SOURCE CODE PLAGIARISM DETECTION FOR PHP LANGUAGE

Richard Všianský[1], Dita Dlabolová[1], Tomáš Foltýnek[1]

[1] *Mendel University in Brno, Czech Republic*

EUROPEAN JOURNAL OF BUSINESS SCIENCE AND TECHNOLOGY

Volume 3 Issue 2
ISSN 2336-6494
www.ejobsat.com

ABSTRACT

This paper introduces a system for detection of plagiarism in source codes written in the PHP computer language, part of the plagiarism detection tool Anton. We used the greedy string tiling algorithm together with tokenization and hash calculation. The efficiency of the system was tested on both an artificial dataset and on real data coming from a course taught at our university. Our results are compared with other similar systems and solutions, concluding that Anton can detect all examined types of plagiarism with higher accuracy than other systems.

KEY WORDS

source-code plagiarism, anti-plagiarism system, PHP, Anton

JEL CODES

C88, I23

1 INTRODUCTION

Since 1990s, academic integrity has become a central preoccupation for all stakeholders in higher education (Bretag, 2015). In earlier stages of this period, universities focused mainly on plagiarism. Even as the scope has become broader, plagiarism remains one of the most important academic integrity issues appearing in student assignments undertaken individually or in groups, without direct supervision. In the digital era, massive amounts of information are available to reuse for anyone struggling with an assignment who is tempted to plagiarise (Flores et al., 2011).

Student writers are typically advised to build on the ideas of the authors they have read and to incorporate the material in one of three ways: paraphrase, summary or quotation (Jamieson, 2015). In programming assignments, of course students should make use of the work of the others as well. However, the methods for doing so are completely different. Students are advised to use standard libraries and standard

VŠIANSKÝ, Richard, DLABOLOVÁ, Dita, and FOLTÝNEK, Tomáš. 2017. Source Code Plagiarism Detection for PHP Language. *European Journal of Business Science and Technology,* 3 (2): 106–117. ISSN 2336-6494, DOI http://dx.doi.org/10.11118/ejobsat.v3i2.100.

algorithms. On the other hand, these algorithms are often required to be coded by students and not simply downloaded from the internet.

Parker and Hamblen (1989, p. 94) define plagiarism in source code, namely a plagiarised program, as: "a program which has been produced from another program with a small number of routine transformations". The definition differs from "standard" definition of plagiarism, where the fact that someone else's work is presented as author's own is substantial. Unlike natural text, source code has its given precise structure with many possibilities of small changes (the above mentioned routine transformations) which may – without actual knowledge of what the source code is about – change the text completely and hide any plagiarism.

There are several common ways of changing source code in order to hide plagiarism (Clough, 2000). Joy and Luck (1999) distinguish two basic types: lexical changes and structural changes. Mirza and Joy (2015) enumerate following categories of changes: in comments, in identifiers, in declarations, added (extra) content of different kind, changes in the structure of selection statements, and in decision logic.

Joy et al. (2011) investigated students' plagiarism in source codes, focusing on understanding and reasons through a survey performed on 18 universities in United Kingdom. Their findings suggest that even if 94% of the students participating in the survey understand that copying the source code from somewhere else is plagiarism, they fail to understand other types of plagiarism such as self-plagiarism, collusion with peers or conversion of the code from different programming language.

In natural text, not every similarity can be considered plagiarism and not every plagiarism can be considered as an academic integrity breach (Bretag, 2015). Analogically, not every similarity in source code constitutes plagiarism. Krpec (2015) enumerates some legitimate causes of similarities: properly referenced third-party source code, automatically generated code, commonly used identifiers, commonly known algorithms. For these reasons, simple text based comparison tools that do not consider the special features of source code are likely to produce many false positives while failing to detect plagiarism that has been intentionally obfuscated.

The system discussed in this paper is called Anton, which is an abbreviation of "anti-plagiarism online", and was originally developed at Faculty of Business and Economics, Mendel University in Brno (MENDELU), Czech Republic as a university's own solution for detecting similarities in natural text documents, namely final theses (Foltýnek et al., 2009). It was used as an integral part of University Information System (UIS, `https://is.mendelu.cz`) for checking final theses from 2009 to 2014. In 2014 it was replaced by the Czech national system "Theses" (`www.theses.cz`). In 2015 Anton was re-launched as a standalone online tool (at `https://anton.mendelu.cz`) mainly for experimental purpose including its further development and growth (Všianský and Dlabolová, 2016). In 2017 new functionality for detection similarities in PHP source codes was added to the system (Všianský, 2017). The examination of this functionality and its efficiency will be discussed in this paper.

The PHP language was chosen as the one that would be implemented in Anton as the language that is used in teaching of different computer science courses at MENDELU – primarily in the course 'Application Software Programes' (ASP). In this course students write their projects in this language, so there was a need for teachers to be able to check the projects and find possible instances of plagiarism or copying from colleagues (Všianský, 2017).

2 THEORETICAL FRAMEWORK

Literature and researchers describe multiple approaches for detecting similarities or plagiarism in source codes. The approaches can be divided in two main groups – feature comparison and structure comparison (Prechelt et al., 2000; Arwin and Tahaghoghi, 2006). Feature comparison is based on a certain software metric (e.g. number of comment lines or number of some particular kind of tokens), structure comparison is based on the similarity of the structure of the investigated source codes – source codes are parsed to tokens and the tokens are compared (Arwin and Tahaghoghi, 2006). Another approach which can help to recognize plagiarism in source codes and also to identify the author of the source code is analysis of the coding style (Mirza and Joy, 2015).

Detailed comparison of systems for the detection of plagiarism in source codes can be found e.g. in (Lancaster and Culwin, 2004). In the following paragraphs, selected systems which enable users to compare (at least partly or roughly) source codes written in the PHP language are briefly described.

The MOSS system (Measure of Software Similarity) supports several programming languages, some of which are related to PHP, but does not support PHP. It uses structural comparisons, namely the so-called winnowing method described in the paper "Winnowing" (Schleimer et al., 2003). It is free for non-commercial use as a web service (MOSS, 2017) and it belongs among popular tools (Lancaster and Culwin, 2004).

JPlag is another program that supports several programming languages and text in natural language, but also excludes PHP. It uses structure comparison, it is offered for free (until recent time as a free web service), and is licensed under GNU General Public License (jPlag, 2017; Prechelt et al., 2000). Its main advantages are reliability and support of the main programming languages, which make it one of the most popular tools both among teachers of programming at MENDELU and among experts (Lancaster and Culwin, 2004).

Sherlock is an open source system developed at University of Warwick that checks similarities in source-codes (procedural and object-oriented languages with optimizations for Java) and in text in natural language (Joy, 2014). The system itself does not have a graphical user interface (GUI, see Sherlock, 2017), but it is incorporated in the University of Warwick's online submission system BOSS, which provides it the GUI (Joy, 2014). It uses its own algorithm for structure comparison. According to the authors, the accuracy of the results is comparable with other systems, but Sherlock provides the results in significantly shorter time (Joy and Luck, 1999; Mozgovoy et al., 2005).

Checksims is an open source system developed at Worcester Polytechnic Institute. According to Lauer (2015) and Checksims (2015) it was under active development until 2015. It is not constrained for use over a specific language or group of languages. It works with any programming language, the primary algorithm used for the detection of similarities is Smith-Waterman algorithm. The authors compared Checksims with MOSS and claim that the system has the same success as MOSS within large collections of source codes and that it was even more successful with small collections (Heon and Murvihill, 2015).

The PHP plagiarism recognizer implemented at the Faculty of Information technologies, Brno University of Technology uses Halstead metrics and the Levenshtein algorithm applied to JSON object for first check. Suspicious pairs are then examined more carefully by document fingerprint using the winnowing algorithm and an abstract syntax tree comparison using Sasha's algorithm. Details about the results are not provided, authors just admit that their "results are not totally accurate every time yet" and rely on users' judgement (Krpec, 2015).

Moussiades and Vakali (2005) describe their solution named PDetect which is mainly intended for bulk processing of source codes and searches groups (clusters) of similar parts of source code. The system was tested on C++, but its authors claim that keywords for

any programming language can be added for adaptation to any programming language.

Cross-language plagiarism detection is discussed in paper "Towards the detection of cross-language source code reuse" (Flores et al., 2011) whose authors found that methods applied for natural text (specifically n-gram comparison) work for Java, C and Python too. The other method might be comparison of an intermediate code produced by a special compiler suite. This method is in fact a monolingual comparison, but depends on existence of compilers for different languages producing comparable intermediate code (Arwin and Tahaghoghi, 2006).

3 METHODOLOGY AND DATA

Before the implementation of the new module, an analysis of the current university solution and requirements for a new solution was performed. Lecturers in the course ASP used only a text-based recognition in jPlag. This solution does not support PHP language natively, hence it is inappropriate to be used for recognition of similarities in PHP source codes.

In its initial phase the Anton system used only text-based methods to detect similarities in documents of different text formats. Using this method, documents are processed through four phases. First, the documents are converted to plain text. This means that all information which does not constitute the content itself is deleted. The second phase reduces the text further, e.g. deleting all words shorter than three characters, stop words, numbers, etc. The third phase makes hashes from the remaining words and lastly, in the fourth phase the system compares all hashes and calculates the final results (Floryček, 2015).

For comparison of PHP source codes and source codes in general, text-based methods are unsuitable. As it was mentioned earlier, there are many transformations of a source code which could lead to a different text form. These transformations can be made easily (e.g. to change names of variables, add comments, swap cycles, etc). Text based methods cannot detect these transformations. Therefore, methods developed specifically for source code plagiarism detection need to be used.

The *Greedy string tiling* algorithm (Prechelt et al., 2000) with tokenization (Murao and Ohno, 2010) can be used to compare PHP source codes in the system Anton (Všianský, 2017), which is briefly described below. Firstly, the source code is transformed to a sequence of tokens with tokens being the smallest semantic element of a source code (Shao, 2015). These tokens, then, represent a structure of the document. PHP includes its own tokenizer, which can be used via function *token_get_all* (The PHP Group, 2017) which transforms the code to a sequence of numbers. In addition, the function can record also content of the tokens and their position in the document (Všianský, 2017).

The sequences of tokens are compared by *Greedy string tiling* in the next step. This algorithm, which is also used in jPlag, firstly takes the sequences from a document containing source code (document A) and compares them to sequences from the second document (document B). If they match, the another pair is compared. All results are recorded and all matched tokens are marked, which means they don't have to be compared anymore.

If the pair does not match, the system continues and compares all other possible combinations of both documents. All matches must meet the minimum threshold and specified length, otherwise the match is not recorded. A longer length provides more accurate results, but it can neglect smaller changes in documents. The length also depends on the structure of the programming language and purpose of the document. With more complex projects and documents, the length can be increased. For purpose of this study the length of seven tokens has been chosen as most appropriate to students in beginner 'Application Software Programes' classes, because students in beginner classes tend to use less complicated codes.

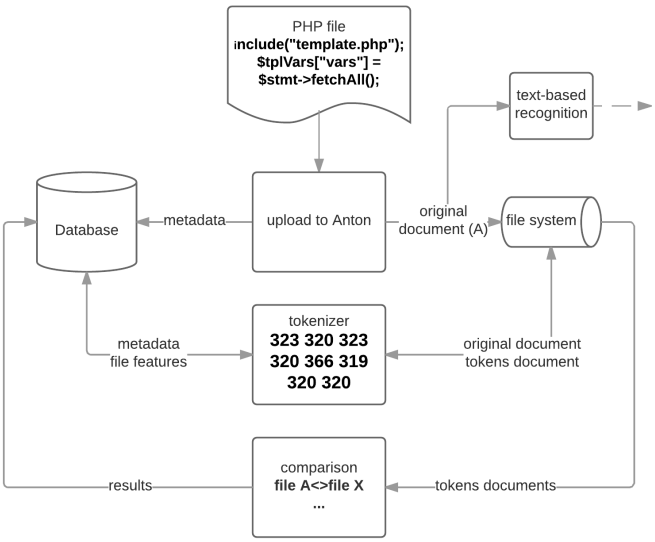

Fig. 1: Processing of a file with source in Anton

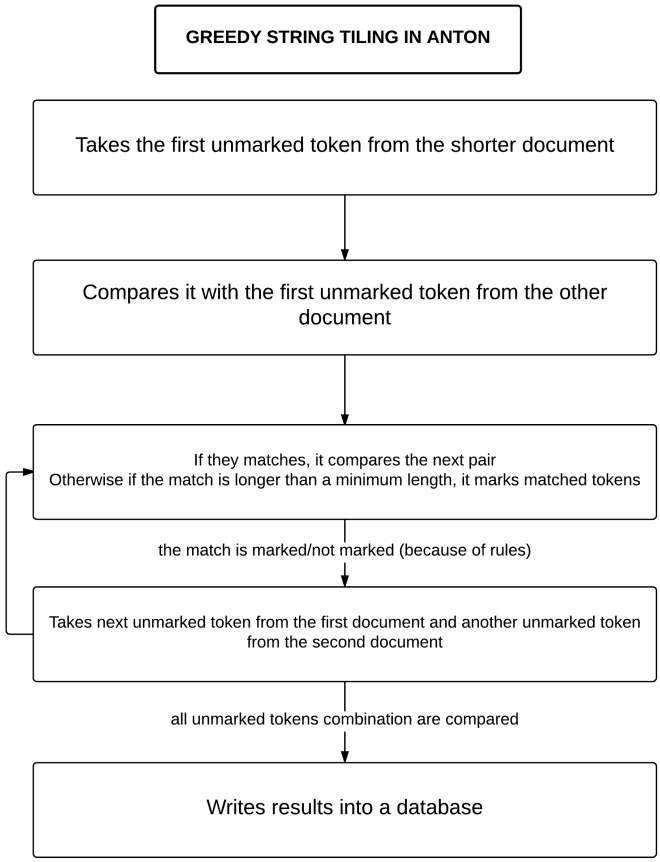

Fig. 2: Greedy String Algorithm in Anton

For comparison of source code, the text-based method is also used. However, it only compares all strings from the document (Všianský, 2017). In this manner, a user can recognize similarities among names of functions, etc. E.g. if a student just translates the names to a different natural language, like translated Czech names of functions to English, it is recognizable, as the structure and meaning are visibly the same. Some basic features of the document are recorded too. In the current iteration, the system saves counts of lines, strings, tokens, comments and whitespaces but this list can be adjusted in the future to include many different characteristics that can point to a specific writing style of particular student.

For implementation of the system, PHP language was used. The main reason was the availability of the above-mentioned native PHP tokenizer and the fact that the rest of the system Anton is written in PHP as well.

The module for detection of similarities in PHP source codes was implemented as a part of the current Anton system available online. Before uploading a file, users select whether they intend to work with natural language texts or with the source codes.

The implementation is divided into several parts. The first part is a decoder. In this part, all new uploaded documents are inserted into a function, which creates tokens from the documents and inserts them into a new file on the file system. This function called getTokensPHP uses a parser file. This file is another part of the implementation, it contains the tokenizer of PHP and it produces results in a correct format. The format is designed specifically to contain all tokens information in one array in a given order. A developer can add a new parser of a different language using a new function containing a tokenizer of the language with the appropriate output.

The last part is the comparison. The system compares all not-compared documents from one course directory (defined before uploading documents) with each other. The function 'stringTiling' is used, unmarking all tokens in both documents, i.e. leaving all tokens are marked with a number zero. Then, according to the algorithm, all unmarked tokens are tested and if they match, they are marked (with number one). This function is using the minimal length of non-overlapping matches (the threshold) which was set previously. The results are saved in the database.

The current MySQL database was used to record the results. One of the most important tables is *tokens_pos*, which records a position of a token sequence across two documents with lengths and line position specifically according to each document. Data stored in this table then allows Anton to show similarities in graphical form. Percentages of similarities are stored in two separate tables. One of them contains a similarity ratio from the point of view of one particular document; the second one contains similarity of two documents. The second index shows similarity emphasizing common length, whereas the first result could be distorted by different lengths.

Our test data were acquired in two different ways. The first dataset consists of a set of real students' works in a course "Application Software Programmes" (ASP), where students learn basics of PHP language taught by members of the Faculty of Business and Economics, MENDELU. Overall, 66 files of student projects were collected. From all these files, the segments written in other languages such as CSS and HTML were removed. These data were chosen because the projects have been written in clear form of PHP without use of any frameworks or libraries.

The second dataset consisted of 7 files was created specifically for testing the system. Each of the created files represented a different method of source code plagiarism:

- The first file was a basic file with a solution for the task of creating a profile page of a user obtained from the author's university project which was elaborated in ASP, (the same project as in the first dataset).
- The second file was a modified version of the first one with changed identifiers and added comments.
- The third file was made of 50% of the first file with the second half composed of a file originating from a completely different

project. This procedure was chosen because the file should be structured as a normal PHP file code. Only the first half should be matched with other files in comparison.

- The fourth file was a new file with no connection to the previous files.
- The fifth was a combination of the third and the fourth file.
- The sixth file was the same as the first one with swapped halves.
- The last file was a file containing solution of the same task but had been created by another student.

By using these different types of files, files it was possible to determine how well the system was able to handle different phenomena including changing identifiers, packaging functions, replacing parts of a code, adding and deleting comments. It was also possible to observe the results of situations in which two projects from different students working on the same task are combined.

4 RESULTS

The similarity detection module in Anton is in the first place intended for teachers of basics of programming in the PHP language, who expect their students to submit individual assignments. Having tens of the source code documents with the solutions from the students, the user (teacher) wants to be sure that the students did not copy from each other. After uploading all the assignments, Anton reports the similarities among the documents. Since the PHP language has quite limited vocabulary, the teacher with regards to complexity of the assignments expects some natural matches among the documents, nevertheless the matches which are above the expected threshold (e.g. above 25%) are suspicious. Anton enables visualisation of the matches in the suspicious documents (example can be seen at the Figure 3) to help the teacher to decide whether the match is plagiarism or not. The final decision about plagiarism is up to the teacher; anyway the aim of Anton is to provide the teacher the best possible support.

The first round of testing of problem solutions was performed on the artificial data prepared specially for this purpose which are described above. The results are displayed in Tab. 1. This table shows the expected results based on the previous knowledge of the particular data and the results provided by Anton.

The results showed promise for the utility of the system. They did not contain any false-negative results (i.e. all similarities were revealed).

The test case A shows that the system is resistant to one of the most prevalent form of plagiarism – renaming variables and identifiers, or adding comments. The system also identified the identical parts of the code as it is visible from the test results B–E. From the graphical representation of the matches, it can be seen that the system correctly identified the same parts. This proves that the system is resistant to swapping larger parts of codes. The resistance to swapping smaller pieces of code depends on the settings of the minimal length of match (i.e. number of tokens), which needs to be set carefully because low threshold would cause reporting of coincidental matches. The coincidental matches – i.e. false-positive results are visible in the cases F and G. There were similarities discovered even in source codes which were not supposed to be similar at all. The reason for this is the simplicity of the PHP dictionary and the tokenizer which incorporates only limited amount of types of tokens. Dictionaries of all programming languages are less complex than those of natural languages: programmers use only a few keywords whereas natural languages have thousands of words). Source codes can contain different functions and variables with the same structure. While PHP source codes can contain different functions and variables within the same structure, the functions appear identical to the tokenizer, which does not take

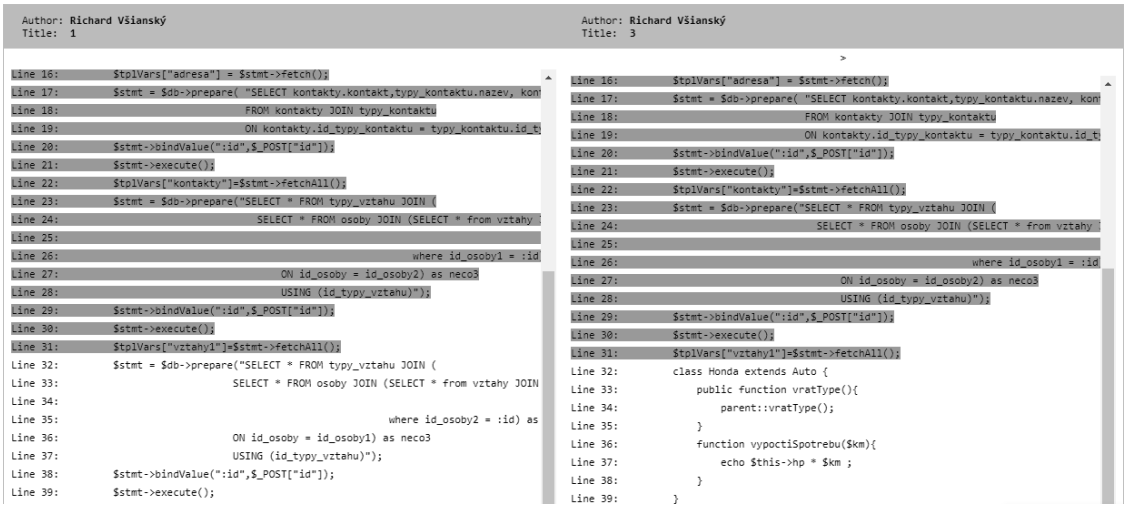

Fig. 3: Example of a screen with a result of comparison of two documents

Tab. 1: Test results over artificial test data

Test case	The compared file	Same data	Expected match	Real match	Modification
A	1	2	100%	100.00%	Identifiers, functions and variables were changed, extra comments
B	1	3	50%	59.40%	Identical half
C	5	3	50%	49.20%	Identical half
D	5	4	50%	69.80%	Identical half
E	1	6	100%	100.00%	Switched halves of the code
F	7	1, 2	Two different files	55.24%	Coincidental similarity in two works on the same assignment
G	4	1, 2, 7	0%	20.62% 20.62% 17.01%	Two different files

into account the effect of context. In this case, the user must take care to define a minimum match length that will exclude most false positive results, while retaining sufficient accuracy to find significant similarities. In the future, Anton could be used with more complicated languages where it would have greater accuracy. Better functionality could also be achieved by developing a tokenizer with the ability to recognize context (e.g. it could divide functions according their return types, position etc.). These are issues for further development.

In terms of the dataset of the 'Application Software Programmes' course, the results are less meaningful, because they cannot be compared to expected outcomes. However, a few observations can be noted. The module can find and identify 100% plagiarism, whereas currently used jPlag finds only 96% similarity across same files. This difference is caused by configuration of jPlag, which examines words only with some minimal length, to allow for the fact that many files have a high degree of similarity because students are using same learned functions, algorithms and techniques. This allowance is also made because in this course, the codes are not very complex, as students have just started to learn how to write a PHP code. Users should take care to set appropriate thresholds for recognition and observe all similarities personally to decide what size words may, when duplicated, constitute plagiarism. The graphical overview of similarities should make this decision easier.

Users can see which parts of the code match with which part from the other document, they can also see the results of comparison (percentage of similarity). The system also shows the text-based similarities between files' strings for better understanding of the code contents (see Fig. 3).

5 DISCUSSION

The issue of searching for plagiarism in source code is often discussed in the academic environment, where educators must address possible academic integrity breaches of their students. A variety of systems are available for revealing similarities in source codes with MOSS and jPlag among the most popular among specialists over the world (MOSS, 2017; jPlag, 2017; Lancaster and Culwin, 2004), but they do not support all programming languages (e.g. the PHP, which is needed in the programming courses at Mendel University in Brno) and do not meet the needs of all teacher. For this reason, many educational institutions develop their own solutions – University of Warwick's system, Sherlock (Joy, 2014); Worcester Polytechnic Institute's system, Checksims (Heon and Murvihill, 2015); Brno University of Technology's system, (Krpec, 2015); and MENDELU's system Anton which is discussed in this paper.

Anton was tested on two sets of test data. One set consists of artificial data made directly for the testing purpose. Results were satisfactory – the system detected all similarities, however there were a significant number of false-positive matches caused mainly by the nature of the testing data and of the PHP language itself. The high match in the case F depicted in the Tab. 1 was also caused by the fact that the tested source codes were school assignments – the assignment was the same for both source code, it was rather uncomplicated and both authors used similar approach that they learned at school.

This is certainly an issue for further development as 20% rate of false positives would discourage teachers from using the system. Because the system was designed primarily to detect deliberately obfuscated plagiarism a high false positive ratio is predictable. Moreover, false positives are more likely in programming language than in a natural language due to limited vocabulary and due to general recommendation to reuse source code. For example, in PHP, there are only 135 possible tokens which naturally lead to false positives. This can be addressed by using a filter set to hide matches below 20, 25 or even 30 per cent so user is not confused. On the other hand, small changes remain undetected, which should not be critical problem. Setting the correct threshold will be considered as soon as the system is used more widely.

The other set of data tested here are 'real' data from students – a set of 66 assignments from a course taught at MENDELU. Anton provided better results using these data than jPlag which had been used for the examination of the assignments. Since the jPlag is not appropriate for the PHP language, it did not reveal all matches.

In our opinion Anton's user interface is also more user-friendly than jPlag, also Anton's visualisation of the resulting matches provides more information. JPlag highlights the matches in the source codes, Anton also depicts text similarities in the names of the functions and a table of properties, which serve as additional information for the user to support him in the decision whether the matches are plagiarism or not.

As for the comparison with other systems mentioned earlier in this work – the widely popular system MOSS is available only as an online service and a Linux script needs to be used for the submission (MOSS, 2017). The Sherlock system provides the results of the matches only in plain text form. The system's output is a list of pairs of filenames and the percentage value of their match. There is no information about which particular part of the source code

caused the match (Sherlock, 2017). Also, the system Checksims operates only in a command line and does not have any graphical user interface. According to available information the system PDetect seems not being developed after year 2005 (Moussiades and Vakali, 2005). According to Krpec (2015) their solution from Brno University of Technology does not always provide accurate results. Hence none of these systems fully meets the requirements and needs of the educators at MENDELU.

In the current state Anton enables to work with the files only separately, it does not support any bulk operation, which might be considered as a drawback for its use as a tool for examination of school assignments. Hence for now, the system has been tested by several individual teachers. All teachers at the department will be encouraged to use it as soon as the batch upload of files is available. The implementation of the batch operations is currently under development and it should be ready for usage during the spring semester of the academic year 2017/2018 when the course ASP is being taught and when the anti-plagiarism system in this course will be needed. Hence the teachers will be able use the system comfortably for examination of the projects without any limitations.

6 CONCLUSIONS

This paper deals with the system Anton and its extension for detection of similarities in PHP source codes. The principle of searching for similarities in source codes in Anton is based on tokenisation (using a PHP native tokenizer) and the tokens are compared, using the Greedy string tiling algorithm. The reported matches are reported according to the setting of minimal threshold (i.e. the minimum number of matching tokens). For the purpose of this study, the threshold was set to seven.

When used to evaluate samples in the PHP language, Anton provides more accurate results then jPlag. The results are depicted in the graphical form where matches are highlighted and also further information on the source code's similarities are provided to the user.

The topics for future development include enabling bulk operations and accuracy improvement of the thresholds so the system so as to reduce false-positive results.

Anton's specialization on PHP language can be seen as a limitation, but other languages can be added easily by addition of the appropriate tokenizers. One main goal of future development is to recognize similarities across programming languages. When it becomes possible to convert tokens from different programming languages into comparable tokens, we will then gain the significant advantage of being able to discover source codes which were rewritten from one programming language to another.

7 ACKNOWLEDGEMENTS

This paper was supported by Internal Grant Agency of Faculty of Business and Economics of Mendel University in Brno (project code: PEF_TP_2016001).

The paper is based on results of bachelor thesis of Richard Všianský (2017).

8 REFERENCES

Arwin, C. and Tahaghoghi, S. M. M. 2006. Plagiarism Detection across Programming Languages. In: *Proceedings of the 29th Australasian Computer Science Conference*, vol. 48, pp. 277–286.

Bretag, T. 2015. *Handbook of Academic Integrity*. USA: Springer. ISBN 978-981-287-097-1.

Clough, P. 2000. *Plagiarism in Natural and Programming Languages: an Overview of Current Tools and Technologies*. Sheffield: Department of Computer Science, University of Sheffield. [online]. Available at: `http://ir.shef.ac.uk/cloughie/papers/plagiarism2000.pdf`. [Accessed 2017, October 31].

Flores, E., Barron-Cedeno, A., Rosso, P. and Moreno, L. 2011. Towards the Detection of Cross-Language Source Code Reuse. *Proceedings of 16th International Conference on Applications of Natural Language to Information Systems, NLDB2011*. Springer. ISBN 978-3-642-22326-6.

Floryček, J. 2015. *Optimalizace antiplagiátorského řešení na Mendelově univerzitě v Brně*. Brno: Mendelova univerzita v Brně. [online]. Available at: `http://theses.cz/id/vgizl0/zaverecnaprace.pdf`. [Accessed 2017, October 31].

Foltýnek, T., Procházka, T. and Rybička, J. 2009. Plagiarism Detection System at Mendel University in Brno, Czech Republic. [DVD-ROM]. In *IVKI 2009. Inovácia výskumu katedier informatiky*, pp. 50-53. ISBN 978-80-8094-579-4.

Checksims. 2015. *GitHub – Checksims*. [online]. Available at: `https://github.com/Checksims/checksims`. [Accessed 2017, December 20].

Heon, M. and Murvihill, D. 2015. *Program Similarity Detection with Checksims: A Major Qualifying Project Report*. [online]. Available at: `https://web.wpi.edu/Pubs/E-project/Available/E-project-043015-122310/unrestricted/CheckSims.pdf`. [Accessed 2017, October 25].

Jamieson, S. 2015. Is it Plagiarism or Patchwriting? Toward a Nuanced Definition. In Bretag, T. (ed.). *Handbook of Academic Integrity*. USA: Springer. ISBN 978-981-287-097-1.

JPlag. 2017. *JPlag – Detecting Software Plagiarism*. [online]. Karlsruhe: Institute for Program Structures and Data Organization. Available at: `https://jplag.ipd.kit.edu/`. [Accessed 2017, April 5].

Joy, M., Cosma, G., Yau, J. Y. and Sinclair, J. 2011. Source Code Plagiarism – A Student Perspective. [online]. *IEEE Transactions on Education*, 54 (1), 125–132. DOI: 10.1109/TE.2010.2046664. Available at: `http://ieeexplore.ieee.org/document/5451097/`. [Accessed 2017, October 25].

Joy, M. and Luck, M. 1999. Plagiarism in Programming Assignments. [online]. *IEEE Transactions on Education*, 42 (2), 129–133. Available at: `https://pdfs.semanticscholar.org/f161/83ebb570fe9d485a5d36f415e94215cf9ad3.pdf`. [Accessed 2017, October 27].

Joy, M. 2014. *Sherlock – Plagiarism Detection Software*. [online]. Available at: `http://www2.warwick.ac.uk/fac/sci/dcs/research/ias/software/sherlock/`. [Accessed 2017, October 27].

Krpec, O. 2015. Plagiarism Recognizer in PHP Source Code. *Excel@FIT 2015 Conference Proceedings*. [online]. Available at: `http://excel.fit.vutbr.cz/submissions/2015/076/76.pdf`. [Accessed 2017, October 26].

Lancaster, T. and Culwin, F. 2004. A Comparison of Source Code Plagiarism Detection Engines. [online]. *Computer Science Education*, 14 (2), 101–112. DOI: 10.1080/08993400412331363843. Available at: `http://www.tandfonline.com/doi/abs/10.1080/08993400412331363843`. [Accessed 2017, October 25].

Lauer, H. C. 2015. Extensions and Enhancements for Checksims. In: *Computer Science WPI*. [online]. Available at: `http://web.cs.wpi.edu/~lauer/MQP/Checksims_MQP_topics.htm`. [Accessed 2017, October 25].

Mirza, O. and Joy, M. 2015. Style Analysis For Source Code Plagiarism Detection. In: *Plagiarism Across Europe and Beyond: Conference Proceedings*. Brno: MENDELU, pp. 53–61. ISBN 978-80-7509-267-0.

MOSS. 2017. *A System for Detecting Software Similarity* [online]. Available at: `http://theory.stanford.edu/~aiken/moss`. [Accessed 2017, April 30].

Moussiades, L. and Vakali, A. 2005. PDetect: A Clustering Approach for Detecting Plagiarism in Source Code Datasets. [online]. *The Computer Journal*, 48 (6), 651–661. DOI: 10.1093/comjnl/bxh119. Available at: `http://academic.oup.com/comjnl/article/48/6/651/358280/PDetect-A-Clustering-Approach-for-Detecting`. [Accessed 2017, October 27].

Mozgovoy, M., Fredriksson, K., White, D., Joy, M. and Sutinen, E. 2005. Fast Plagiarism Detection System. In Consens, M. and Navarro, G. (eds.). *String Processing and Information Retrieval*. [online]. Springer, pp. 267–270. DOI: 10.1007/11575832_30. Available at: http://link.springer.com/10.1007/11575832_30. [Accessed 2017, October 27].

Murao, H. and Ohno, A. 2011. A Two-step In-class Source Code Plagiarism Detection Method Utilizing Improved CM Algorithm and SIM. *International Journal of Innovative Computing, Information and Control*, 7 (8), 4729–4739. Available at: http://www.ijicic.org/ijicic-10-05012.pdf. [Accessed 2017, October 31].

Parker, A. and Hamblen, J. O. 1989. Computer Algorithms for Plagiarism Detection. [online]. *IEEE Transactions on Education*, 32 (2), 94–99. DOI: 10.1109/13.28038. Available at: http://ieeexplore.ieee.org/document/28038/. [Accessed 2017, October 31].

Prechelt, L., Malpohl, G. and Philippsen, M. 2000. *JPlag: Finding Plagiarisms Among a Set of Programs*. Karlsruhe: Fakultat fur Informatik Universit at Karlsruhe. [online]. Available at: http://page.mi.fu-berlin.de/prechelt/Biblio/jplagTR.pdf. [Accessed 2017, October 31].

Schleimer, S., Wilkerson, D. S. and Aiken, A. 2003. Winnowing. In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data – SIGMOD '03 New York*, p. 76. DOI: 10.1145/872757.872770. [online]. Available at: http://portal.acm.org/citation.cfm?doid=872757.872770. [Accessed 2017, October 25].

Shao, Z. 2015. *Compilers and Interpreters*. New Haven: Yale University. [online]. Available at: http://flint.cs.yale.edu/cs421/lectureNotes/c02.pdf. [Accessed 2016, November 17].

Sherlock. 2017. *The Sherlock Plagiarism Detector*. [online]. Available at: http://www.cs.usyd.edu.au/~scilect/sherlock/. [Accessed 2017, October 27].

The PHP Group. 2017. *PHP – Tokenizer*. [online]. Available at: http://php.net/manual/en/book.tokenizer.php. [Accessed 2017, May 14].

Všianský, R. 2017. *Rozpoznávání podobností zdrojových kódů v systému Anton*. Brno: MENDELU.

Všianský, R. and Dlabolová, D. 2016. Deployment and Improvements of System Anton. In: *PEFnet 2016*. Brno: MENDELU.

AUTHOR'S ADDRESS

Richard Všianský, Department of Informatics, Faculty of Business and Economics, Mendel University in Brno, Zemědělská 1, 613 00 Brno, Czech Republic, e-mail: xvsiansk@mendelu.cz

Dita Dlabolová, Department of Informatics, Faculty of Business and Economics, Mendel University in Brno, Zemědělská 1, 613 00 Brno, Czech Republic, e-mail: dita.dlabolova@mendelu.cz

Tomáš Foltýnek, Department of Informatics, Faculty of Business and Economics, Mendel University in Brno, Zemědělská 1, 613 00 Brno, Czech Republic, e-mail: tomas.foltynek@mendelu.cz