Plagiarism in natural and programming languages: an overview of current tools and technologies

Paul Clough – July 2000
Department of Computer Science, University of Sheffield

cloughie@dcs.shef.ac.uk

Abstract

This report discusses in detail methods of plagiarism and its detection in both natural and programming languages. The increase of material now available in electronic form and improved access to this via the Internet is allowing, with greater ease than ever before, plagiarism that is either intentional or unintentional. Due to increased availability of On-line material, people checking for plagiarism are finding the task increasingly harder. Techniques for detecting both plagiarism in natural and programming languages are discussed in this report to provide the reader with a comprehensive introduction to this area. Also provided are examples of common techniques used in natural language plagiarism.

Keywords

Plagiarism, copy detection, paraphrasing.

1   Introduction

When the work of someone else is reproduced without acknowledging the source, this is known as **plagiarism**. Probably the most frequent cases appear in academic institutions where students copy material from books, journals, the Internet, their peers etc. without citing references. Although sometimes intentional, there are many cases where students actually plagiarise unintentionally simply because they are not aware of how sources should be used within their own work. This problem is not just limited to written text, but also regularly found in software code where chunks are copied and re-used without reference to the original author/s.

Most students are now expected to submit written work and program assignments in electronic form. Although convenient and easier for both student and lecturer alike, the electronic version provides the student with an easier opportunity to plagiarise. With advanced word processors it is much easier to cut-and-paste large amounts of text to create a single work from a number of electronic sources including the Internet, electronic journals, books, newspapers and magazines etc.

Helping to make access to electronic versions of written text easier is the Internet. The Internet is growing at a remarkable rate and fast becoming a common resource for students. A recent study by IBM, Compaq and Alta Vista involved analysing more than 600 million unique pages[1] across a wide variety of subjects. It is probably true to say that a search on the Internet today for even the most obscure of topics will almost certainly return some relevant result. The Internet provides a global resource accessible by anyone from anywhere in the World that makes keeping track of electronic documents much harder than ever before and plagiarism much easier. However, the teacher's foe can also be their friend. Using Internet search engines such as Alta Vista and Yahoo, teachers can search for "unusual" phrases they find in student's work to identify potential sources (5).

---

[1] See: http://www1.compaq.com/pressrelease/ 0,1494,wp%7E14583_2!ob%7E29892_1_1,00.html

This report discusses various issues involved with plagiarism in both natural and programming languages. The aim is to provide the reader with a comprehensive introduction to plagiarism and increase their awareness of this problem in education. A number of old and new tools and techniques for detecting plagiarism are provided. However, the lists of tools are not exhaustive, but as full and accurate as possible up to the current time.

More examples of plagiarism in natural language are given over programming languages because of the focus of the author's interests. However, more detail is given to methods of plagiarism detection used in software code over text simply because of availability of more specific sources. Greater attempts appear to have been made in formalising plagiarism of programming languages than written text. This provides a more complete and reliable foundation upon which to base experiments for comparing various tools. No comparisons between methods of plagiarism detection are provided in this report. This is because the primary aim of the report is to introduce the techniques not apply them. Further more, obtaining examples of plagiarism is also very hard simply due to the confidentiality of student's work and hence reluctance by teachers to disclose examples.

2   Plagiarism in software code and written text

2.1   Introduction

Plagiarism of written text and software code can occur in many areas and is not simply limited to academic institutions. For example, in 1988 Barbara Chase-Riboud brought a plagiarism lawsuit against the film director Steven Spielberg and Dreamworks SKG over their latest film called Amistad (20). Chase-Riboud argued that the characters, scenes and other aspects of her book entitled Echo of Lions had been illegally copied without her permission. However, on February 9th 1988 Chase-Riboud dropped the charges and the film was released in December.

A further example, this time involving software copyright infringement took place before Mr Justice Jacob in The High Court of Justice, February 1994 (35). The case was between IBCOS Computers Ltd. and Barclays Mercantile Highland Finance Ltd. The case about a programmer working for one company and then the other but allegedly re-using previously developed software. The outcome of the trial was that copyright *was* infringed.

However, my interest in plagiarism lies within academic institutions as this is the area in which I work. Any establishment that teaches students is a potential source for plagiarism. Students may be under pressure to produce an assignment in a limited time scale, or may not want to put the necessary time and effort into the research required to create an original report. In some cases, students plagiarise simply because they do not know how to use sources properly. In other cases, students copy from each other or work together to complete assignments. This is known as **collusion**. This does not mean that students cannot help each other, but individual assignments should be written independently. A further description and discussion of the terms *plagiarism* and *collusion* can be found in (16). Some areas, such as computer science, also suffer from plagiarism in programming assignments not just written essays and reports. Many of the techniques and tools discussed in this report come from this area.

A closely related project undertaken by the Departments of Computer Science and Journalism at the University of Sheffield is METER (MEasuring TExt Re-use). The aim of the project is to investigate the re-use of Press Association text released to paying customers within the British press (7). This is not actual plagiarism as the newspapers pay for use of the PA material and are allowed to do whatever they want with the text. In (7), two types of text re-use are identified: 1) Verbatim re-use where text is copied from PA word-for-word and 2) Re-write where the original PA is restated using different words and sentence structure (similar to paraphrasing).

Perhaps the common area that relates both the METER work with plagiarism detection is **paraphrasing**. It occurs whenever a passage is re-written to restate the original, but using different words. An interesting review of **syntactic** versus **semantic** approaches to paraphrasing can be found in (15).

Also discussed in this report is a **copy detection system** which determines whether textual electronic documents stored in a repository have been copied and re-transmitted electronically, for example in Usenet articles or Web documents. Further areas of interest include **forensic linguistics** (38), **computational stylometry**, **authorship attribution** (31), **copyright detection in music/text** and **text summarisation**.

2.2   Penalties of Plagiarism and Copyright Law

In academia, plagiarism is dealt with severely. Within the Department of Computer Science in the University of Sheffield, the students sign a declaration form stating that the work they hand in is original. The penalty for plagiarism is to be given a zero mark for the assignment. Within other universities, the penalties range from withholding presenting a student with their degree to expulsion. The decision of whether a student has plagiarised is generally down to the lecturer and an external marker.

Figures for plagiarism reported within universities are not easily obtainable. This can be due to universities having to keep students' work confidential, not wanting to admit cases of plagiarism and whether or not figures are actually recorded for cases of plagiarism. In some cases plagiarism goes undetected because lecturers have too many students' assignments to mark or the class work is marked by more than one person. In our department, figures for detected cases of plagiarism are not recorded and published. A lecturer, however, did tell me of one year when a third of the class had copied from just a few sources. For an offence dealt with so seriously, it does not appear that adequate systems are in place to both teach students on avoiding plagiarism and automatically detecting cases of "unusual" similarity.

> *"Recently, two students handed in exactly the same essay. One stole the essay from a computer used by both. The two copies differed only in the name at the top. The culprit assumed that, given a class of 200 students in which the work is marked by five different tutors, the forgery would go undetected. That was, in fact, a pretty safe assumption since the discovery was entirely accidental."*
>
> The Guardian Higher, Gerard DeGroot, page 4H, Tuesday February 23, 2000.

From a legal point-of-view, proving plagiarism can be very hard. For a start, copyright can only be enforced if the plaintiff can prove that words were copied or trivially transformed (i.e. paraphrased). Even matching verbatim text between two sources does not prove plagiarism. If the two texts are written about the same topic, then it should not be a surprise that some information will be shared. For example names of people, places, technical terms or structure words of the language (i.e. English word classes such as prepositions, conjunctions etc.). In (16), Susan Finlay reports that in her work, independent texts have as much as 50% or more shared vocabulary overlap. With paraphrasing this is even harder because it must be proved the two suspected areas of plagiarism mean the same.

Within most of the references to plagiarism, people use the term "finding unusual patterns". The "unusual" pattern may include an unusually long sequence of matching characters between two texts, an unusually similar style of writing, or the same spelling mistakes, (often a very good indicator of copying). More "unusual" or "significant" finds include similarity in errors regarding truth of facts, the same order in

presenting the facts, or a similar dispersion of words and phrases. In (35), the verdict of infringement of software copyright was given on several factors. These included the defendant not being able to clearly explain the order in which the program code was written, the same spelling mistakes between the programs and identical comment headings. Also in common were redundant and unexplained code and various other features.

It is worth remembering that plagiarism is not just an academic, but also legal offence. The original author of the work is the owner and any re-use of the work must be with the owners' permission. Without this, the owner can sue the plagiarist in a court of law (in the US court this is violation of copyright). A further point to remember is that in most cases, when work is published, the publisher then owns the copyright to the text. This means they can bring copyright infringements against an author even when re-using their own material (in the case of Cambridge University Press - if the re-used text is used within a document being sold for personal financial gain).

2.3   Teaching students how to cite properly

One method of reducing unintentional plagiarism is by teaching students the correct way to cite and reference sources. There are a number of sources that provide examples of how to avoid plagiarism, one of the most helpful being (37). Although students may unintentionally plagiarise, some universities still class this as violation of regulations and deal in the same way as students who plagiarise intentionally. If this line of policy is adopted, students must be made aware of plagiarism and correct citation so they cannot claim ignorance.

An automated tool for helping teach students to avoid plagiarism is called the Glatt Plagiarism Teaching Program. This is discussed in 3.5.2.

2.4   Self plagiarism

A final area to consider is that of self-plagiarism. Issues surrounding this topic can be found in (39). As mentioned previously, when an author publishes a text, in most cases the publisher takes ownership of the copyright. Therefore if an author re-uses text with selling in mind, then the publisher has a right to bring copyright charges against the author. There is also the issue of whether it is ethical and fair academic practice to re-use material again unless only to summarise previous work. In one case (10), an academic researcher was found to re-use several previous papers when submitting to the EURO PAR 95 conference. The author did not cite himself or acknowledge his previous work in this area – an act of blatant plagiarism.

3   Plagiarism in written text

3.1   Introduction

Plagiarism of written text, whether intentionally or unintentionally, is increasing. An example supporting this view is the recent popularity of **on-line term paper mills**[2]. These are web sites that offer students access to previous academic term papers. The papers are available in a variety of languages and sometimes free to students wanting to be dishonest. John Hickman in (24) states there are an estimated 72 on-line paper mills now running on the Internet, a 279% increase from 28 at the beginning of 1997. Web sites such as these have now made it easy to obtain information from the Internet on a huge variety of subjects and copy this with no reference made at all to the author.

The problem is not just contained to America. A recent article in The Times[3] describes attempts by a couple to create the first Net essay bank in the UK. Students are paid £5 every time another undergraduate uses their work. The couple insistent the website is primarily for reference, not plagiarism.

---

[2] Examples include http://www.SchoolSucks.com, http://www.CheatHouse.com, http://www.Cheater.com and http://www.PhreePapers.com.
[3] Threat to block student cheats' Net essay bank, David Carter, page 9, Monday June 12th 2000.

Term paper mills and essay banks defend themselves by using their rights to free speech and dissemination of information. They will even deny plagiarism occurs involving students using their material. If students do use the material and plagiarise, the web sites argue they do not know how the text they provide will be used and that they cannot be blamed for any misuse.

3.2   Identifying plagiarism in written text

Identifying plagiarism by students in written text is normally down to the course tutor or students' supervisor. If they are familiar with the writing style of the student, they may be able to identify irregularities within the students' work compared to previous work, or unusual differences in the language or vocabulary used. Initially by identifying inconsistencies in authorship, potential plagiarism can be highlighted. Further searching can then be done to find the sources.

Some methods have already been discussed for detecting plagiarism in general. Further factors that could be used to distinguish authors of written text and detect plagiarism include:

- **Uses of vocabulary** – analysing the vocabulary used for an assignment against previous vocabulary could help determine whether a student had written the text. Finding a large amount of new vocabulary (in particular more advanced vocabulary) could help determine whether a student legitimately wrote a text.
- **Changes of vocabulary** – if the vocabulary used changes significantly within a single text, this can indicate a possible cut-and-paste plagiarism.
- **Incoherent text** – if the flow of a text is not consistent or smooth, this can could indicate the author has either not written with thought or consistency or that part of the text is not their own work.
- **Punctuation** – it is unlikely that two writers would use punctuation in exactly the same manner (even if taught the same English grammar).
- **Amount of similarity between texts** – there will always be a certain amount of similarity between texts written about the same topic such as names, domain-specific terms etc. However, it is unlikely that independently written texts would share large amounts of the same or similar text.
- **Common spelling mistakes** – an obvious feature to use. It is very unlikely that independently written texts would have the same spelling mistakes, or same number of mistakes.
- **Distribution of words** – it is unlikely that the distribution of word usage throughout independent texts would be the same. For example, having the same parameter values for a statistical distribution used to describe term usage.
- **Syntactic structure of the text** – it may be indicative that plagiarism has occurred if two texts share exactly the same syntactic structure. It is likely that the most common syntactic rules used by separate authors would be different.
- **Long sequences of common text** – it is unlikely that independently written texts (even about the same subject) would share long sequences of consecutive characters or words in common.
- **Order of similarity between texts** – if the order of matching words or phrases between two texts is the same in both texts this may indicate plagiarism. Although taught to present facts in a certain manner (e.g. introduction, body then conclusion), it is less likely that the same facts would be reported in the same order.
- **Dependence on certain words and phrases** – an author may prefer using particular words or phrases. Consistent use of these words and phrases in a text written by someone else with different word preferences may indicate plagiarism.
- **Frequency of words** – it is unlikely that words from two independent texts would be used with the same frequencies.
- **Preference for the use of long/short sentences** – authors may without knowing have a preferred sentence length that would be unusual combined with other features.
- **Readability of written text** – using metrics such as the Gunning FOG index, the Flesch Reading Ease Formula or the SMOG index can help determine a readability score for a particular author. It is again unlikely that different authors would share the same score.

- **Dangling references** – if references appear in the text but not in the bibliography, this may indicate a cut-and-paste plagiarism where the author has not also copied the references.

Determining an author's style would involve using more than one of the previous features to create an accurate profile – a set of characteristics identifying a writing style. A good survey on writing style and authorship attribution can be found in (31).

What constitutes style is an issue that computational Stylistics and Authorship Attribution address. One source (38) identifies some simple features that can be used for **statistical techniques**. These involve counting the frequency with which certain features from gathered text occur. They can be used to identify author styles and include:

- the average length of sentences (words),
- the average length of paragraphs (sentences),
- the use of passive voice (expressed as a percentage),
- the number of prepositions as a percentage of the total number of words,
- the frequency of "function words" used in each text.

There are, however, some words and phrases that are more likely to appear in common between texts written about the same topic, even if written independently. These may include:

- Names,
- Dates,
- Locations,
- Domain-specific terms,
- Common knowledge terms.

However, if any new information that is not common knowledge is introduced without reference (i.e. if not original ideas) then this is also plagiarism.

3.3   Examples of plagiarism

The following examples of plagiarism are taken from a guide for undergraduate Chemistry students from the University of Kentucky (45). It describes the most common forms of plagiarism in natural language:

1. **Copying directly from the source** – this means to take words or sentences verbatim from the original source (with or without footnotes). If the source is copied, it should be put in quotes and cited. The source (45) suggests the best method for students to overcome this is to read the source and write in their own words, although avoiding the other types of plagiarism listed below (particularly paraphrasing). An example of this is taken from (48):

   Source:
   "How important is our power of nonanalytical thought to the practice of science? It's the most important thing we have, declares the Princeton physicist historian Thomas Kuhn who argues that major breakthroughs occur only after scientists finally concede that certain physical phenomena cannot be explained by extending the logic of old theories. Consider the belief that the sun and the planets move around the earth, which reigned prior to 1500. This idea served nicely for a number of centuries, but then became too cumbersome to describe the motions of heavenly bodies. So the Polish astronomer Copernicus invented a new reality that was based on a totally different `paradigm' or model--that the earth and planets move around the sun" (Hoover, 124).

   Plagiarised Version:
   Non-analytic thought is considered very important to the **practice of science** by **Princeton physicist historian Thomas Kuhn** who claims **that major breakthroughs** happen only when **scientists finally concede** that some **physical phenomena** defy explanation by **extending the logic of old theories**. One idea which **served nicely** for many centuries but then **became too cumbersome** was **the belief that the sun and**

planets revolved **around the earth**. This was held **prior to 1500** until **Copernicus invented a new reality: the earth and planets move around the sun**.

The words in bold are copied directly from the source. Note that not all words are copied, but it is the lack of quotation marks, reference and repeated use of exact wording and sentence structure that make this plagiarism. This is similar to paraphrasing (described below).

2. **Rewording a sentence (paraphrasing)** – an original sentence is rewritten in a copier's own words, but still no use of quotation marks or referencing is used. This form of plagiarism can take several guises and these are discussed further in section 3.3. Examples of different permutations of a source that are classed as paraphrasing and deemed plagiarism are shown below, taken from (45).

Source:
"Those complexes that contain unpaired electrons are attracted into a magnetic field and are said to be paramagnetic, while those with no unpaired electrons are repelled by such a field and are called diamagnetic"

Plagiarised versions:
- "Complexes that contain unpaired electrons are those that are attracted to a magnetic field. These are called paramagnetic, while those with no unpaired electrons are repelled by a magnetic field and are said to be diamagnetic."
- "Those complexes that contain paired electrons are repelled by a magnetic field and are said to be diamagnetic, whereas those with no paired electrons are attracted to such a field and are called paramagnetic."
- "Compounds that have unpaired electrons are attracted to a magnetic field and are called paramagnetic. Compounds with no unpaired electrons are repelled by this field and are said to be diamagnetic."

3. **Submitting someone else's work** – an obvious example of plagiarism. This can include copying someone else's homework, having identical passages in a laboratory book, copying or using material from previous years or submitting other's computer input/output as your own (42).

4. **Failing to reference/footnote source material** – as new facts are presented to people not familiar with the field, a footnote should be presented to reference the source material (42). Example text to be referenced includes results from other researchers, verbatim or paraphrased sentences, concepts and ideas, drawings and graphs, laboratory procedures and anything else not the plagiarists own work.

5. **The Internet "pastiche"** – the copying of a collection of paragraphs from a variety of electronic sources and pasted together in a word processor to form a report (13). Different sources may have different styles, but this can be easily overcome using paragraphs from articles in the same domain or from the same author. The easiest "pastiche" involves copying text from the Internet.

There may be other types of plagiarism, but these are just some examples of the most common. Of the above, perhaps the hardest to detect is paraphrasing. This is discussed in more detail below.

3.4   Paraphrasing - further examples of plagiarism

A paraphrase is simply the **re-write** of an original piece of written or oral text in the copier's own words (2). To paraphrase successfully involves understanding the concepts presented in the text to enable these to be restated using the copier's own **vocabulary**. A "good" paraphrase will use the copier's own writing style and voice to make the whole document flow more smoothly. There are a number of different strategies that can be used to paraphrase a piece of text (1) and include the following:

1. **Using appropriate synonyms** – a common form of paraphrase where the writer will use a **thesaurus** or **dictionary** to change either one word for another or one for many that express the same meaning (also called **word-for-word substitution**). For example:

Source:
> "He was brutally attacked and murdered"

Paraphrase:
> "The man was violently set upon and murdered"

2. **Changing the sentence type** – the original sentence can be altered by using different **transition words**. Examples of transition words include "but", "so", "because", "in addition to", "due to", "even though" etc. This can change the pattern of the sentence. For example:
   Source:
   > "Technology can cause a disaster"

   Paraphrase:
   > "A disaster is possible due to technology"

3. **Changing the order of a sentence -** the order in which ideas are expressed in the original document can be changed by permuting words or phrases within the sentence, for example:

   Source:
   > "Technology can improve the quality of life if we plan carefully for the future."

   Paraphrase:
   > "If we plan carefully for the future, technology can improve the quality of life."

4. **Reducing a clause to a phrase** – the clause has both a subject and verb, but can be reduced to a phrase that contains just one or the other. This technique is normally used with part-of-speech change or sentence re-ordering. For example:

   Source:
   > 1) Sociology is a "part of the science of human behaviour 2) that attempts to find the causes and effects 3) that are a part of social relations among persons. In addition, sociology studies the causes and effects of inter-communication and interaction among persons and groups"

   Paraphrase:
   > 2) Attempting to find the causes and effects 3) of social relations among people, 1) sociology, a part of the science of human behaviour, 3) studies the inter-communication and interaction among persons and groups.

Notice above that the order of the sentences have been changed. The "–that" clauses have been changed and sentence 3 has been split into two and used in different places of the paraphrase.

5. **Changing the part-of-speech** – this involves changing the word class depending on the context of the word. For example, a noun can be changed to a verb (called **nominalisation**) , an adjective to a noun, a verb to a noun, an adverb to adjective etc. Also the inflection of a verb may change if the tense of the original document is altered. For example:

   Source:
   > Attempting (verb) to find the causes and effects of social relations among people, sociology (noun), a part of the science (noun) of human behaviour, studies (verb) the intercommunication and interaction among persons and groups.

   Paraphrase:
   > In an attempt (noun) to find the causes and effects of social relations among people, a sociologist (new noun), who is a scientist (new noun) of human behaviour, undertakes studies (noun) of the intercommunication and interaction among persons and groups.

6. **Making abstract ideas more concrete** – although the concrete version is re-written, this is still plagiarism. For example:

Source:
> The creation of an overall design for a complete computer system is the responsibility of a systems analyst; whereas, the implementation of the design plan is often the duty of the computer programmer.

Paraphrase:
> The systems analyst designs the entire computer system, and the computer programmer makes the proposed system work.

Worth mentioning for its relation to paraphrasing is **text summarisation**. In one journal article (25), the various types of operations involved in summarisation are very similar to those listed above. The operations identified by the authors in generating a text summary using a cut-and-paste technique are:

- Sentence reduction
- Sentence combination
- Syntactic transformation
- Lexical paraphrasing
- Generalisation/specification
- Sentence reordering

3.5 On-line Internet Plagiarism services

As mentioned previously, the amount of accessible text available in electronic form on a wide variety of subjects is growing. This is due to rapid growth of the Internet and term paper mills that has already given rise to greater cases of cut-and-paste plagiarism from web sources. Due massive volumes of information, detecting whether a document has been plagiarised from another source is time-consuming. Web search engines can be used to find potential sources, but checking for plagiarism within in every piece of student work is unfeasible.

However, there are now services and tools available to teachers and students to detect plagiarism in submitted work against the Internet. Three such on-line plagiarism services include: 1) Plagiarism.org, 2) Integri-guard and 3) EVE2. None of the services discuss in detail implementation of their detection system, hence the vagueness of these descriptions.

3.5.1 Plagiarism.org[4]

This is the perhaps the largest on-line plagiarism detection service available[5]. The service has been developed to deal with an increase in on-line term paper mills and growth in Internet-available information.

The service allows a tutor to register their class with the system. When students finish their report, they upload it to the Plagiarism.org website. The system then creates a different representation of the documents that can be statistically checked against previous term papers, material from the WWW and reports from other universities. The reports are then emailed to the course instructor and the potential plagiarised parts highlighted.

Plagiarism.org give each document a "fingerprint", a representation that they claim is immune to students changing words, adding sentences, or reorganising their documents. This detection system they call iThenticate ©. Plagiarism.org emphasis that they have tested and optimised the system to deal with each plagiarism technique. The product is claimed to be able to detect plagiarism even if a student replaces half of

---

[4] This can be found at http://www.plagiarism.org.
[5] This is the author's speculative view from surfing the Internet.

the words in a paper with synonyms (they still call this text an altered version, not a different text). Another form of plagiarism they discuss is sentence or paragraph addition (again they question how many sentences must be added until the document becomes original). Plagiarism.org claim that the iThenticate© algorithm is able to detect embedded paragraphs from multiple sources and that this actually helps them to detect a plagiarised document.

Plagiarism.org maintain a multi-gigabyte database of documents from the Internet, term paper mills and previous student assignments. They use an undisclosed technique for narrowing the search enabling a more detailed analysis of potential plagiarised documents (Figure 1).

Plagiarism.org discuss using a **plagiarism index** ranging from 0 to 1. The value 0 means no similarity and 1 an exact copy. This value is compared with the probability of word substitution (example graph for one document given in Figure 2 - the horizontal line indicates the point at which plagiarism becomes originality – in this case after about 50% for probability of word substitution).
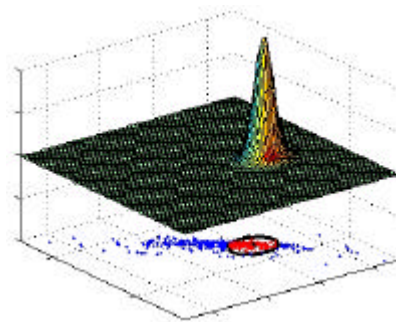

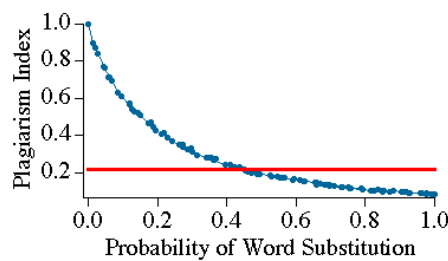*Figure 1 – narrowing the search*

Figure 2 - the document plagiarism index

Comparing this document plagiarism index with all the documents in the collection allows the plagiarised document to be clearly seen (Figure 3).
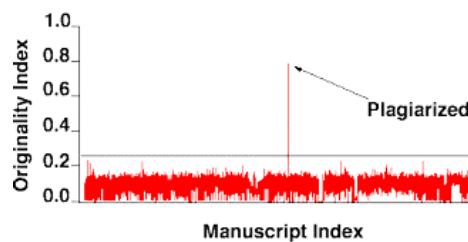

Figure 3 – the plagiarised document stands out among the rest

Plagiarism.org allow a free trial of the system but full use must be paid for (negotiable upon request).

3.5.2 IntegriGuard[6]

This on-line service is provided for educators to check for potential plagiarism. The instructor registers with the service and obtains a unique reference number. This is then used in the registration of the students who obtain a username and password. The students submit their work via a web page[7] and IntegriGuard then analyses the text. The results are emailed to the instructor.

IntegriGuard do not give any information on the internal implementation of the system but do mention that a statistical algorithm is used to select the most likely match if more than one matching sentence is found.

The cost of IntegriGuard is based upon an annual subscription fee of $59.40 ($4.95 per month).

3.5.3 EVE2[8]

EVE2 processes documents in plain-text format and uses "advanced searching tools" to locate potential sources of plagiarism from the Internet. The system returns links to these pages, a task that manually may take a significant amount of time. Once finished searching, EVE2 provides a report on each plagiarised paper including the percent plagiarised and an annotated version highlighting plagiarism in red.

The system is supposed to employ a technique that prevents too many matches being found and a method for finding suspect search sites that does not search every site on the Internet.

The cost of EVE2 is $19.99 and a 15-day trial copy is also available. EVE2 also claims to perform better than Plagiarism.org at a much faster rate.

3.6 Other text plagiarism detection tools

Apart from the Internet-based plagiarism tools, there are also a number of commercial and experimental tools available. Included in these are a copy detection mechanism for digital (40) called SCAM, a system for authorship attribution/plagiarism called Copycatch (53) and a tool for creating a profile of the words that an author generally uses called WordCheck.

3.6.1 CopyCatch[9]

CopyCatch is primarily aimed at University and College departments for automated detection of collusion and plagiarism. The program has evolved from a set of linguistic analysis tools developed to assist forensic linguists in looking at short texts. This work stems from the forensic linguistics group at Birmingham University headed by Professor Malcolm Coulthard.

The program is used to determine whether any plagiarism has taken place between a group of individuals. Similarity between documents is measured in terms of lexical similarity between pairs of texts allowing investigation of the vocabulary and phrasal similarities between the texts (53). The program reads in a set of documents and compares each with every other to show the number of words in common. The percentage similarity is based upon word occurrences as opposed to vocabulary occurrences based on the hypothesis that the replacement words or phrases will not be consistently applied so traces of the original will still appear. Where identical or similar frequencies appear, the likelihood of copying increases. Of particular

---

[6] http://www.integriguard.com.

[7] http://www.paperbin.com.

[8] http://www.canexus.com/eve/abouteve.shtml.

[9] http://www.copycatch.freeserve.co.uk.

interest are the **hapax legomena** words (those only appearing once in the text). The authors express that any more than around 40% overlap of the content carrying words is "normal" and that anything over 70% requires further analysis. In (16), Susan Finlay demonstrates that independently written texts can have overlaps of 50% or more where similar vocabulary is used, but is used in different proportions by the writers. She shows that in these cases, the hapax legomena level can be used to discriminate between potential plagiarism and similarity due to chance.

> *"CopyCatch in fact makes use of the inherent structure of a coherent text as follows:*
> *A 1000 word essay will be roughly split into 550 function words and 450 lexical words.*
> *The 450 lexical words will be based on a vocabulary of between 200 and 300 words.*
> *Taking the lower possibility 70% of the 200 words will only be used once.*
> *To change one text into another by using synonyms would therefore require the changing of 140 words to do a total job.*
> *In comparing independently produced texts the number of hapax words is frequently in single figures, so there is much work to do and still be making sense."*
>
> *Email from David Woolls, Wednesday April 12,1999.*

The results of the program are presented in a tabular form showing a breakdown of the tokens of each text and the amount of overlap. The hypothesis CopyCatch uses is that if the ratio of infrequently occurring words distinguishes texts then the words themselves would be different even when answering the same question. Students answering the same question have showed this where the hapax legomena similarity is very low.

CopyCatch also uses an **abridgement tool** based upon **lexical cohesion** to reduce the amount of text compared by removing the "irrelevant" sentences but still preserving the document meaning.

3.6.2   Glatt Plagiarism Services Inc[10]

Glatt Plagiarism Services Inc. provide three basic software tools to help deter and detect plagiarism:

- Glatt Plagiarism Teaching Program,
- Glatt Plagiarism Screening Program,
- Glatt Plagiarism Self-Detection Program.

The **teaching program** is a tutorial designed to illustrate what constitutes plagiarism, provide examples of direct and indirect plagiarism and demonstrate how to reference properly to avoid plagiarising other works. The program is aimed at students and also demonstrates how to paraphrase without plagiarising.

The **screening program** is aimed at teachers to detect plagiarism in student's work. The program works on the basis that people have their own writing style and linguistic pattern. The program uses a technique developed by journalist Wilson Taylor (43) called the "cloze procedure". This technique addresses the questions of redundancy in the English language and works by deleting every nth word from a written passage. The Glatt screening program deletes every 5th word from a suspect work and the student is asked to

---

[10] Available from http://www.plagiarism.com.

supply the missing words. As each person is assumed to know their own writing style instinctively, the time taken to fill the blanks should be relatively quick. The screening program uses a combination of the correct number of words for the blanks, time taken to fill the blanks and "various other factors" to compute a final Plagiarism Probability Score. Glatt claim within the 10-year period of development, there have been no wrong accusations due to extensive testing in the classroom.

The final Glatt tool is the **self-detection program** that is for individuals to gain an insight into their own writing style to help avoid plagiarism. The program is available on-line[11] or to buy. The program is very simple and asks the user to enter a piece of text. Each 5th word is then removed and the user must fill in the blanks. The resulting percentage of correctly entered words is presented as the final score.

3.6.3   WordCheck Keyword Software[12]

The WordCheck software is a tool designed to help detect plagiarism and protect intellectual property as a means of identifying how information has been misused or stolen. This program seems perhaps the simplest of all those reviewed. All the tool does is count the number of occurrences of each word within a set of documents. This information can be used to determine how well they "match". The approach is likened to one used in a simple Information Retrieval (IR) system. The program is shown in Figure 4.
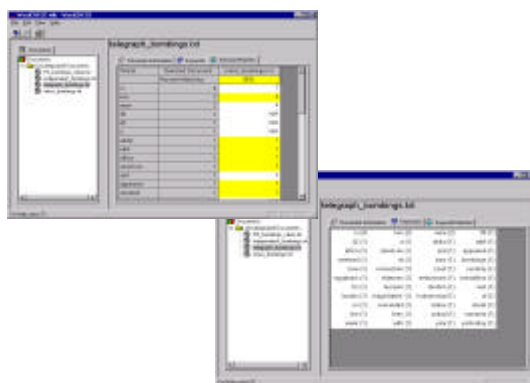

Figure 4 – document keyword frequencies and percentage matches with other documents

3.6.4   YAP3[13]

The YAP (Yet Another Plague) series of tools are primarily aimed at detecting software plagiarism. A more detailed analysis of YAP1 and YAP2 can be found in 4.1.4. Michael J. Wise, previously working in the Computer Science Department at the University of Sydney, now working with the Centre for Communications Systems Research at Cambridge University has developed these series of tools over a number of years. The most recent version of YAP, YAP3 (49) has been adapted for use with the English language, although only limited research and testing has been performed. The main problem being limited amount of source text available for potential plagiarism analysis.

YAP3 is the third extension of YAP that makes use of a novel algorithm called **Karp-Rabin Greedy-String-Tiling** (or RKS-GST) (51) used to compare pairs of strings allowing for **transposition** of substrings. The algorithm is similar to the **Longest Common Subsequence** (LCS) - (8) or (17) **and Levenshtein distance** algorithm, both solutions for comparing the similarity between two strings. The problem with both methods, however, is that they are **order preserving** meaning that if a complete block of text is moved the

---

[11] http://www.plagiarism.com/self.detect.htm.

[12] http://www.wordchecksystems.com/

[13] Michael Wise's YAP web page: http://www.ccsr.cam.ac.uk/~mw263/YAP.html

algorithms treat this as a number of single line differences rather than a single block move. Both Heckel (22) and Tichy (44) give alternate solutions but both have problems (50).

The YAP approach does not attempt a full parse of the target language, but compares token strings made up from keywords drawn from the target language's lexicon. This is particularly useful for English, as obtaining a full parse of the language is practically impossible.

The system works in the following manner:

- A tokeniser-generator is used to parse a number of texts and then determine the lexicon used to generate the token strings.

- A first lexicon-generating parse eliminates all numbers, words consisting of one or two letters, proper nouns and all "common" words (about 150 as defined in a stoplist). The stoplist is extended using simple stemming (for example if "keep" is in the stoplist, then so to will be "keeps" and "keeping" etc.).

The remaining words are stemmed using the PC-Kimmo recogniser (version 1.08) and the Englex10 rule and lexicon sets[14].

The tokeniser-generator and resulting lexicon were applied to a series of essays using YAP3, although Wise reports no cases of plagiarism detected.

3.6.5 COPS

The COpy Protection System (COPS) is an experimental working prototype of a **copy detection system** that can be used in a digital library (6). The work is a small part of the Stanford Digital Library Project[15], aimed at providing a sophisticated information repository. The COPS part of the project is to detect exact or partial copies of documents in the library due to the ease with which information can be copied and plagiarised. The system is technically referred to as **a copy detection service** and deals only with copy detection within textual documents in either a Tex, DVI or troff format. The documents are converted to straight ASCII before registration and similar document detection (see Figure 5). The system looks for documents with significant overlap as well as exact copies.
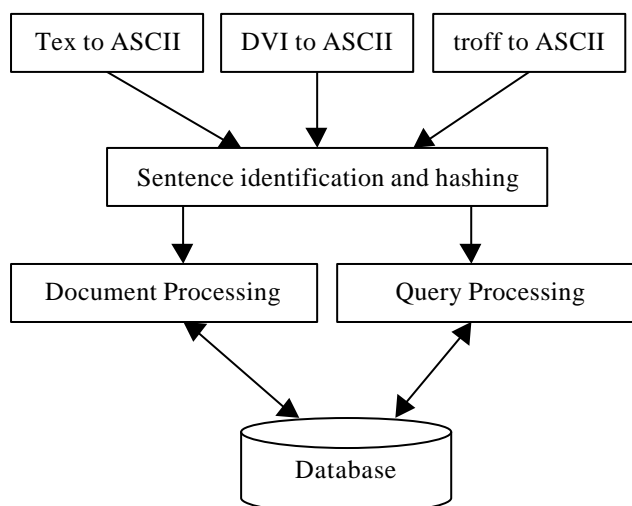

Figure 5 – the main modules of the COPS architecture

---

[14] Both PC-Kimmo and Englex 1.0 are available from the Consortium for Lexical Research, http://clr.nmsu.edu and a postscript catalogue of all tools is available form CLR/catalogue.ps
[15] See the project at: http://www-diglib.stanford.edu

The documents are broken into sentences (called units) and these are further grouped together in sequences of sentences called chunks. The sentences are stored in a registration server that is simply a large hash table using a "standard" hashing algorithm. The document chunks are compared with those of other documents in the repository to check for overlap. If the documents share a pre-set number of sentences then a violation is flagged. A human then looks at the violation to determine the problem.

The system has much in common with Siff, a program designed by Udi Manber for finding similar files in a file system (28). Manber's technique involves selecting a few words as "anchor" points and then computing checksums of a following window of characters for comparison.

The possible violations that can occur between documents include plagiarism of a few sentences, exact replication of the document, and stages in between.
Brin, Davis and Garcia-Molina (6) admit that plagiarism is particularly difficult to test for and would require human decision. COPS implements Ordinary Operational Tasks (OOTs) to carry out tests for plagiarism, subset and overlap that can be implemented efficiently.

Preliminary testing of COPS used 92 Latex, DVI and ASCII technical documents consisting of approximately 7300 words and 450 sentences. The documents formed 9 topical sets and the results showed that COPS managed to cluster most of the documents correctly, although discrepancies were found with sentences that were common to all documents. The authors provide improvements to lessen the extent of the problems.

3.6.6   SCAM

SCAM (Stanford Copy Analysis Mechanism) was developed from the experiences gained from building COPS by Narayanan Shivakumar and Hector Garcia-Molina - (40) and (41). The system is designed for detecting plagiarism, copies, extracts and strongly similar documents in digital libraries. The main objective for building the system was as a method for supporting the copyright of documents stored in a library to detect cases of unauthorised copying. SCAM came into the spotlight in 1995 when the system correctly reported 13 cases of possible plagiarism (verified by the original authors) by an academic researcher initially sparked by similarities in a submission to the EURO PAR 95 conference (10).

The main difference of SCAM from COPS is that SCAM is a word-based scheme, whereas COPS was sentence-based. The problem with simply comparing sentences is that partial sentence overlaps are not detected. Further problems of COPS included detecting sentences with figures, equations and abbreviations confusing the system and checking for overlap involved many random probes into the hash table causing an efficiency problem.

The documents are divided into words (units) and these are grouped to form chunks. The chunks are inserted into the repository in an inverted index structure and used to compare with new document arrivals (see Figure 6). SCAM uses words as chunks for comparison allowing the system to detect partial sentence overlap. SCAM uses a derivative of the **Vector-Space Model** to measure similarity between documents. This is a popular IR technique and operates by storing the normalised frequency of words within the document as a **vector**. The vectors are then compared for similarity using a measure such as the vector **dot product** or **cosine** similarity measure and a resulting value, if exceeding a pre-defined threshold, is flagged. Problems with both methods given in (41) gave rise to the development of a new similarity measure called the **Relative Frequency Model** (RFM).

The RFM uses the relative frequencies of words as indicators of similar word usage and combines this with the cosine similarity measure. Initially a closeness set is defined that indicates all words that have a similar number of occurrences in two documents. This set is used in the measure for a subset test that determines whether D_1 is a subset of D_2. The test is asymmetric therefore the similarity set is the maximum value of the subset of D_1 compared with D_2 and subset value of D_2 compared with D_1.
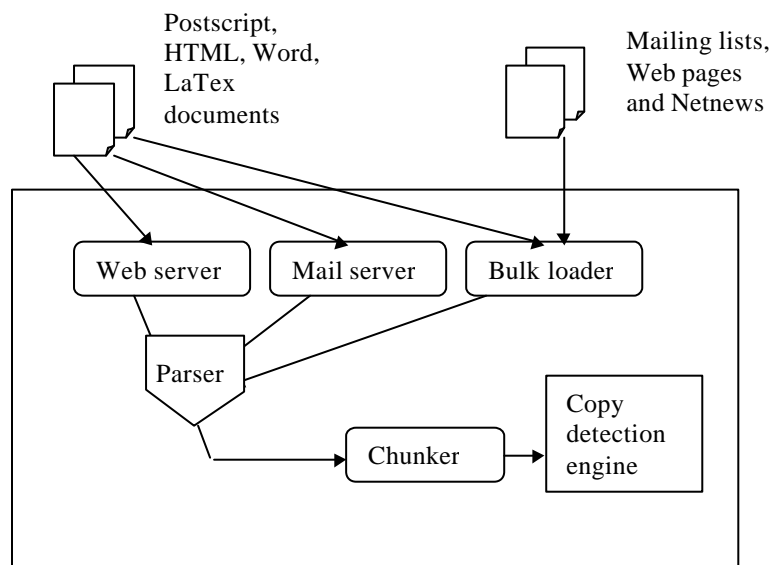
Figure 6 - The functionality of SCAM

Testing was performed on 1233 Netnews articles (with headers removed) and compared against each other for overlap. The overlap results were then compared with values for COPS and using a standard IR technique. The overlap values were then compared with manually examined texts to determine cases of 1) Plagiarism, 2) Subset 3) Copy and 4) Related. The texts could fall into more than one class. The overlap values were then compared with the human judged results and the results indicate that SCAM worked better than IR methods for separating document pairs and had a lower number of false positives (those texts that are plagiarised but are not detected by the system). SCAM also performed better than COPS in detecting overlaps although in some cases the number of false positives for SCAM was higher than for COPS.

3.6.7 CHECK

The final example of a plagiarism detection tool is a document plagiarism detection system called CHECK (42). Similar to previous copy detection systems, CHECK maintains a database of registered documents that are used for comparison against a new document. However, CHECK is different in that two problems are addressed:

1) With previous systems, many time-consuming and unnecessary comparisons are made between documents that are unlikely to be sources of plagiarism because of differences of content,
2) Security of previous copy detection systems is weak because minor modifications can be made to each sentence to by-pass the detection mechanisms.

CHECK is unusual in that Information Retrieval (IR) techniques are applied first to filter out likely plagiarism candidates. This IR process is applied recursively on different levels of granularity from sections, subsections, paragraphs to finally sentences. Comparison between documents is based upon keywords that are supposed to identify the **semantic meaning** of the document.

The authors of CHECK also recognise the differences between comparing plagiarism of natural and programming language (42). They comment that computer programs are well structured and preserve the parse-tree of the original program even if modified. However, plagiarism in written text is much harder as a document can preserve the semantics of the original, but undergo many more changes than a computer program (i.e. the parse tree is changed).

CHECK incorporates additional information (weighted keywords) into the document parse tree to capture a better representation that is more resistant to simple document modifications. At the time of writing, CHECK only recognised LATEX documents.

CHECK works in the following manner:

1) **Document Recognition** – a LATEX recogniser parses the document and creates a document tree. This is a tree-like structure that resembles the structure of the document at various levels of abstraction such as section, subsection and paragraph. For each of the input words, a form of lemmatisation is applied that converts plurals to a singular form, verbs to their present infinitive form and all suffixes such as adjectives and adverbs to their fundamental stems.

2) **Keyword Extraction** – IR techniques are used to extract words that best describe the semantics of a document. These are categorised into **open-class** (nouns, verbs, adjectives and adverbs) and **closed-class** (prepositions, pronouns, conjunctions and interjections) words. Some heuristics such as formatting commands within the LATEX document are used to aid keyword extraction.

3) **Generate Structural Characteristics** – for each document, a Structural Characteristic (SC) is generated. This is the document structure merged with the extracted set of keywords. The keywords are assigned weights at the section, subsection and paragraph levels.

The result of this process is a tree for each document with weighted keywords assigned to each node of the tree. These resulting document representations are added to a registration server that can be used to compare with new documents presented to the system. Similarity between the new and existing documents is measured using the **dot product** between **normalised vectors** representing the keywords for a document. This comparison is repeated for each level of document abstraction, but *only* if the previous comparison results in a similarity value greater than some predefined threshold level (i.e. avoid unnecessary comparisons at lower abstraction levels of un-related documents).

Tests of the CHECK system were made on 15 technical documents and involved measuring the precision and recall for the following experiments:

1) Identification of identical documents (exact copy)
2) Behaviour of CHECK with documents describing similar topics (no copy),
3) Identification of actual copying activities,
4) Behaviour of CHECK with plagiarised documents describing unrelated subjects.

As far as I can make out, the copying activities that CHECK is subjected to are simply copying from a range of documents and the merging together of a number of paragraphs into one (i.e. verbatim cut-and-paste). There seems to be no mention of substitutions or synonymous phrases and no study appears to have been undertaken on actual plagiarised examples, they are all experimental. However, the precision and recall results show that the system was able to discriminate between the small set of plagiarised and non-plagiarised documents used for testing.

4   Software plagiarism

4.1   Introduction

More literature regarding plagiarism detection in software code was found over that for detection in written text. Part of the reason is because detecting plagiarism in software code is simpler than within natural language. The complete grammar for a programming language can be defined and specified, but natural language is much more complex and ambiguous making it much harder to build a successful plagiarism system. Parker and Hamblen (34) define plagiarism in software as: "a program which has been produced from another program with a small number of routine transformations". The **transformations** that can take

place can range from the very simple (such as changing comments in the code, or changing the names of variables) to the more complex (replacing control structures with equivalents e.g. replacing a "for" loop with a "while" statement). This can be pictorially viewed using the six levels of program modification in a plagiarism spectrum (see Figure 7) as given by Faidhi and Robinson (14).

A particularly useful source for software plagiairism is Whale (47) who lists potential methods or disguising programs including:

- Changing comments,
- Changing data types,
- Changing identifiers,
- Adding redundant statements or variables,
- Changing the structure of selection statements,
- Combining copied and original program statements.

Whale goes on to discuss evaluation of plagiarism detection systems and various similiarity detection techniques including his own tool called Plague (discussed later in this report).

An important property of any system is the correct detection of **suspicious code**. That is, a clear distinction between similarity that is due to chance and similarity due to plagiarism. This is similar to detecting "unusual" words or phrases in written text.

The first techniques used for plagiarism detection in software have been classified as **attribute-counting** techniques (48) or **ranking measures** (47). The first programs appeared in the 1970s and used either Halstead's software science metrics (19), McCabe's cyclomatic complexity (30) or scope number (21).
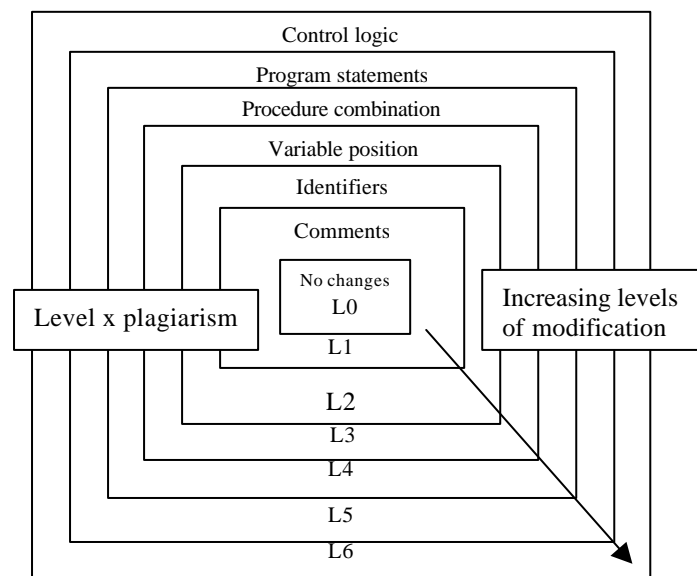

Figure 7 –Program plagiarism spectrum (Faidhi and Robinson)

Halstead's metric used the following quantities:

n_1 = number of unique or distinct operators.
n_2 = number of unique or distinct operands.
N_1 = total usage of all the operators.
N_2 = total usage of all the operands.

Halstead then described the *vocabulary* (n) as:

$$n = n_1 + n_2$$

And implementation *length* (N) as:

$$N = N_1 + N_2$$

One of the primary measures to come from the element counts was the size measure known as volume (V):

$$V = N \log_2 (n)$$

McCabe's cyclomatic complexity is used to measure control flow through a program by calculating the number of execution paths. The measure, V(G), is derived from a flow graph representation of the program – nodes are blocks of sequential statements and edges are processing paths:

$$V(G) = e - n + 2p$$
where:

e = the number of edges in the graph.
n = the number of nodes in the graph.
p = the number of components in the
    graph (p=1 for one module).

The nesting level metric measures the average nesting depth of a program or module by assigning each line of code a depth indicator. The sum of all the statements produces the total nesting depth and an average can be calculated by dividing the total measures by the number of statements in the program/module.

The first person to use Halstead's metrics for plagiarism detection was Ottenstein (33) from Purdue University who developed a program to detect plagiarism in Fortran programs. This technique was found to be ineffective for very short programs (could not cope with non-trivial program alterations), and thus began a transition between attribute-counting techniques and **control-flow** (or **structure metrics**) techniques. Donaldson et al. (11) proposed a system called ACCUSE that made use of both types of metric to detect plagiarism in Fortran programs (originally developed by Grier at the US Air Force Academy in Fortran programs). The most recent systems use structure metrics and compare string representations of the program structure. They assess the similarity of token strings and do not require exact matches. Examples of these include Plague, Sim and YAP.

An example of a very simple algorithm to detect plagiarism based upon string comparisons (34) is:

1. Remove all comments.
2. Ignore all blanks, extra lines, except when needed as delimiters,
3. Perform a character string compare between the two files using UNIX *diff*, *grep* and *wc*.
4. Maintain a count of the percentages of characters that are the same (called character correlation).
5. Run for all possible program pairs.
6. Create a summary of comparisons containing the character correlation in descending order and examine.

Other methods worth mentioning at this stage are those of: 1) Faidhi and Robinson (14) who use 24 metrics to assess similarity. The first 10 are most likely to be altered by a novice plagiarist and the rest by an experienced plagiarist. 2) Rees and his STYLE system that was also used to award points for the program based upon style, 3) Rees and Robinson's CHEAT program, 4) Jankowitz who analysed programs using static execution trees obtained by parsing the main program body and procedures and 5) the Plague tool developed by Whale.

Metric	Description	Reference
Volume	Halstead's n, N and V measures. The V measure is said to be a reflection of the number of mental comparisons required to generate a program.	Halstead, (19)
Control Flow	McCabe's cyclomatic complexity calculates the number of execution paths through a program and thus gives an indication of the programs'complexity.	McCabe, (30)
Structure	By considering indicators representing the degree of coupling between modules, the data and control transfer of a program can be represented.	Leach, (27)
Data Depend-ency	In a similar manner to measuring control flow, the dependency of data can be meaured. A suggested Generalised Program Graph (GPG) has been suggested in which nodes denote prodicate clauses and variable definititions.	Bieman and Debnath, (4)
Nesting Depth	The average nesting depth of a program or module can be calculated.	Dunsmore. (12)
Control Structures	These measures assign weights to various control structures of a program, e.g. an IF..THEN statement may be given a weighting of 3. The sum of the weighting is used to denote the complexity of a program. Two examples of this indicator are the MEBOW (Measure Based on Weights) and NPATH.	Jeyaprakash, (26) and Nejmeh, (32).

Table 1 – Characteristics that can be used alone or together to detect software plagiarism

Verco and Wise (50) analyse both the Accuse system and the Faidhi-Robinson approach to evaluate the performance of both techniques. Wise et al. have produced a series of tools called YAP (Yet another Plague) that can be used on a wide range of languages. A further related area of interest is that of software forensics. This area attempts to determine authorship of a code fragment with the main intention being to determine the author of programs in cases of computer fraud, security breach and plagiarism. A good introduction to software forensics is given by Sallis et al. (38) who also go on to summarise the most popular characteristics that can enable or improve plagiarism detection (see Table 1). Three of the characteristics we have already seen: Halstead's metrics, McCabe's cyclomatic complexity measures and the nesting level metric.

The rest of this section presents further examples of the most popular software plagiarism detection tools including some of those developed more recently. I do not present some of the older tools such as ACCUSE or Ottenstein's original program as the attribute-counting techniques on their own have been shown to be less successful than structure-based approaches or a combination of both.

4.1.1 Sim[16]

The SIM (Software Similarity Tester) - developed by Dick Grune (18) at Vrije Universiteit - tests program similarity by measuring the lexical similarity of texts written in C, Java, Pascal, Modula-2, Miranda and natural language.

Dick Grune, (18), states that SIM is able to:

- Detect plagiarism in software projects.
- Detect potentially duplicated code fragments in large software projects.

The output of SIM can be processed using shell scripts to produce histograms or a list of suspect submissions. The basic algorithm is:

1. **Read the program files (pass 1)** – each file is read using a lex-generated scanner appropriate for the input and the 1-byte tokens are stored in an array.
2. **Prepare a forward-reference table** – each text substring of a minimum size is compared with every other to the right of it. The result of this is a forward-reference table where the index of the next

---

[16] http://www.few.vu.nl/~dick/sim.html and source code (version 2.12) available from:
ftp://ftp.cs.vu.nl/pub/dick/similarity_tester/

substring that hash onto the same location are stored. If no similar next substring is found, the index value is 0.

3. **Determine the set of interesting runs** – the algorithm works through all the files to test and determines the best substring match (called a run).
4. **Determine the line numbers of the interesting runs (pass 2)** – this phase finds the start and end line number for each chunk known by token position only.
5. **Print the contents of the runs in order (pass 3)** – the stored runs are then retrieved in order of importance and using line numbers prints and formats the runs for easier analysis.

No test results are reported in the accompanying software documentation.

4.1.2 Siff[17]

The original application for this tool was to find similar files in a large file system (28) and called Sif. More recently, the technique has been used in conjunction with other methods to measure the similarity in Java bytecode files (29). The program has been renamed Siff.

Siff was originally designed to compare a large number of text files finding similarity amongst them. Diff is a UNIX tool that can compare two text files for similarity, but assuming 1 second per comparison, all pairwise comparisons between 5,000 files would require more than 12 million comparisons taking about 5 months of CPU time. This is impractical and hence the Siff algorithm was created to reduce this time. Siff creates a compact representation of a file called an **approximate fingerprint**. This fingerprint is consists of an **anchor** – a string of characters (e.g. *acte*) – and 50 characters from this point (may include words such as *character*). A **checksum** is computed of the 50 characters and this compared between files. So that the chosen anchors are reasonably representative of the text, a form **of random sampling** is used. This **fingerprint** can be compared quickly, but allow for differences in the files (as little as only 25% similarity). If two files have between 5-10 shared fingerprints, they are classed as similar (will depend on file size). The advantage of this method is that the fingerprint for two similar files will have a large intersection and two non-related files will have a very small intersection with high probability. Typically the probability of finding the same 50-byte string in two unrelated files is very low, but the method is susceptible to **"bad" fingerprints**. For example, formatting text in documents can account for many file similarities, especially if the actual document text is small. This could cause two unrelated files to be grouped as similar. This could be overcome by extracting just the text from the documents.

This method of comparing files is completely **syntactic** and no mention of file size limits are given where the method may breakdown. Uses for Siff that are mentioned in (28) include:

- Comparing different revisions of program code for plagiarism,
- Finding duplicate copies of files in a large data store (e.g. directory structure),
- Finding similar files on the Internet to improve searching and reduce looking through directories (i.e. ftp directories) where there could be similar files as those already viewed,
- For publishers to detect plagiarism,
- For academics wanting to detect plagiarism in assignments,
- For grouping together similar files together before they are compressed,
- Comparing revisions of files on home, portable and work machines.

Future improvements could involve selecting the anchors with more "intelligence" that are representative of the file contents (i.e. using "important" words of a document) and comparing only the text of a document not the formatting. Also more empirical studies to determine the size of checksum window and the amount of matches for large and small files to be classed as similar would be beneficial.

---

[17] More information can be obtained from: http://glimpse.arizona.edu/javadup.html

4.1.3   DUP

DUP is a technique developed by Brenda Baker (3) that finds long sections of software code that *almost match*. The technique uses the idea of a **parameterised match** (or p-match) that will match two segments of code if they are textually the same, but contain simply systematic changes to variable names. The user specifies a threshold length over which DUP reports all p-matches between files. The output is the percentage of duplication between two files and a profile showing the matches involved in each line in the input and a plot showing where the matches occur.

The p-matches are calculated using **parameterised suffix trees** (p-suffix tree) that represents offset encodings of suffixes of the token string in the form of a compacted **trie**. A trie is simply a tree structure where each node represents one character and the root represents the null string. The advantage of this representation is the time taken to find a string is **linear** and the process to build and add more elements to the the tree is very easy.

DUP is able to identify p-matches by replacing actual tokens such as identifiers by their offsets. This removes identify, but preserves the distinctness of different identifiers. The first occurrence of an identifier is replaced by 0, later ones by the number of tokens since the previous one. For example, given two code segments: *u(x,y)=(x>y)?x:y* and *u(w,z)=(w>z)?w:z*, both would be encoded as 0(0,0)=(6>6)?5:5 because u and v occur in the same positions, as do pairs x and w and y and z.

Further uses of the **suffix tree** includes:

- Finding all occurrences of q as a substring of S
- Finding the longest substring common to a set T of strings S_1 .. S_k
- Finding the longest palindrome in S
- Finding the longest repeated string in a text.

DUP has also been used for software maintenance in finding identical sections of code (except for systematic changes of parameters) and plagiarism detection (29).

4.1.4   Plague

The Plague (47) system (so called to act as a reminder of the practice of plagiarism) is an extension of the structure-based plagiarism detection approach and uses details of program structure for comparisons. In his paper (47), Whale describes the lack of common basis for comparing different techniques and often vague results supporting a method. Whale suggests measures that can be used for classifying matches and defines four terms that are used in his evaluation:

- **Recall**: the number of relevant documents retrieved as a proportion of all relevant documents.
- **Precision (p)**: the proportion of relevant documents in the retrieved group.
- **Sensitivity**: the threshold where a given submission pair is nominated as being similar (ability to detect similarities).
- **Selectivity**: the ability to maintain high precision with increasing sensitivity (ability to reject differences).

A detection system should allow the sensitivity to be adjusted (this is similar to a text retrieval system allowing a user to broaden or narrow a query). This allows a trade-off between the number of documents and similarity to be adjusted. There should be a threshold
where the sensitivity will correctly select all the plagiarised documents but not those dissimilar (limiting sensitivity).

The precision is defined as (p = m/n) where m is the number of positive detections (those which are plagiarised cases) from n documents. In the ideal case, the ratio would be 1 (i.e. of 8 documents selected, 8 are known plagiarism cases). A common value used for comparisons in text retrieval is (p = 0.6).

Using the above characteristics, it is possible to compare the different techniques. For example, the graph in
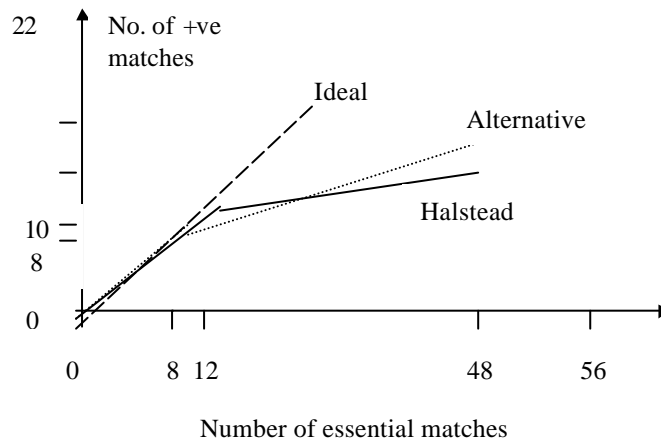

Figure 8 – Detection responses for Halstead and Alternative profiles

Figure 8 shows the number of essential matches (the matches between pairs of documents that form a minimum-spanning graph, as oppose to redundant matches) against the number of positive detections for the ideal plagiarism detector. Comparisons are made between Halstead's attribute counting method and another using an alternative method (structure-based). The graph is based on results obtained by Robinson and Soffa (35) using programs from 44 students over 3 problems.

Plague essentially works in 3 phases (50):

1. Create a sequence of tokens and a list of structure metrics to form a structure profile. The profile summarises the control structures used in the program and represent iteration/selection and statement blocks. The structure profile takes the form of a generalised regular expression.
2. An $O(n^2)$ phase compares the structure profiles and determines pairs of nearest neighbours. The majority of submissions are assumed unpaired and any that are paired move into the next stage.

3. The final stage is to compare the token sequences using a variant of the **Longest Common Subsequence** (8) for similarity.

Problems with Plague include:

1. Plague is hard to adapt to new languages; it is very time-consuming.

2. The results of Plague are two lists ordered by indices H and HT that need interpretation. The results are not immediately obvious.

3. Plague suffers from efficiency problems and relies upon a number of additional UNIX tools. This provides portability problems.

The results of Plague are good. From 44 documents, Plague managed to correctly identify 22 of 24 positive detections. Table 2 shows the results compared to different plagiarism detection tools and Plague establishes a clear margin between itself and the nearest tool.

First Author	Method	Detections @ p = 0.6	Perfor- mance index
various	Halstead	0	0.00
Berghel	Alternative	1	0.04
Donaldson	profiles	2	0.07
Robinson	basic blocks	6	0.23
Donaldson	strings	6	0.25
Whale	Plague	22	0.84

Table 2 – Performance indices for several Plagiarism detection systems

4.1.5   YAP1 and YAP2

The YAP (stands for Yet Another Plague) series of tools are based upon the Plague plagiarism detection tool. Michael Wise created the original version (YAP1) in 1992 (52) and followed this with a more optimised soon after (YAP2). In 1996 Wise produced the final version of YAP (YAP3) that could deal very effectively with **transposed sequences** (this version is discussed in the text plagiarism section – 3.5.4). The main goal of YAP was to build on the foundations of Plague (47), a tool that proved more successful than previous attribute-counting and structure-based techniques, but to overcome some of the problems experienced with this technique.

All YAP systems essentially work in the same manner and have two phases of operation:

1. The first phase generates a token file for each submission.
2. The second phase compares pairs of token files.

The tokens represent "significant" items in the chosen language that could be either language structures or built-in functions. All identifiers and constants are ignored. A small assignment may have typically 100-150 tokens, while a larger assignment may have 400-700. The tokenisation phase for each language is performed separately, but all versions have the same structure (52):

1. Preprocess the submitted reports:
    *   remove comments and print-strings,
    *   remove letters not found in legal identifiers,
    *   translate upper-case letters to lower-case,
    *   form a list of primitive tokens.

2. Change synonyms to a common form, e.g. in C_YAP strncmp is mapped to strcmp. This is similar to mapping words to their hypernyms.

3. Identify function/procedure blocks.

4. Print out function blocks in calling order. Repeated function blocks are only expanded once, subsequent calls are replaced by a numbered token representing that function. This prevents an explosion in tokens.

5. Recognise and print out tokens representing parts of the target language from a given vocabulary. Function calls are mapped to FUN and other identifiers ignored.

In the comparison phase, the UNIX utility *find* is used to gather a list of previously prepared token files and *sdiff* used to compare each token file with every other. The output analysed using a simple *awk* script. This

phase is slow being $O(n^2)$, but was speeded up in YAP2 using a custom-written C program implementing Heckel's algorithm (22).

The result of YAP1 is a value from 0 to 100 for each comparison, 0 representing no match and 100 an exact copy. A **mean** and **standard deviation** are then computed and any results appearing 3 or more standard deviations from the mean are worth exploring further.

YAP1 was tested against Plague on 167 student Pascal assignments. Plague produce 640 pairs above a threshold of 10 and YAP1 654 pairs with 1.53 standard deviations from the mean. A surprising result was that 391 matches found by YAP1 did not appear in Plague and 377 of the matches found by Plague did not appear in YAP1. As many as 110 matches given by YAP1 and not supported by Plague were confirmed as matches where students had split procedures. By expanding procedures in calling order, the similarity becomes more obvious.

Wise, (52), claims YAP is able to cope with:

- **Changing comments or formatting** – YAP is immune to this.
- **Changing identifiers** – YAP is immune to this.
- **Changing the order of operands** – YAP is immune to this.
- **Changing data types** – YAP is immune to this.
- **Replacing expressions with equivalents** – YAP is immune to this or records only a small difference (1 token).
- **Adding redundant statements or variables** – YAP is immune to additional variables. Additional statements result in only small changes (1 token per statement).
- **Changing the order of independent statements** – e.g. rearranging Prolog clauses. YAP has problems with this and can cause significant differences. The assumption that YAP makes, however, is that to be able to do this effectively the student must understand the program and how independent parts interact. If procedures or function definitions are re-ordered YAP is immune to this.
- **Changing the structure of iteration statements** – YAP is largely immune to this.
- **Changing the structure of selection statements** – YAP is largely immune to this due to using the synonym replacements e.g. case maps to elseifTEST and so does if … then ... else.
- **Replacing procedure calls by the procedure body** – YAP is largely immune to this.
- **Introduction of non-structured statements** – these create only 1 token differences.
- **Combining original and copied program segments** – these generally show up as a group of assignments with matches well above the mean.

The results proved YAP was able to counter most of the ploys used by students in software plagiarism (as given by Whale) and YAP is at least as accurate as Plague in finding high similarities and as good in avoiding finding matches where none exist.

4.1.6 MOSS[18]

MOSS (Measure of Software Similarity) was developed in 1994 by Alex Aiken at Berkeley as a system for measuring the similarity of source code written in C, C++, Java, Pascal, Ada, ML, Lisp or Scheme. The main aim being for plagiarism detection in programming classes. The tool is supplied as an Internet service and given a list of source files, returns an HTML page listing of pairs of programs with similar source code. Supplied code and libraries can be eliminated from the similarity tests which removes false positives – that is lines of code that the program thinks are plagiarised when in fact they are not.

No information is given about the algorithm except it is "more sophisticated than systems based on counting the frequency of certain words in the program text" - MOSS Web page [16]; the tool probably uses a structure technique.

---

[18] More information available from: http://ftp.cs.berkeley.edu/~aiken/moss.html

No test results or success results are given.

4.1.7 JPlag[19]

The amount of information given about JPlag is very sparse, but the tool is mentioned because members of The Department of Informatics at the University of Karlsruhe where Professor W. Tichy is head of the research group have developed it. The tool finds similarities between source code written in Java, C, C++ and Scheme. The tool compares bytes of text and syntax/program structure making it more robust against plagiarism disguise. JPlag can be accessed from a Web page and once an account is obtained, the tool can be used for free.

There are no test results on the success of JPlag.

4.1.8 Bandit[20]

This tool has been developed by Malcolm Shute, Koshor Mistry, Haydn Robinson and Adrian West as part of the computer science programming laboratory called ARCADE at the Victoria University of Manchester (46).

The tool collects work from students and checks for copied code between current and previous student work. The code, of at least a few tens or hundreds of lines, is broken down into a stream of lexical tokens, independent of program layout, comments and variable names. The similarity between program files is then found by comparing the token streams and looking for sequences of similar tokens. The search for similar sequences is performed using a Dotplot (23) approach where the token stream of one of the program pair is drawn against the X-axis and the token stream for the other against the Y-axis. A cross is placed where two tokens match. The detector seeks unbroken diagonal lines that indicate sequences of tokens in common. The method is resistant to re-arrangement of tokens as broken or displaced diagonals correspond to insertion, deletion or re-arrangement of tokens and can be easily seen. The tool is only language specific with respect to the input tokeniser, the rest of the analysis is relatively generic.

Bandit returns back a list of matching sequences described by length and position within the source programs. The longer the match, the more significant this is viewed. Generally the most suspicious cases are drawn to the top, falling quickly to a noise level moving further down the list.

Bandit produces a graph of programs versus similarity score (see Figure 9) that can be used to quickly identify potential suspect programs (46).
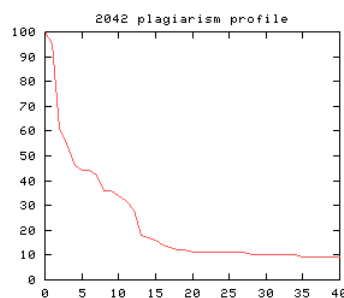

Figure 9 – an example similarity score for 40 programs (programs with a score greater than about 20% require further investigation)

---

[19] More information available from: http://wwwipd.ira.uka.de:2222/

[20] More information available from: http://socrates.cs.man.ac.uk/~ajw/pd.html

Bandit also provides a graphical interface where the programs can be easily inspected for plagiarism (see Figure 10).

The authors claim the only way to fool Bandit is to re-arrange all of the tokens in the program so no suspicious structural correspondences remain, an action that would require a great understanding of the program code (what is wanted from a programming assignment anyway). In the four years of running at Manchester, the detector has caught a number of students who have performed plagiarism.

Figure 10 – the Bandit user-interface

A further outcome of the tool has been to highlight people who are struggling with the course but do not ask for help as these people plagiarise in a more generally amateur manner and are therefore more easily caught.

4.1.9   Cogger

Cogger (an Anglo-Irish slang word copying homework or other exercises) is a novel approach to plagiarism detection in computing assignments using a Case Based Reasoning (CBR) approach (9). The problem of finding similarity in programs is made analogous to the problem of case retrieval in CBR.

In CBR, the objective is to look for similarity between cases to allow the reuse of old solutions in new situations. The similarity between cases may be based upon surface features, or features more abstract and structural. These features are used to create a profile that can be used for comparison.

A method called Base Filtering is used to select a small set of candidates from the base case that are similar to the target case. This results in a classification problem and techniques such as nearest neighbour or decision trees can be used. The main difference between techniques is whether all case-bases need to be searched. Cogger uses a form of decision tree based on an information theoretic approach.

In the case of Cogger identifying plagiarism, the programs were converted to a string representation and an attempt made to find common substrings. Shallow features used were simply ordered frequency counts of identifiers in the programs. This was shown ineffective thus counts of reserved words only were used. Although an improvement, still not effective for many plagiarism cases. The final representation for the profile was to use less predictive keywords.

The test results are very vague and no success rates given.

5   Conclusions

I have given a number of examples in this report of tools used to detect plagiarism in natural and programming languages.   Examples of techniques used for plagiarism have been included for both plagiarism areas and issues regarding why plagiarism occurs and what can be done to detect it.

A number of basic techniques in plagiarism of written text have been discussed with an emphasis made on paraphrasing, one of the hardest techniques to detect.

Recent publicity in newspapers has focused on plagiarism in written text from the Internet including from term-paper mills and Net essay banks. With the increasing volume of information available on-line to students, manually detecting plagiarism is becoming harder. This emphasises the need for an automatic technique to detect similarities. A number of systems have been built to deal with Internet-based plagiarism, but seem naïve in the methods used to detect similarity (for example, do not take into account paraphrasing). Currently, there seems little evaluation by a third party to compare these tools and identify how accurate and successful each tool really is.

Much more detailed information about plagiarism detection came from research from the software side. There are obvious similarities between methods used for both software and text plagiarism detection. These include the replacement of synonyms, re-ordering of sentences, insertion and deletion of text, change of author style etc. Therefore, methods used for software plagiarism detection may well work for text also. The analysis of software plagiarism and evaluation has taken a formal approach and I would like to see a similar approach taken to textual plagiarism.

Software plagiarism detection tools use either an attribute-counting method or structure-based approach. Careful comparison has shown that the structure-based methods are much more effective than those counting attributes. It would be interesting to try a similar approach to plagiarism of written text where methods based on structure of text may prove to be more successful than those based simply on attribute-counting (such as words). For example, it would be interesting to compare an Information Retrieval approach (such as vector representation and cosine similarity) that would compare the text as a "bag-of-words" and a method that aligned the texts and compared similarity (i.e. considering the structure of the text). Currently, comparing software plagiarism detection techniques with written text does not appear to have been undertaken. There are similarities between detection of plagiarism in each area and these could be exploited further. However, there are also obvious differences between plagiarism in written text and programming languages. Not more so than simply the ambiguity and complexity of natural language compared to programming languages that would make applying the same technique to both areas potentially unsuccessful.

A further improvement would be experiments based upon a common evaluation technique. Whale provided this basis for software that allows the success of techniques to be easily illustrated and ranked. This same evaluation strategy could be made with plagiarism in written text.

I am also interested in using the style of an author to detect unusual patterns in both software code and written text. This may provide useful research into how the style of an author changes over time and it would be interesting to determine what features could be used to successfully determine authorship.

I think teaching students about avoiding plagiarism is also very important as perhaps many do not know how to cite properly. This would help reduce cases of plagiarism where students claim ignorance or express a lack of training in citing. Providing examples and software to aid developing skills in citing may help to reduce unintentional plagiarism.

Speaking to lecturers at Sheffield University, most have come across cases of cut-and-pasting from the Internet. I feel it would be beneficial to introduce a tool that could help detect these cases automatically. The lecturers helped confirm that is was a change in the writing style of the student or use of "advanced" vocabulary that caused suspicion.

Finally, although attempts have been made at automatically detecting plagiarism, it will still require the intervention of a human to prove the plagiarism and provide the final verdict.

6   References

(1)   Karl F. Aiken, *The USCA Writing Room: paraphrasing*, The University of South Carolina, 1999,
(http://www.usca.sc.edu/scaonlinewr/hos/ para.htm).

(2)   University of Arizona , *Paraphrasing textual material*,
1999, (http://www.gened.arizona.edu/eslweb/paraphra.
htm).

(3)   Brenda S. Baker, *Parameterized Pattern Matching: Algorithms and Applications*, Journal of Computing System
Science, 52, pp(28-42), February 1996.

(4)   J. M. Bieman and N. C. Debnath, *An analysis of software structure using a generalized program graph*, In
Proceedings of COMPSAC'85, pp(254-259), 1985.

(5)   David A. Black, *Tracing Web Plagiarism – A guide for teachers*, Internal Document, Department of
Communication, Seton Hall University, Version 0.3, Fall 1999, (http://icarus.shu.edu/dblack/webcheat/one
page.html).

(6)   S. Brin, J. Davis and H. Garcia-Molina,*Copy detection mechanisms for digital documents*, Proceedings of the
ACM SIGMOD Annual Conference, San Francisco, CA, May 1995.

(7)   Paul Clough, *Identifying Re-use between the Press Association and newspapers of the British Press*, Department
of Computer Science, University of Sheffield, Internal Report, June 2000.

(8)   Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest, *Introduction to Algorithms*, MIT Press, 1990.

(9)   Padraig Cunningham and Alexander Mikoyan, *Using CBR techniques to detect plagiarism in computing
assignments*, Department of Computer Science, Trinity College, Dublin (Internal Report), September 1993.

(10)   P. J. Denning (chair of the ACM publications board), *Editorial: Plagiarism in the Web*, Communications of the
ACM, Vol. 38, Number 12, December 1995.

(11)   J. L. Donaldson, A. Lancaster and P. H. Sposato, *A plagiarism detection system*, ACM SIGSCI Bulletin 13(1),
pp(21-25), February 1981.

(12)   Dunsmore, *Software metrics: an overview of an evolving methodology*, Information Processing and
Management 20, pp(183-192), 1984.

(13)   John R. Edlund, *What is "Plagiarism" and why do people do it?*, University Writing Centre Director, California
State University, LA, 1998, (http://web. calstatela.edu/centers/write_cn/plagiarism.htm).

(14)   J. A. Faidhi and S. K. Robinson, *An empirical approach for detecting program similarity and plagiarism within
a university programming environment*, Computing in Education, Vol. 11, pp(11-19), 1987.

(15)   Agnès J. Fauverge, *A critical assessment of Mel'cuk's Meaning Û Text Model and its Contribution to the Study
of the Paraphrase*, Mémoire de Maîtrise L.V.V Anglais à l'Universite de Montpellier III, September 1993.

(16)   Susan Finlay, *CopyCatch*, Masters Dissertation, University of Birmingham, 1999.

(17)   G. H. Gonnet and R. Baeza-Yates, *An Analysis of Karp-Rabin String Matching Algorithm*, Information
Processing Letters 34, pp(271-274), 1990.

(18)   Dick Grune and Matty Huntjens, *Het detecteren van kopieën bij informatica-practica (in Dutch)*, Informatie 13,
pp(864-867), 11 November 1989.

(19)   M. H. Halstead, *Elements of software science*, North Holland, New York, 1977.

(20) Bruce Handy, *Steven Stealberg? The director's new film is hit with a $10 million plagiarism suit, but isn't history free to all*, Time magazine, Vol. 150, No. 22, November 24th 1997 (http://www.time.com/time/ magazine/1997/ dom/ 971124/the_arts_show.steven_ stealb.html).

(21) W. A. Harrison and K. L. Magel, *A complexity measure based upon nesting level*, ACM SIGPLAN Notices 16(3), pp(63-74), March 1981.

(22) Paul Heckel, *A Technique for Isolating Differences between Files*, Communications of the ACM 21(4), pp(264-268), April 1978.

(23) Jonathan Helfman, *Dotplot: A program for exploring self-similarity in millions of lines of text and code*, Journal of Computational and Graphical Statistics, 2(2), pp(153-174, June 1993.

(24) John Hickman, *Cybercheats*, The New Republic, March 23rd 1998.

(25) Hongyan Jing and Kathleen R. McKeown, *The decomposition of human-written summary sentences*, In Proceedings of SIGIR'99, pp(129;136), 1999.

(26) Jeyaprakash, *MEBOW: A comprehensive measure of control flow complexity*, In Proceedings of COMPSAC'87, pp(238-244), 1987.

(27) R. J. Leach, *Using metrics to evaluate student* programs, ACM SIGSCI Bulletin 27, pp(41-43), 1995.

(28) Udi Manber, *Finding similar files in a large file system*, In USENIX, pp(1-10), San Francisco, CA, January 1994.

(29) Udi Manber and Brenda S. Baker, *Deducing similarities in Java sources from bytecode*, 1998 USENIX Technical Conference, New Orleans, June 1998.

(30) T. J. McCabe, *A complexity measure*, IEEE Transactions on Software Engineering, SE-2 (4), pp(308-320), December 1976.

(31) Tony McEnery and Mike Oakes, *Authorship identification and computational stylometry*, Internal report, Department of Linguistics, Lancaster University, 1998.

(32) Nejmeh, *NPATH: A measure of execution path complexity and its applications*, Communications of the ACM 31, pp(188-200), 1988.

(33) L. M. Ottenstein, *Quantitative estimates of debugging requirements*, IEEE Transactions of Software Engineering, Vol. SE-5, pp(504-514), 1979.

(34) Alan Parker and James Hamblen, *Computer algorithms for Plagiarism Detection*, IEEE Transactions on Education, Vol. 32, Number 2, May 1989.

(35) Philip Roberts, *Ibcos Computers Ltd. and another v. Barclays Mercantile Highland Finance Ltd. and others*, Fleet Street Reports, In The High Court of Justice – Chancery Division, 1994.

(36) S. S. Robinson and M. L. Soffa, *An instructional aid for student programs*, ACM SIGCSE Bulletin 12 (1), pp(118-129), February 1980.

(37) John Rogers, *Plagiary and the Art of skillful citation*, Department of Immunology, Baylor College of Medicine, 1996, (http://condor.bcm.tmc.edu/Micro-Immuno/courses/igr/homeric.html).

(38) Philip Sallis, Asbjorn Aakjaer and Stephen MacDonell, *Software Forensics: old methods for a new science*, In Proceedings of Software Engineering: Education and Practice (SE:E&P'96), Dunedin, New Zealand, IEEE Computer Society Press, pp(481-485), 1996.

(39) Pamela Samuelson, *Legally Speaking: Self plagiarism or Fair Use?*, Communications of the ACM, 21, August 1994, (http://www.ilt.columbia.edu/text_ version/projects/copyright/papers/samuelson.html).

(40)   N. Shivakumar and H. Garcia-Molina, *SCAM: a copy detection mechanism for digital documents*, Proceedings of 2nd International Conference in Theory and Practice of Digital Libraries (DL'95), Austin, Texas, June 1995.

(41)   N. Shivakumar and H. Garcia-Molina, *Building a Scaleable and Accurate Copy Detection Mechanism*, Proceedings of 1st ACM Conference on Digital Libraries (DL'96) Bethesada, Maryland, March 1996.

(42)   Antonio Si, Hong Va Leong, Rynson W. H. Lau, *CHECK: A document plagiarism detection system*, In Proceedings of ACM Symposium for Applied Computing, pp(70-77), February 1997.

(43)   Wilson Taylor, *Cloze Procedure: A news tool for measuring readibility*, Journalism Quarterly, Vol 30, pp(415-433), 1953.

(44)   Walter F. Tichy, *The String-to-String Correction Problem with Block Moves*, ACM Transactions on Computer Systems 2(4), pp(309-321), November 1984.

(45)   The university of Kentucky, *Plagiarism: definitions*, examples and penalties, Department of Chemistry, 1998, (http://www.chem.uky.edu/Courses/common/ plagiarism.html).

(46)   Adrian West, *Coping with plagiarism in Computer Science teaching laboratories*, Computers in Teaching Conference, Dublin, July 1995.

(47)   G. Whale, *Identification of Program Similarity in Large Populations*, The Computer Journal, Vol. 33, Number 2, 1990.

(48)   The University of Wisconsin Writing Centre, Writer's handbook: Quoting and Paraphrasing, 1997, (http:// www.wisc.edu/writing/Handbook/QuoSamplePara phrases.html).

(49)   Michael J. Wise, *YAP3: improved detection of similarities in computer programs and other texts*, presented at SIGCSE'96, Philadelphia, USA, February 15-17 1996, pp(130-134), (ftp://ftp.cssr.cam.ac.uk/pub /michaelw/doc/yap3.ps).

(50)   Kristina L. Verco and Michael J. Wise, *Software for Detecting Suspected Plagiarism: Comparing Structure and Attribute-Counting Systems*, 1st Australian Conference on Computer Science Education, Sydney, Australia, July 3-5, 1996.

(51)   Michael J. Wise, *String similarity via Greedy String Tiling and Running Karp-Rabin Matching*, an unpublished paper, December 1993, (ftp://ftp.cssr. cam.ac.uk/pub/michaelw/doc/RKR_GST.ps).

(52)   Michael J. Wise, *Detection of similarities in student programs: YAP'ing may be preferable to Plague'ing*, SIGSCI Technical Symposium, Kansas City, USA, pp(268-271), March 5-6, 1992.

(53)   David Woolls, *PAPERS: authors, affiliations and abstracts*, IAFL Conference '99, University of Birmingham, 1999, (http://www-clg.bham.ac.uk/ forensic/IAFL99/ conf_abst.html).