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf

[6] Raphael. A. Finkel et. al., "*Signature extraction for overlap detection in documents*", Retrieved on 8/2/2005 from: http://crpit.com/confpapers/CRPITV4Finkel.pdf

About Authors

Naeem Esfahani

B.Sc. undergraduate student in computer engineering, University of Tehran (Year 4 : 2005)

Homepage: http://khorshid.ut.ac.ir/~naeem
Email: naeem A@T acm.org

Misagh Bagherian

B.Sc. undergraduate student in computer engineering, University of Tehran (Year 4 : 2005)

Homepage: http://khorshid.ut.ac.ir/~misagh
Email: misagh.bagherian A@T gmail.com