Journal of Computing and Information Technology - CIT 21, 2013, 3, 185–194
doi:10.2498/cit.1002193

A New Model for Semiautomatic Student Source Code Assessment

Emil Stankov, Mile Jovanov, Ana Madevska Bogdanova and Marjan Gusev

Faculty of Computer Science and Engineering, University Ss. Cyril and Methodius, Skopje, Macedonia

Programming courses at university and high school level, and competitions in informatics (programming), often require fast assessment of the received programming tasks solutions. This problem is usually solved by the use of automated systems that check the produced output for some test cases for every solution.

In this paper, we present a new model for semiautomatic student source codes assessment for a given programming task, based on our approach of representation of the program codes as vectors. It represents a human and computer collaborative effort. Our research on the use of these vectors in data mining analysis of the source codes, with the achieved excellent results on the number of correctly clustered items, is a solid foundation for the proposed model. At the end, we present the results of the preliminary testing of the model.

Keywords: programming code, source code assessment, code similarity, clustering, human – computer collaboration

1. Introduction

Programming courses at university and high school level (especially introductory ones) often include lots of exercises in order to ease the adoption of the programming language syntax, and also to help the students develop algorithmic way of thinking. Since programming is a compulsory course in every computer science educational curriculum, usually lots of computer science students enroll in these courses. This leads the course lecturers to the problem of mass number of solutions to exercises that have to be graded – the assessment can no longer be done manually in a reasonable amount of time. For example, at our institution, Faculty of Computer Science and Engineering in Skopje, in 2012, around 600 students enrolled in the first-year introductory programming course. During the 12 lecture weeks in the semester, every student was required to solve 3 to 5 exercises on a weekly basis. Cumulatively, this means that approximately 30000 solutions to exercises have to be assessed during the semester. Moreover, it is highly desirable that every set of 3 to 5 exercises is assessed by the end of each week in which it is presented to the students, in order for the students to be able to continually keep track of the progress they make in learning programming.

The need for fast assessment has been perceived even earlier, in the organization of competitions in informatics. These competitions were introduced for the first time approximately 40 years ago, with the idea to attract talented young people to the art of algorithmic thinking and computer programming. The term "competition in informatics" usually represents a synonym for competition in algorithmic programming (other types of competitions include architecture, design, development, specification, assembly, testing scenarios, etc.). Nowadays, the international programming competitions require submission of program codes – solutions to specific algorithmic problems, from the participants. The main focus of the competitions is on the design of appropriate algorithms for solving the problems at hand.

In this paper we present a new model for a semiautomatic student source codes assessment for a given programming task. The model represents a human and computer collaborative effort. The intention is to offer a better assessment of programming solutions than the traditional automated assessment typically employed at competitions in informatics and in educational environment.

The rest of the paper is organized as follows: in Section 2 we describe the automated source code assessment performed commonly by the contemporary grading systems and emphasize the main problem with this type of assessment. In Section 3 there is a description of the related work on source code analysis. In Section 4 we review our hybrid approach for source code similarity detection, presented in [11]. In Section 5 we describe the new model for semiautomatic source code assessment, and in Section 6 we elaborate the evaluation of this model and analyze the obtained results. Finally, in Section 7, we give a conclusion and some directions for future work.

2. Contemporary Systems for Automated Source Code Assessment

In most cases, the competitions in informatics are based on automated assessment of the submitted solutions. The automation of the assessment is necessary not only because of the large amount of solutions, but also in order to have the results in reasonable time. This is accomplished by running them on batches of input test cases and testing correctness of the output by comparing it to the expected output. Additionally, time and memory limits are usually enforced during this evaluation process, which allows obtaining an assessment not only in terms of the correctness of the solutions, but also in terms of their time and space complexity [8]. Thus, the efficiency of the used algorithms is also taken into consideration.

The same or slightly similar method can be used as a solution to the previously mentioned problem of fast program codes assessment in the educational environment. There are many existing systems that are used for this purpose [1], and the benefits are numerous, as described in [12].

Automated assessment using test cases can also provide an environment in which a contestant (student) can receive feedback during the contest (training, exam, etc.). The contestant can be informed if his/her solution passes the given tests, or on how many of the test cases it produces a correct result.

At our institution, we have developed an automated assessment system that is currently being used in some courses, called MENDO [5]. It employs the previously described type of grading and represents a modern online contest management system. It has also been successfully used for organization of the Macedonian national competitions in informatics for the last 4 years [3]. Similar types of automated grading systems are used at the International Olympiad in Informatics [2], [7].

The automated grading of the programming solutions is quite rough and strict. The grade (usually expressed in terms of number of gained points) assigned to a particular program may give a completely wrong impression about how good (and efficient) is the algorithm that it implements. As an illustration, one possible situation where this type of automatic assessment would assign zero points is with a program that, in essence, represents an implementation of a complete and 100% correct algorithm for solving the problem at hand, but uses a wrong format when printing the output data.

The question that we address is: is there an automated way to determine similarity of a code (the one that scored low or zero points on the grading system) to another code (that scored full score), in order to reconsider the grading of the first one? Our idea is to use data mining techniques to construct a model that will lead to better and more equitable evaluation of programming codes in the future.

3. Related Work on Source Code Analysis

The submissions of the students (solutions to practice or exam exercises) for a given university or high school programming course, or the submissions of the participants at a particular programming competition, can be viewed as pools of program source codes, where each pool contains source codes that represent solutions to the same algorithmic problem. In order to be able to apply data mining methods for the problem under consideration, we have to extract some knowledge from the input data (source codes).

Source code analysis is the process of extracting knowledge about a program from its own source code. This paradigm has many applications into

a variety of software engineering tasks, including: clone detection, debugging, source code optimization, source code comparison, reverse engineering, performance analysis, and many others. There are many existing software tools that are used for source code analysis. Most of them have been designed to make the analysis for the major purpose of discovering software plagiarism.

Commonly, the first step of a typical source code analysis is a construction of an abstract model of the program that is not programming language specific. In many cases, the first model that is created is the abstract syntax tree (AST) – a tree where each node is a construct in the source code. AST is usually used as a base for creating more complex graph structures (models) representing various aspects of the source code. Therefore, different models are used by different source code analysis algorithms [4].

According to Roy and Cordy [9], source code similarity detection algorithms can be classified as algorithms based on:

- **Strings**: this approach involves searching for exact textual matches, for example five-word runs. This is a fast procedure, but has the disadvantage that it can be easily disturbed (e.g., by renaming identifiers).
- **Tokens**: this is a similar approach to the one based on strings, but with the difference that it uses a lexer to convert the program into tokens first. Here, whitespaces, comments and identifier names are discarded, so the procedure is more robust to simple text replacements. Most academic plagiarism detection systems are based on this approach, using different algorithms to measure the similarity between token sequences.
- **Parse trees**: this approach includes building and comparing parse trees, which allows detection of higher-level similarities. For example, tree comparison can normalize conditional statements, and detect equivalent constructs as similar to each other.
- **Program dependency graphs (PDGs)**: in this approach, a structure called program dependency graph is built. This structure captures the actual flow of control in a program and allows much higher-level equivalences to be detected, but at greater expense in complexity and calculation time.
- **Metrics**: here, numerical parameters called metrics are recorded for each program. These parameters capture "scores" of source codes according to different criteria, like "number of for loops" or "number of if statements".
- **Hybrid approaches**: combinations of some of the previously mentioned approaches.

We have proposed a new hybrid approach for determining similarity between source codes that includes the use of data mining methods.

4. Description of Our Hybrid Approach for Source Code Similarity Detection and Results of Its Use

We presented our new hybrid approach for source code similarity detection in [11]. Here, we will give a short overview.

Given a set (pool) of source codes that represent solutions to the same algorithmic problem, we perform the following three main steps:

1. We build a parse tree for each of the source codes under consideration;
2. We extract attributes that represent key characteristics of the source codes by calculating metrics from the constructed parse trees. We obtain attribute representations that describe each source code's structure numerically;
3. We apply data mining clustering methods on the dataset formed by these attribute representations in order to discover the existence of similarities among them.

4.1. Determination of the attribute representation of the source codes

First step of our approach is to describe each program's source code by a vector of attribute values. We have done meticulous analysis, and proposed large number of attributes that can represent some of the source code features. We followed the principle that the selected attributes should: (1) be as abstract as possible; (2) accurately describe the original source code. In order to select the most descriptive vector attributes, we applied a greedy search process. More precisely, we conducted the experiments

outlined in the following section using the chosen set and processed the obtained results. In each of the following iterations, we tentatively removed a single attribute in the current attribute set and did the experiments again, this time using the resulting set. Each evaluation of the current attribute set produced a numeric measure (average percentage of correctly clustered vectors) of the expected performance of that set. We quantified the effect of removing each attribute in turn by this measure; we chose the best candidate for removing, and continued with the next iteration. The entire process was repeated until no attribute produced an improvement when removed from the current attribute set.

The final attribute set contains 62 attributes and is presented in Table 1. The attributes are grouped together according to the aspect of source code's structure they describe, and a short description of each respective group is given.

4.2. The EMAx tool

We've created a software tool called EMAx for the purpose of conducting the first two steps of the proposed procedure for source code similarity detection (building parse trees and attribute extraction) [10].

EMAx has been designed to be used as a tool that provides vector representations of source codes further used in solving the problem of source code comparison. The tool has been tested with real sets of source codes taken from programming competitions and has proved as

Attributes	**Description of attribute values**
"for", "ranged for", "while"	Total number of loops (for, while)
"if", "switch", "case"	Total number of conditional statements (if, switch)
"not nested for", "nested for 2", . . . , "not nested while", "nested while 2",. . .	Number of nested loops on different levels (for loops nested only inside for loops, while loops nested only inside while loops)
"not nested if", "nested if 2",. . .	Number of nested conditional statements on different levels (if statements nested only inside if statements)
"nested for|while 2", "nested for|while 3",. . .	Number of mixed nested loops on different levels (while loops nested inside for loops, for loops nested inside while loops)
"1 if inside 1 for/while", "2 if inside 1 for/while", . . . , "1 if inside 2 for/while", "2 if inside 2 for/while",. . .	Number of conditional statements nested inside loops on different levels (if statements nested inside for or while loops)
"1 for/while inside 1 if", "2 for/while inside 1 if", . . . , "1 for|while inside 2 if", "2 for|while inside 2 if",. . .	Number of loops nested inside conditional statements on different levels (for or while loops nested inside if statements)
"function definitions", "recursive functions", "functions with try block"	Modularity of the program (number of modules, i.e. functions)
"variables", "unused variables", "1 dim. arrays", "2 dim. arrays",. . .	Number of different variable identifiers
"classes not derived", "classes derived from 1 base class", "classes der. from 2 base. . .	Number of classes
"namespace definition"	Number of sub-spaces in which the space of visibility of the program entities is partitioned
"break", "return", "continue"	Total number of jump statements
"new expression", "cast expression", "using directives"	Number of other useful expressions

Table 1. Description of attributes used in the source code representation.

very efficient in performing the task for which it has been intended. One of the main advantages that the tool offers is that it analyzes the source codes by simulating each code's execution from a given starting point.

The tool takes a set of C/C++ source codes as input, and generates a ".CSV" file containing the attribute representations of all the source codes in that set, one per line. In accordance with the standard for this file format, the first line of the generated file contains the attribute names, and all the data are separated by commas. This format of the output file was selected to be further processed with the WEKA software [6].

4.3. Description of the conducted experiments

We selected 8 different pools (sets) of program source codes from the MENDO system. Each pool contained solutions to a particular programming task that were submitted on MENDO by the participants at one of the Macedonian national competitions in informatics.

All the programs were written in the C++ programming language. Only executable programs (that compiled without errors) were included in the pools.

The selected pools contained 676, 449, 171, 141, 273, 231, 94 and 365 source codes, respectively. In the next step, using EMAx, we obtained the corresponding files that describe these pools.

Given that our goal was to determine similarity between source codes belonging to the same pool, we concluded that the most appropriate data mining approach would be to seek for groups of vectors (descriptors of the source codes) that belong together, i.e. to use clustering methods.

Intuitively, similar source codes should be grouped in the same cluster, so if we manage to obtain clusters that contain source codes from a single pool, then this will imply that we have detected similarity between each pool's source codes. Of course, it is expected that each pool may contain different types of solutions that represent implementations of different algorithms, but if each such solution type has many representatives (programs) in the pool, then all of them will be grouped in the same cluster.

In order to conduct the clustering, we formed all possible combinations of 2, 3, 6, 7 and 8 source code pools, and saved the respective attribute representation of each combination in a separate file. We obtained, respectively, 28, 56, 28, 8 and 1, files. Then, we applied two appropriate clustering methods (the standard K-Means and EM algorithms). These methods receive the number of clusters to be sought as input (for our problem, this parameter was known in advance), to produce the required clusters. Finally, we evaluated the resulting clusters using the "Classes to clusters" evaluation technique, described in [13]. For the purpose of conducting this evaluation, we added a single attribute to the source code representation. Its value represents the ordinal number of the actual pool to which the respective source code belongs. This attribute played the role of "class attribute" in the evaluation process.

4.4. Results and analysis

The results from our experiments, presented in [11], are given in an extended form in Table 2 and Table 3. Table 2 gives the average, the best and the worst percentage of correctly clustered instances using the K-Means method, for combinations of different number of pools. Furthermore, the last column of the table shows the average area under the ROC curve (AUC) for the clusters found by this method.

Number of combined source code pools	Average % of correctly clustered instances	Best % of correctly clustered instances	Worst % of correctly clustered instances	Average AUC (Area under the ROC curve)
2	88.94%	98.38%	71.65%	0.848
3	79.32%	97.23%	59.56%	0.813
6	57.76%	69.81%	47.45%	0.724
7	55.30%	60.26%	48.30%	0.724
8	55.09%	55.09%	55.09%	0.731

Table 2. Results from the "Classes to clusters" evaluation for the clusters found by the K-Means method.

The results show that we have achieved over 88% average of correctly clustered instances,

and even over 98% of correctly clustered instances for some combination of 2 pools.

One has to keep in mind that we are talking about vector representations of solutions for a given task that are not all correct. Furthermore, there is a possibility that many different approaches (algorithms) are employed in different solutions to the same problem. Hence, 88.94% average of correctly clustered instances and an average of 0.848 AUC value are very good and promising results.

The same statistics for the EM method are shown in Table 3. The estimated performance of K-Means doesn't differ significantly from the one of EM, and this claim remained true when different numbers of pools were combined, i.e. different numbers of clusters were required. Figure 1 confirms this graphically, by showing the curves of average correctly clustered instances (source codes) for different numbers of clusters.

Number of combined source code pools	Average % of correctly clustered instances	Best % of correctly clustered instances	Worst % of correctly clustered instances	Average AUC (Area under the ROC curve)
2	88.24%	98.92%	67.00%	0.854
3	74.65%	96.08%	47.54%	0.791
6	58.08%	67.51%	47.90%	0.732
7	55.16%	58.60%	51.28%	0.714
8	51.71%	51.71%	51.71%	0.703

Table 3. Results from the "Classes to clusters" evaluation for the clusters found by the EM method.

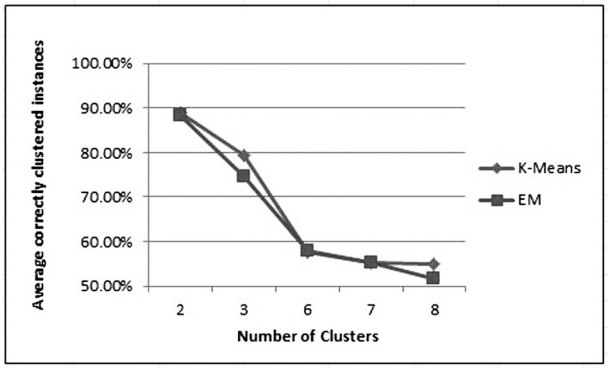

Figure 1. The estimated performance of K-Means and EM expressed in terms of average percentage of correctly clustered source codes for different numbers of combined pools (clusters) [11].

Additionally, here we present information on the average precision and recall values for the clusters found both by K-Means and EM (Table 4). These data confirm our conclusions.

Figure 1 also shows that the percentage of correctly clustered instances declines, as the number of combined different pools rises. Nevertheless, this declination is rather slower than if we consider the probability of random guess of the class, when there are 3, 6, 7 or 8 classes.

Number of combined source code pools	Average precision (K-Means)	Average precision (EM)	Average recall (K-Means)	Average recall (EM)
2	90.11%	88.48%	88.94%	88.24%
3	80.98%	75.65%	79.39%	75.17%
6	60.06%	56.21%	58.66%	59.27%
7	57.80%	49.65%	56.21%	55.69%
8	60.80%	50.30%	56.50%	51.40%

Table 4. Average precision and recall values for the clusters found by the K-Means and the EM method.

5. Description of a New Model for Semiautomatic Student Source Code Assessment

Based on the results from the previously described series of experiments on our approach for code similarity detection, we have concluded that clustering methods give quite good results in detecting similarity between source codes. This has encouraged us to consider the possibility of applying these methods in seeking an answer to our main research question – determining similarity of a given code (that scored low or zero points on a grading system) to another code (that scored full score), in order to reconsider the grading of the first one.

Let's assume we are given a pool of source codes, for which we know the grades assigned by an automated grading system. More precisely, for each of the programs in the pool, we know whether it passes all the test cases or fails on at least one of them. In order to decide which codes in the pool under consideration are potentially good solutions (implementations of appropriate algorithms) to the problem for which they have been written, we propose the following algorithm:

1. Generate the attribute representation of the pool;
2. Apply data mining clustering methods on the representation in order to form appropriate clusters of similar source codes within the pool;
3. For each of the obtained clusters:

- If the cluster doesn't contain programs that pass all the test cases, then reject it – none of the codes that belong to that cluster are candidates for reassessment;
- If the cluster does contain programs that pass all the test cases, then all the other programs (that fail on at least one of the test cases) in the cluster are candidates for reassessment.

With this algorithm we define a model that involves a joint effort of an automated system and a human, in order to produce a better assessment of the students' work.

The model attempts to offer a quality improvement of the results from the automated assessment. The proposed solution is obviously slower than the fully automated approach, but can be used in situations where automated assessment is considered to be very strict (e.g. students are beginners and/or very young, so it is necessary to give them better motivation with some appropriate partial points).

We evaluate the proposed algorithm with the experiments described in the following section.

6. Evaluation of the Proposed Algorithm

In this section, we describe the experiments that we have conducted to test the successfulness of the proposed algorithm, as well as the obtained results.

6.1. Experiment description

For this series of experiments, we selected 3 different pools of source codes from the MENDO system. Each pool contained solutions to a concrete programming task, submitted on MENDO by first-year students as part of an introductory programming course exam held at our institution. The selected pools contained 96, 70 and 33 source codes, respectively. All the programs were written in the C programming language. Like in the previous series of experiments, only executable programs were included in the pools.

First, an automated assessment of the selected pools was performed using MENDO. All the source codes that were not assigned the maximum points by the system were subsequently assessed manually by the teaching assistants, as part of the course exam. All of the assistants in question had previously graded more than 1000 programming codes (in their teaching experience). This provided us with an objective grade assigned by a professional – data that served as a way to check the successfulness of the proposed algorithm.

In the next step, we generated the corresponding files that describe the pools using EMAx, and we applied the same two clustering methods as in the first series of experiments. It should be noted that for this series of experiments the required number of clusters was not known in advance, so a multiple application of the clustering methods was necessary in order to determine the most appropriate partition for each of the pools. We experimented by seeking 2, 7 and 10 clusters with K-Means, and by seeking 2 and 10 clusters with EM. Also, for each of the pools we ran an additional experiment in which we didn't supply the number of clusters as parameter to EM – thus letting the method to determine the appropriate number of clusters on its own.

6.2. Results and analysis

The results obtained by applying the algorithm for determining candidates for reassessment on each of the 3 pools of source codes are presented in Table 5, Table 6, and Table 7, respectively. Here we must emphasize that we use the expression "incorrectly clustered instances" to mark those instances that have been assigned to a cluster which doesn't contain programs that pass all the test cases, and have received a solid grade when assessed manually by the teaching assistants (at least 40% of the total possible number of points).

Table 5 shows the results for the first pool. The best results were obtained when the number of required clusters was 10, regardless of which of the two clustering methods was applied.

	K-Means			EM		
No. of clusters	2	7	10	–	2	10
PCI	27	12	10	21	21	17
ICI50%	2	3	4	3	3	3
ICI40%	5	8	9	7	7	8

Table 5. Results from applying the proposed algorithm on the first pool of programs (96 codes, 25 of them pass all the test cases). PCI – number of instances assigned to clusters which contain codes that pass all the test cases (excluding those that pass all the test cases); ICI50% – number of incorrectly clustered instances that have received more than 50% of the possible points when assessed manually; ICI40% – number of incorrectly clustered instances that have received more than 40% of the possible points when assessed manually.

The clusters formed using K-Means suggest only 10 programs as candidates for reassessment, while the number of candidates suggested by EM is 17. This means that, for the first case, the amount of programming solutions that should be reassessed manually has been reduced to 14% using the proposed algorithm for determining candidates (in the worst scenario, candidates for manual assessment would be all the codes that fail on at least one of the test cases). For the second case, using EM, the number of candidates for reassessment has been reduced to 24%. The benefits for the teachers in both cases, in terms of the time saved, are extremely large.

The best results for the other two pools were also achieved when 10 clusters were required using any of the two clustering methods, as can be seen from Table 6 and Table 7.

	K-Means			EM		
No. of clusters	2	7	10	–	2	10
PCI	33	23	22	27	43	24
ICI50%	2	2	2	2	0	2
ICI40%	2	2	2	2	0	2

Table 6. Results from applying the proposed algorithm on the second pool of programs (70 codes, 27 of them pass all the test cases).

The benefit for the human evaluators of the second pool of programming solutions is that they have to manually assess 51% (according to K-Means) or 56% (according to EM) of the solutions that they would have to assess without using the algorithm for determining candidates. These percentages for the third pool are both equal to 38%.

It is interesting to note that the number of clusters found by EM (when applied without supplying this number as its parameter) was 5, 3 and 4, respectively, for the first, second and third pools. However, the obtained results (in terms of the ratio PCI versus ICI) in this case were worse as compared to the case when the number of clusters supplied to EM was 10, for each of the pools.

	K-Means			EM		
No. of clusters	2	7	10	–	2	10
PCI	23	14	11	20	29	11
ICI50%	1	3	3	3	0	1
ICI40%	2	5	5	4	0	3

Table 7. Results from applying the proposed algorithm on the third pool of programs (33 codes, 4 of them pass all the test cases).

The results from the conducted experiments confirm that data mining clustering methods can be used for estimating the "potential" of a given source code – particularly, one that has been assigned a weak grade by an automated grading system. Namely, even if the code does not pass on some or all of the test cases, the methods can predict whether it potentially represents implementation of a good or approximately good algorithm for the appropriate problem. In other words, these methods can very accurately predict if the program solution under consideration would receive more than 50% of the total possible number of points when assessed manually by a professional human evaluator.

7. Conclusion

In this paper we explained the need for fast assessment of program source codes that occurs in competitions in informatics and also in university and high school level programming courses. Given that automated assessment can result in very rough and strict grades, we proposed a different model that can improve the quality of grading. The solution involves data mining methods.

We presented the results of experiments on combined sets of solutions to different programming tasks, with the intention to prove that data mining methods can give good results in detection of code similarity. Based on the results achieved in correctly clustering the different task solutions, we presented a new model for semiautomatic student source code assessment, with human – computer collaboration. The presented results from the preliminary testing of the model are very promising, and we can conclude that the model can be a strong base for a new approach to the assessment of student programming codes.

Future work that should be done is to improve the model in the sense of building a complete environment with a user friendly interface that will allow easy employment of the model.

Acknowledgments

The research presented in this paper is partly supported by the Faculty of Computer Science and Engineering, at Ss. Cyril and Methodius University in Skopje.

References

[1] P. IHANTOLA, T. AHONIEMI, V. KARAVIRTA, O. SEPPALA, Review of Recent Systems for Automatic Assessment of Programing Assignments. In: *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*. New York, NY: ACM, 2010.

[2] INTERNATIONAL OLYMPIAD IN INFORMATICS. `http://www.ioinformatics.org`

[3] M. JOVANOV, B. KOSTADINOV, E. STANKOV, M. MIHOVA, M. GUSEV, State Competitions in Informatics and the Supporting Online Learning and Contest Management System with Collaboration and Personalization Features MENDO. *Olympiads in Informatics, an International Journal*, **7** (2013), 42–54.

[4] R. KIRKOV, G. AGRE, Source Code Analysis – An Overview. *Cybernetics and Information Technologies*, **10**(2) (2010), 60–77.

[5] B. KOSTADINOV, M. JOVANOV, E. STANKOV, A new design of a system for contest management and grading in informatics competitions. In: *ICT Innovations 2010, Web proceedings*, pp. 87–96.

[6] Machine Learning Group at the University of Waikato, New Zealand, WEKA 3: Data Mining Software in Java, (2012).

[7] S. MAGGIOLO, G. MASCELLANI, Introducing CMS: A Contest Management System. *Olympiads in Informatics, an International Journal*, **6** (2012), 86–99.

[8] M. MARES, Perspectives on Grading Systems. *Olympiads in Informatics, an International Journal*, **1** (2007), 124–130.

[9] C. ROY, J. CORDY, *A Survey on Software Clone Detection Research*. School of Computing, Queen's University, Canada, 2007.

[10] E. STANKOV, M. JOVANOV, A. BOJCHEVSKI, A. MADEVSKA BOGDANOVA, EMAx: Software for C++ Source Code Analysis. *Olympiads in Informatics, an International Journal*, **7** (2013), 123–131.

[11] E. STANKOV, M. JOVANOV, A. MADEVSKA BOGDANOVA, Source Code Similarity Detection by Using Data Mining Methods. In: *Proceedings of the 35th International Conference on Information Technology Interfaces (ITI 2013)*, pp. 257–262. IEEE, 2013.

[12] D. TRUSSO HALEY, P. THOMAS, A. DE ROECK, M. PETRE, Seeing the Whole Picture: Evaluating Automated Assessment Systems. *ITALICS e-journal of the Learning and Teaching Subject Network for Information and Computer Science (LTSN-ICS)*, **6**(4) (2007), 203–224.

[13] I. WITTEN, E. FRANK, M. HALL, *Data Mining: Practical Machine Learning Tools and Techniques*. 3rd Edition. Morgan Kaufman Publishers, 2011.

Received: August, 2013
Revised: September, 2013
Accepted: September, 2013

Contact addresses:

Emil Stankov
Faculty of Computer Science and Engineering
University Ss. Cyril and Methodius
st. Rugjer Boshkovikj 16
Skopje
Macedonia
e-mail: `emil.stankov@finki.ukim.mk`

Mile Jovanov
Faculty of Computer Science and Engineering
University Ss. Cyril and Methodius
st. Rugjer Boshkovikj 16
Skopje
Macedonia
e-mail: `mile.jovanov@finki.ukim.mk`

Ana Madevska Bogdanova
Faculty of Computer Science and Engineering
University Ss. Cyril and Methodius
st. Rugjer Boshkovikj 16
Skopje
Macedonia
e-mail: `ana.madevska.bogdanova@finki.ukim.mk`

Marjan Gusev
Faculty of Computer Science and Engineering
University Ss. Cyril and Methodius
st. Rugjer Boshkovikj 16
Skopje
Macedonia
e-mail: `marjan.gusev@finki.ukim.mk`

EMIL STANKOV is a PhD student at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. He is a member of the Executive Board of the Computer Society of Macedonia and has actively participated in the organization and realization of Macedonian national competitions and Olympiads in Informatics since 2009. His research interests are in the field of intelligent systems, machine learning and the application of CAA techniques in teaching programming as well as in programming competitions. He has participated in more than ten domestic and international conferences with paper presentations, and in several workshops and schools.

MILE JOVANOV is a teaching and research assistant at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. He is President of the Computer Society of Macedonia and has actively participated in the organization and realization of Macedonian national competitions and Olympiads in Informatics since 2001. He has finished his PhD thesis on online collaborative ontology building in e-learning environment in 2013. He is the author of over thirty scientific papers in the area of computer science published in the reviewed conference proceedings and international scientific journals. His research interests are in the fields of semantic web, algorithms and programming, digital presentation of national heritage, e-learning and ICT in education.

ANA MADEVSKA BOGDANOVA is an associate professor at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. She is a member of the Computer Society of Macedonia and has participated in the organization and realization of Macedonian national competitions and Olympiads in Informatics. Her research interests are in the fields of intelligent systems, bioinformatics, machine learning and ICT in education. She is the author of over forty scientific papers, chapters in the books on computer science and presentations for popularization of informatics.

MARJAN GUSEV is a professor at the Faculty of Computer Science and Engineering, University "Ss. Cyril and Methodius", in Skopje. His research interests are in the fields of parallel processing, computer networks, Internet technologies, e-business, mobile and wireless applications. He has authored/co-authored 18 books, 51 publications in reviewed international journals and 91 articles in reviewed international conference proceedings. He has participated in more than 86 conferences, workshops and seminars.