Problem of plagiarism and its detection

Branko Kaučič, Dejan Sraka
Faculty of Education
University of Ljubljana
Kardeljeva ploščad 16, 1000 Ljubljana, Slovenia
{branko.kaucic,dejan.sraka}@pef.uni-lj.si

Abstract. *Rapid development of internet technologies simplified sharing any kinds of data. Extremely notable is also sharing the source codes. Consequently, today's "copy-paste" generation is a subject of a notable problem of plagiarism. It is present in many areas, from educational and research areas to software development. We started a project of studying known plagiarism detection systems and developing a framework that would detect plagiarism between different types of files. At first stage we are focused on source codes of applications. In this contribution, most known systems are revised and compared among themselves. Our framework and its usage are also presented.*

Keywords. Plagiarism, cheating, trust, cut and paste culture, electronic plagiarism detection service, programming

1 Introduction

Rapid growth of technology, internet and powerful search engines yielded massive amount of accessible information. In a world that is populated by the information technology, traditional searching for information from the books could be seen as a thing of the past. Namely, most needed information is already available in electronic format on the internet. The internet is the largest public repository of information, that was ever created, and much information is available on several web pages. Because there is no way to forbid users to use it, using the internet and online resources pose some serious problems. Although the search and the research is easier, the internet also provides a more conducive means to plagiarism – there is only a limited control over copied sources and it is difficult to verify the reliability of information resources. Plagiarism continues to plague all disciplines in secondary and higher education because of easy online access to source documents.

Many researchers report that plagiarism is increasing and present a serious problem in education. At present, in schools there is a battle for a higher number of (good) students, number of staff is not increasing, while the ratio student:staff is. By the incoming Bologna process, also the contact hours are reducing. The opportunities for a 'tutor' to identify the students that need additional help will consequently decrease, and the only indicator if someone needs help will be the assessment results. It is obvious that by increasing the number of students, the energy put to a single assessment is also reducing. Students know that very well and some cheat because of that. More and more students use the internet to obtain analysis, interpretation or even complete assignments and then submit these as their own work. Consequently, if assessment is copied from someone else and the 'tutor' doesn't found that out, student leaves that course with not enough knowledge and can later in life have problems because of that. More worrisome, studies show that students not only plagiarize regularly but also believe that it is okay to do so. Although, the temptation to plagiarize in the academic setting is not limited to students, in this paper we will restrict ourselves to the students only.

Today's "copy-paste" generation is therefore a subject of more and more notable problem of plagiarism. It is present in many areas, from educational and research areas to software development. As in all similar faculties, because of the popularity of the ICT we are also facing this problem. Consequently we started a project of studying known plagiarism detection systems and developing a framework for professors

and assistants that would detect plagiarism between different types of files. Electronic plagiarism detection services can be a useful help for 'tutors', but are not self-sufficient. Trust and student honesty must remain to obtain a successful academic system.

At first stage we are focused on source code of applications. Systems that detect source code plagiarism have a rich future. First systems were developed in 70s. Mostly used systems today detect similarity in programs by comparing their structure. These systems were developed as an answer to criticisms to the older systems which work by counting and comparing program's features. In this contribution, most known systems are revised and compared among themselves. Our framework and its possible usage are also presented.

The organization of the paper is the following. Section 2 presents the problem of plagiarism, why some people use it, and how the plagiarism is noticeable in the programming courses. In Section 3 and 4 we present plagiarism detection systems, based on the attribute counting methods and by mutual comparing of structures. In Section 5 our framework is briefly described and the last section concludes the paper with some ideas for future.

2 Plagiarism

Encyclopedia Britannica [6] defines plagiarism as "*the act of the writings of another person and passing them off as one's own. The fraudulence is closely related to forgery and piracy - practices generally in violation of copyright laws.*" Similarly, Webster's dictionary [20] defines it as "*a piece of writing that has been copied from someone else and is presented as being your own work.*" However, in the context of university education, plagiarism does not have a single meaning and can range from the citation of a few sentences without attribution to the copying of an entire work. There are a variety of ways students can use ICT inappropriately while completing their assignments. Very common approach is using of research papers purchased or downloaded from web sites. As stated in the introduction, students can easily gain access to what they need just by typing keywords in any internet search engine.

Schiller [17] reports about three different forms of plagiarism. The first form consists of simply copying word-for-word from a book, journal article, or page from the internet without placing the copied material in quotation marks and acknowledging the source. Submitting a term paper written by someone else, including a paper purchased from a commercial company, would be categorized as this form of plagiarism. The second form of plagiarism is to paraphrase another's organization and language without acknowledging the source. Extensive use of a source, even when a writer changes a few words, omits a few sentences, or reorders ideas, requires citing the source. The third form of plagiarism is basing material solely on the ideas of another. In this case the work is considered plagiarized because the writer has contributed no original thought. Writing an article without acknowledging that it follows the content, outline, and ideas from others' written or orally presented work is also considered as the plagiarism. However, there are several additional creative approaches to academic dishonesty that are available to students through technology [17]. Students can cut and paste sections from the internet articles into their assignments without attributing the work. Information from CDs such as encyclopedias, databases, and study guides can be inserted into assignments. Students may ask for assistance from others through electronic discussion groups and then cut and paste answers from other people into their work without acknowledging that assistance was received.

The ease with which text, numbers and computer codes can be moved between students and institutions unfortunately has the potential to undermine traditional forms of learning and assessment. Although course syllabi warn students that plagiarism is punishable, the effort the instructor must invest to pursue a case of plagiarism effectively guarantees that almost no prosecution will occur. It is important that educators address the issue of plagiarism, despite the additional time and emotional burden required to confront offenders.

2.1 Why students plagiarize

There are many reasons why students copy from each other, or collude when performing a specific piece of work. These include the following [17].

- A weak student produces work in close collaboration with a colleague, in the belief that it is acceptable.
- A weak student copies, and then edits, a colleague's program, with or without the colleague's permission, hoping that this will go unnoticed.
- A poorly motivated (but not necessarily weak) student copies, and then edits, a colleague's program, with the intention of minimizing the work needed.

At the level of the individual student, three categories are often explaining why certain individuals commit non-trivial plagiarism [1]. The three categories concern students' personal circumstances, personal traits, and whether the means and opportunity to plagiarize are readily to hand:

Means and opportunity: the widespread of internet, and online academic journals have contributed much to the rising incidence of plagiarism, as they have

made it possible for students to find and download materials from diverse sources with little reading, effort or originality. In addition, the web services with such materials are customized and thus difficult to detect using anti-plagiarism web crawling software. However, while the internet undoubtedly facilitates plagiarism, it does not possess the moral power to incite otherwise honest students to cheat. Lack of rules and prosecution for cases of plagiarism could encourage students to indulge in the practice.

Personal traits: Internal beliefs that academic cheating is immoral and dishonest are known to discourage plagiarism. However, the strongest motive for student cheating (according to Bjorklund and Wenestam's in [1]) is the desire to obtain high grades, which itself may depend on other considerations. For example, due to a person's innate need to prove his or her worth to him or herself and/or to the world, or to a pathological fear of failure.

Individual circumstances: For example, students who need to take paid employment to help finance their time at university have less time for study, and high academic workloads may need to be compressed into their available study periods. The time pressures are likely to cause growing numbers of students to resort to plagiarism. It is also interesting, that males are more ready to admit plagiarism than females [1].

2.2 Plagiarism in programming courses

Plagiarism is a common problem also in computer science courses. In most these courses, the completion of programming assignments is part of the course requirements. In many cases, these assignments contribute significantly to the student's grade; thus, each student is usually expected to work independently although sometimes team work is demanded. However, students are mainly collaborating with one another and programmed assignments can be copied and transformed with relative little effort. Consequently, defining and prosecuting plagiarism is difficult because of the fuzzy boundary between allowable collaboration and plagiarism. At some universities, committees concerning this have been formed, e.g. Carnegie-Mellon University.

Probably every instructor of a programming course has been concerned about possible plagiarism in the program solutions turned in by students. Instances of cheating are found, but traditionally only on ad-hoc basis. For example, the instructor may notice that two programs have the same idiosyncrasy in their user interface, or the same pattern of failures with certain test cases. With suspicions raised, the programs may be examined further and the plagiarism discovered. Unfortunately, this leaves much to chance. The larger is the class, more different people are involved in the grading, less is the chance that a given instance of plagiarism will be detected. Parker and Hamblen [14] define software plagiarism as: *a program which has been produced from another program with a small number of routine transformations*. Modifications to hide plagiarism in source code are very diverse, from changing comments, identifiers or formatting to changes made in decision logic of a program. Plagiarism detection is a pattern analysis problem. A plagiarized program is either an exact copy of the original, or a variant obtained by applying various textual
transformations such as those shown below. Faidhi and Robinson listed six levels of program modifications [7]:

- level 1: changes in comments and indentation,
- level 2: changes in level 1 and changes in identifiers,
- level 3: changes of level 2 and changes in declarations
- level 4: changes of level 3 and changes in program modules,
- level 5: changes of level 4 and changes in the program statements
- level 6: changes of level 5 and changes in the decision logic

It would be worth to mention an example, when plagiarist does not make completely no changes in program code. However, such plagiarism would hardly be unnoticed. It is also interesting to debate about a program in which only changes in program statements have been made – is the program still classified as level 5? Such classification on levels could be useful, but can be subjective. In response to that, numerous other classifications appeared in literature [11],[12].

2.3 Plagiarism detection systems

For students who know about various instances of cheating, which instances are detected and which are not, the plagiarism is very tempting. The standard "dumb" attempt at cheating on a program assignment is to obtain a copy of a working program and then change statement spacing, variable names, prompts and comments. By comparing all pairs of solutions against each other for evidence of plagiarism seems like the approach that will detect fraud. However, even the above mentioned case is mostly enough to require a careful manual comparison, which simply becomes infeasible for large classes. Since there is usually more than just a few assignments, programming classes are in desperate need for an automated tools which perform reliable and objective detection of plagiarism.

In programming courses there are two sources of solved assignments: the internet and other students. For the internet, the products like Turnitin and PlagiServe are harnessing the internet for detecting plagiarism, in a similar fashion to what search engines

such as Google has long been doing. However, the second source, other students, is more frequent and different plagiarism detection systems are needed.

Attempts to assess plagiarism, by technical means, run into the difficulty of distinguishing the differences between texts of different kinds. Much work in the past has been devoted to discovering concordances between texts. A plagiarism detection method must produce a measure that quantifies how close two source codes of programs are. Obviously, except for the case of a verbatim copy, detection approaches that use direct comparison of text files are weak, since there is no obvious closeness measure. Also, a simple file "diff" would of course detect only the most obvious attempts of fraud. There are various electronic systems to detect plagiarism in programming courses. From the middle of 70s to the end of 80s of 20th century were prevailing the systems that were finding resemblances based on counting and comparing program attributes. The technique was called attribute counting. Later plagiarism detection systems that were examining and comparing program structures were introduced. Standard software metrics and examination of redundant code were used, too. There are even servers on the web which detect plagiarism. For example, JPlag [15] at Karlsruhe University tries to find pairs of similar programs, and the MOSS server at Berkeley [3] looks for similar code sequences in a set of programs; each system creates a web page where the instructor can see which ones are suspiciously similar. All of these techniques operate by running an analysis program on groups of submissions to detect similarities and to calculate the likelihood of plagiarism. Many approaches take a lexical approach, where the program tokens are classified as language keywords and user symbols. The simple plagiarism detection systems convert the source programs into token strings, and then compare the strings using dynamic programming. Although they are reasonably successful in pointing to pairs or groups who submit similar work, they are limited in identifying the original author of the work.

The following sections present some of most known detection systems of this kind.

3 Detection of plagiarism by attribute counting

Attribute counting detection systems try to detect a plagiarism by scanning whole program line by line while counting system defined attributes. Similarities between programs are detected with mutual comparison of counted attributes. For each attribute counter a range is defined. If deviations of observed pair of attributes are inside of a range, programs are marked as a potential plagiarism.

The first known system is from 1976 by Ottenstein [13]. Based on technical report by Bulut and Halstead [4] Ottenstein defined the following parameters for counting:

n1 – the number of unique operators,
n2 – the number of unique operands,
N1 – the total number of occurrences of operators,
N2 – the total number of occurrences of operands.

Those parameters were later described in details in Elements of Software Science by Halstead [9]. Interestingly, later works of different authors ([2], [7], [19],...) even use the term "Halstead metrics" or "basic software science parameters". Ottenstein's system detects program pairs as possible plagiarism when four-tuples (n1, n2, N1, N2) of two programs match.

Next known system, Accuse, was developed by Grier in 1981 [8]. In order to [8]get more accurate plagiarism detection system, twenty parameters were introduced. After some tests have been made, Grier selected seven parameters that were used in calculation of correlation between program source codes:

1) number of unique operators,
2) number of unique operands,
3) total number of occurrences of operators,
4) total number of occurrences of operands,
5) number of code lines,
6) number of variables declared and used, and
7) total number of control statements.

Accuse compares seven-tuples with correlation function and two constants: "window size" and "importance value" for all seven parameters. Importance value and window size were defined experimentally by writing the source code for three programs (partially written collaboratively, partially individually). Constants for "importance value" were adjusted until those three programs were suspected of plagiarism. For every pair (A,B) of compared programs the correlation factor is calculated as a sum by all the parameters:

$$\sum_{i=1}^{7} Importance\ Value_i - |pcountA_i - pcountB_i|$$

where $|pcountA_i\text{-}pcountB_i|$ is less or equal to window size defined for parameter i. Pair of programs are suspected of plagiarism if correlation factor is greater than 28 (highest possible value for default constant values is 32).

In 1981 Donaldson et al. introduced also a system that in addition to counting attributes, also records the structure of a program [5][5]. System is observing next parameters:

1) total number of used variables,

2) total number of subprograms,
3) total number of input statements,
4) total number of conditional statements,
5) total number of loop statements,
6) total number of assignment statements,
7) total number of calls to subprograms, and
8) total number of statements of type 2-7.

Analysis of compared pair of programs is done in two phases. In first phase system checks programs, line by line and characterizes every program in two ways. First it counts attributes and saves values in a two-dimensional array. Those values are later used in second phase in which a degree of correlation is calculated. To determine similarity or difference factor, three different methods were implemented: sum of the differences, count of similarity and weighted count of similarity.
This system could be even called as an origin of all newer systems since it was the first system that used structure recording. All newer systems base on detection by comparing structure and have their idea in one or another way upgraded and perfected.

In 1982 Rees introduced system Style, automatic assessment program for programs written in Pascal [16][16]. His system is measuring ten parameters; five of them are intended for grading layout and other five for grading identifiers:

1) average line length,
2) use of comments,
3) use of indentation,
4) use of blank lines as separators,
5) degree of imbedded spaces within lines,
6) procedure and function units,
7) variety of reserved words,
8) length of identifiers,
9) variety of identifier names, and
10) use of labels and gotos.

Robson used measured values by Style for post-processor program called Cheat [16]to detect similar programs submitted by students [16]. Program was observing the following features:

1) total number of non-comment characters,
2) percent of embedded spaces,
3) number of reserved words,
4) number of identifiers,
5) total number of lines, and
6) number of procedures/functions.

In 1984 Berghel and Salach presented interesting results of empirical study [2]. Over several semesters of observation they made a list of fifteen key features that are probably most useful in identifying similarities between program pairs. With further factor analysis they excluded eight parameters from the list and formed two metrics: Halstead's and the alternative. They formed a tuple $C=(c_1,c_2,c_3,c_4)$ and considered program pair (P,Q), where $P=(p_1,p_2,p_3,p_4)$ and $Q=(q_1,q_2,q_3,q_4)$, similar if and only if $|p_i - q_i| \leq c_i$ for $i=\{1,2,3,4\}$. After some tests have been made, they conclude that Halstead metric consistently detected similarities which did not exist and alternative metric was more reliable.

Faidhi and Robinson in 1987 claimed that their system is more accurate and sensitive than those in the literature [7]. They claimed that their set of empirical metrics is minimal and their system is using hidden measures that are hard to bypass for a beginner in programming course. They substantiate their claims with three case studies: varying the similarity gauge, checking whether the set of empirical metrics is minimal and checking the sensitivity of all metric sets.
In order to determine the features that may help in successful detection, Faidhi and Robinson selected two sets of measuring attributes. The first set with ten parameters is measuring certain general features of a program code that are most likely to be altered by novice programmer that commits plagiarism. Fourteen attributes in the second set have been extracted from other studies which attempted to quantify the inner and hidden features of a program's structure. Unfortunately in their report Faidhi and Robinson did not present a method for scoring the levels of similarity found between two programs.

Later, Verco and Wise [19] presented a slightly modified version of correlation scoring function used by Grier [19][8]. They determined "window size" by observing the difference values for each parameter, over a small group of programs. Grier's method requires the importance values to be larger than windows sizes which are not appropriate for Faidhi-Robinson system since window sizes could vary greatly. The increment is calculated using the importance value to scale the increment:

$$incr_i = \frac{(window_i - diff_i)}{window_i} \cdot importance_i$$

for the *i*-th parameter.

A year later, Jankowitz introduced a system that could be partially classified as a system with structure detection [10][10]. However, it is classified as attribute counting system since final decision about possible plagiarism is made by comparing counted parameters.
System performs analysis in two phases. In the first phase, the system scans through source program and for each program constructs static execution tree. When a program consists of only three or four procedures, they are analyzed by comparing each procedure in the first program with every other procedure in the second program.
In the second phase, the system compares static execution trees and is scanning for identical branches.

If they are found, the procedures attached to these branches are then analyzed statistically. This section is divided into two separate subphases.
The first subphase inspects the global characteristics of the procedures by comparing these parameters:

1) number of code lines (excluding all I/O statements),
2) number of variables used (excluding procedure and function calls with parameters),
3) number of used reserved words (excluding all Begins and Ends),
4) number of assignment statements,
5) number of If statements,
6) number of Repeat/While statements,
7) number of For statements,
8) number of Case statements,
9) number of With statements, and
10) number of procedure and function calls.

For each of the above pairs, the acceptance region is defined. After that, a mutual comparison of parameter values from two programs is performed. In case when values for same parameter lie within specified range, they are said to be equivalent and favorable-counter is incremented. If both parameter values are zero a null counter is incremented. At the end of this subphase a combined evaluation is made to determine whether or not the process continues into the next phase. Purpose of this evaluation is to select only those procedures, where high chances of matching are expected. Only those are further analyzed in second subphase where for each If, Repeat/While, For, Case and With statement a further analysis is made by recording the following measures:

1) length of statements (simple and compound),
2) number of reserved words,
3) number of variables used in statements,
4) number of If statements,
5) number of Repeat/While statements,
6) number of For statements,
7) number of Case statements,
8) number of With statements, and
9) reference sequence order.

Further analysis is made by comparing a tuple from statement in the first procedure with a tuple from statement in the second procedure. The number of successful matches found is compared with the number of unsuccessful matches. If the value is greater than some general correlation of X%, the procedures are said to be cohesive. If the value is even greater than some Y% then the procedures are said to be equivalent. Both X and Y values are arbitrary values that can regulate detection degree of plagiarism.

4 Detecting plagiarism by comparing structure

More powerful computers enabled use of different approaches in plagiarism detection. Implementations that use direct comparison of program structure replaced attribute counting systems. Most advanced and in literature frequently mentioned systems are MOSS, YAP3 and JPlag.

In 1988 Whale presented Plague **Error! Reference source not found.**. It works in three phases. In the first phase, a sequence of tokens is made for each compared file and structure profile is built, which summarizes the structures used in each program. In the second phase structure profiles are compared and pairs of nearest neighbors are determined by using a combination of language specific distance functions. After the second phase, majority of compared programs is expected to stay unpaired. Others move forward into the third phase where sequences of tokens are compared using a variant of the longest common subsequences algorithm.

Since Plague was implemented for only few programming languages and results are returned in two lists which need to be interpreted with help of manual, Wise consequently implemented system called YAP (Yet Another Plague) **Error! Reference source not found.**. System detects plagiarism in two phases: a generation phase, in which token file is generated for each scanned program, and comparison phase, in which pairs of token files are compared.
The process of creating a token file is same for each programming language:
1. In preprocess phase comments and print-string are removed, upper-case letters are translated to lower-case, letters not found in legal identifier are removed and a list of primitive tokens is made.
2. Synonym functions are renamed to a common name and blocks of functions/procedures are identified.
3. Identified function blocks are reordered in their calling order. First call to each function is expanded to full token sequence and subsequent calls are replaced with token FUN.
4. All tokens that are not from lexicon of target language are removed.
The generation phase is the same for all three versions of YAP. Different versions differ in comparison phase where YAP uses simple UNIX command sdiff, YAP2 uses Heckel's algorithm and YAP3 Running-Karp-Rabin Greedy-String-Tilling algorithm [21].

Later introduced JPlag uses the same comparison algorithm as YAP3 but it uses different optimizations for improving run time efficiency [15]. It is also available as a web service with user interface which generates HTML pages to present results. At the top

level, an overview page presents histogram of similarity values found for all program pairs. User (teacher, professor or assistant) can select each presented pair and make side-by-side comparison where corresponding code is colored in same color.

Well known system is also MOSS (Measure of Software Similarity) which uses winnowing algorithm [18]. It divides programs into substrings of length *k* (called *k*-grams), where *k* is a parameter chosen by the user. Each *k*-gram is hashed and a subset of these hashes is used for program's "fingerprint". MOSS is also implemented as plagiarism detection service available over Internet to all registered users.

5 Our framework

One of study programs at Faculty of Education University of Ljubljana is mathematics and computer science where students learn to become teachers of mathematics and computer science at primary and secondary schools. Among pedagogical, didactical and mathematical subjects they have several subjects from computer science with different level and contents of programming. Several years assistants and professors at homeworks and seminar works cope with copied source codes while being aware that they detect only minority of cases. Therefore, electronic plagiarism detection systems have the potential to help them detect the frauds. Some of these systems already exist but their usage is not yet well reported in the academic literature.

In order to investigate why and how students commit plagiarism, and how to help teachers about that we started a project of studying known plagiarism detection systems and developing a framework that would detect plagiarism between different types of files. At first stage we are focused on source codes of applications, at second we will focus on file types as are LaTeX, Mathematica, Matlab, Linux scripts, and later on other file types as well. We expect that procedures from detecting plagiarism in source codes can be applied to other types of files.

At first stage, which is currently under development, we are also developing our framework that will allow using different plagiarism detection systems and comparing their results. Teachers will use that framework in next study year for detecting plagiarisms for programming assessments in Pascal, C, JavaScript and PHP. The framework is written in Java, it works as a standalone application and as a web application. Its architecture is shown on Figure 1. It is comprised of five subsystems:

1) input system that manages different types of inputs (from files or any kind of input stream),
2) tokenizer system that parses given input and returns collected data from input as are. list of parsed attributes, counters, tokens of text etc.
3) plagiarism detection system that uses information from tokenizer system and performs detection based on algorithms of different detection systems
4) report system that generates reports about comparison of input, and
5) web services that allow using the framework over the web.

The framework is prepared very generally and allows adding subystems as are additional tokenizers and plagiarism detection systems. We strongly believe that in the future it will also contain our own plagiarism detection system.

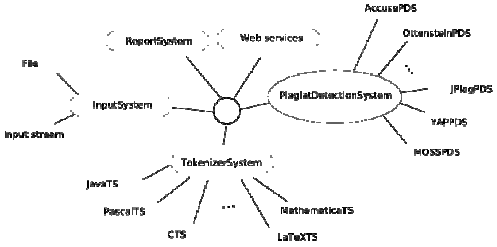

Figure 1: framework of our system

Currently we are gathering assessments from two different study courses:

1) "Computer practice" where students develop basic programs in Pascal. From 4 different assignments we have collected 105 assessments, from totally 38 students.
2) "Programming" where students develop advanced programs in Pascal, C and scripts in JavaScript and PHP. From 10 different assignments we have collected 287 assessments, from totally 69 students. In addition, some students (because of this project) when they submitted their work, they anonymously reported if they copied their work from someone else, from whom they copied, if they solved assessment collaboratively or have any other help. Assistant and professor who marked their assignments did not use this data.

All assessments are gathered with date and time of submission in order to observe the spreading speed of the same source code. Gathering assessments will finish in September, and thorough survey will be done.

6 Conclusion

Emerging technologies are definitively causing a shift in our mental world. The internet is growing at a remarkable rate and is fast becoming a common resource for everyone. With powerful search engines, finding and exchanging data became simple and fast. Demands for better study and research results are

higher than ever. Better means faster, with higher grades and with more published articles.

When the work of someone else is reproduced without acknowledging the source, this is known as a plagiarism. Many teachers discourage students from engaging in plagiarism on the grounds that it is fraudulent, deceptive and involves the theft of intellectual property, but when the plagiarism is detected there are no serious punishments or not at all. Consequently, all reasons to turn our culture into the plagiosphere are present, and research indicates that plagiarism is a problem in today's institutions of higher education. Sadly, research also reports that the magnitude of the plagiarism has increased in recent years.

In the paper, we treated the problem of plagiarism, in general and specifically for programming courses. Different kinds of plagiarism with source codes of programs and different systems to detect the plagiarism are presented. Although we restricted ourselves in the paper to education, the source code plagiarism is a problem not only in education but also for corporations which can be facing with source code and intellectual property theft, patent and copyright violation. It is important to have mechanisms to detect this and to react when plagiarism is detected.

References

[1] Bennett Roger, Factors associated with student plagiarism in a post-1992 university, Assessment & Evaluation in Higher Education, 30(2), 2005, pp. 137 - 162.

[2] Berghel H. L., Sallach D. L.: Measurements of program similarity in identical task environments, ACM SIGPLAN Notices, 1984, 19 (8), pp. 65 - 76.

[3] Bowyer W. Kevin, Hall O. Lawrence, Experience Using "MOSS" ta Detect Cheating On Programming Assigaments, 29th ASEE/IEEE Frontiers in Education Conference, 3, 1999, pp. 18 - 22.

[4] Bulut Necdet, Halstead Maurice H.: Impurities found in algorithm implementations, ACM SIGPLAN Notices, 1974, 9 (3), pp. 9 - 12.

[5] Donaldson John L., Lancaster Ann-Marie, Sposato Paul H.: A plagiarism detection system, ACM SIGSCE Bulletin (Proc. of 12th SIGSCE Technical Symp.), 1981, let. 13 (1), pp. 21 - 25.

[6] Encyclopedia Britannica, www.britannica.com

[7] Faidhi J. A. W., Robinson S. K.: An empirical approach for detecting similarity and plagiarism within a universitiy programming environment, Computers and Education, 1987, 11 (1), pp. 11 - 19.

[8] Grier Sam: A tool that detects plagiarism in Pascal programs, ACM SIGSCE Bulletin (Proc. of 12th SIGSCE Technical Symp.), 1981, 13 (1), pp. 15 - 20.

[9] Halstead Maurice H.: Elements of Software Science (Operating and Programming Systems Series), Elsvier Science Inc., New York, USA, 1977.

[10] Jankowitz Hugo Thomas: Detecting plagiarism in student Pascal programs, The Computer Journal, 1988, let. 31 (1), pp. 1 - 8.

[11] Jones Edward L.: Metrics based plagiarism monitoring, Journal of Computing Sciences in Colleges, 2001, let. 16 (4), pp. 253 - 261.

[12] Joy Mike, Luck Michael: Plagiarism in programming assignments, IEEE Transactions on Education, 1999, 42 (2), pp. 129 - 133.

[13] Ottenstein Karl J.: An algorithmic approach to the detection and prevention of plagiarism, ACM SIGSCE Bulletin, 1976, 8 (4), pp. 30 - 41.

[14] Parker Alan, Hamblen James O.: Computer algorithms for plagiarism detection, IEEE Transactions on Education, 1989, 32 (2), pp. 94 - 99.

[15] Prechelt Lutz, Malpohl Guido, Philippsen Michael: JPlag: Finding plagiarisms among a set of programs, Technical Report 2000-1, Fakultät für Informatik, Universität Karlsruhe, 2000, pp. 1 - 44.

[16] Rees Michael J.: Automatic assessment aids for Pascal programs, ACM SIGPLAN Notices, 1982, 17 (10), pp. 33 - 42.

[17] Schiller Rosita M., E-Cheating: Electronic Plagiarism, Journal of the American Dietetic Association, 105 (7), 2005, pp. 1058 - 1062.

[18] Schleimer Saul, Wilkerson Daniel, Aiken Alex: Winnowing: Local algorithms for document fingerprinting, Proceedings of the 2003 ACM SIGMOD international conference of management of data, 2003, pp. 76 - 85.

[19] Verco Kristina L., Wise Michael J.: Software for detecting suspected plagiarism: comparing structure and attribute-counting systems, Proceedings of the First Australian Conference on Computer Science Education, 1996, pp. 81 - 88.

[20] Webster's Online Dictionary, www.websters-online-dictionary.org

[21] Wise Michael: YAP3: Improved detection of similarities in computer program and other texts, Proceedings of the twenty-seventh SIGCSE technical symposium on computer science education, 1996, pp. 130 - 134.