A NEW WEB CRAWLER FOR GATHERING SOURCE CODE SOLUTIONS FOR DATA MINING PURPOSES

Emil Stankov
Faculty of Computer Science and Engineering
Skopje, Macedonia

Mile Jovanov
Faculty of Computer Science and Engineering
Skopje, Macedonia

Vasja Pavlov
student at
Faculty of Computer Science and Engineering
Skopje, Macedonia

Ana Madevska Bogdanova
Faculty of Computer Science and Engineering
Skopje, Macedonia

ABSTRACT

Teaching programming in the modern educational environment imposes the need for fast assessment of large sets of programming solutions. The same applies to programming competitions, especially the international ones. Although the problem is typically solved by use of computer aided or automated assessment (CAA) systems that perform grading based on test cases, this dominant approach for source code assessment has its own serious drawbacks.

In this paper we present our research in this field, with focus on one important preparatory step before conducting our experiments. We review our proposed model for semiautomatic student source code assessment, and we explain the need for gathering a variety of source code sets. Then, we discuss the problem we have encountered when trying to collect source code sets from a very famous web-based CAA system. Finally, we describe the creation process for a web crawler – as a proposed solution to this particular problem.

I. INTRODUCTION

Teaching programming in the modern educational environment imposes the need for fast assessment of large sets of programming solutions. University and high school programming courses (especially the ones on introductory level) often include lots of exercises in order to make the adoption of the syntax of the programming language that is being taught easier for the students, and also to help them develop algorithmic way of thinking. Programming is a compulsory course in every computer science educational curriculum, so usually lots of students enrol in these courses. This introduces 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.

The need for fast assessment has been recognized even earlier, in the organization of competitions in informatics (programming). The International programming competitions typically require from the participants to submit program codes – solutions to concrete algorithmic problems. However, the competition difficulty is not so much in the programming part, as it is in the design of appropriate algorithms for solving the problems at hand. In most cases, these competitions are based on CAA of the submitted solutions, which is accomplished by running them on batches of input test data and testing correctness of the output by comparing it to the expected output. 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.

The same or slightly similar methods can be used as a solution to the previously mentioned problem of fast assessment of program codes in educational environment. There are many existing systems that are used for this purpose [1], and the benefits are numerous, as described in [2].

The grading of the programming solutions outlined above is quite rough and strict. The grade (usually expressed in terms of number of gained points) assigned to a particular program code 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 too strictly 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 rest of the paper is organized as follows: in Section II we review our new model for semiautomatic student source code assessment, and explain the need for gathering large source code sets. In Section III we present three different web-based CAA systems that gather source codes from students and/or contest participants. One of these systems, Topcoder [13], doesn't allow a direct access to its collected source codes. In Section IV we elaborate on the implementation of a new web crawler for gathering source codes, which has been created for the purpose of collecting source codes from the Topcoder's web site. Finally, in Section V, we give a conclusion and directions for future work.

II. THE NEW MODEL FOR SEMIAUTOMATIC STUDENT SOURCE CODE ASSESSMENT

The main goal of our wider research is to allow improvement of the assessment of programming codes submitted by students as a solution to a particular problem. The idea is to examine the possibility for detection of programs whose structure is similar to that of the correct programming solutions to a given algorithmic task, but have received a weak grade when assessed automatically by a CAA system. The detection of such programs would indicate the potential that they have, as well as the possibility that these programs might have been assessed too strictly by the system. In this way, the detected programs would be isolated as candidates for reassessment using some other strategy. Thus, this approach would significantly improve the quality of grading.

A. Source Code Similarity Detection

In order to determine similarity between source codes, it is necessary to conduct source code analysis. Source code analysis is an important field in computer science. Source code analysis is the process of extracting information about a program, from its own source code. It 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.

According to Roy and Cordy [3], source code similarity detection algorithms can be classified as based on: strings; tokens; parse trees; program dependency graphs (PDGs); metrics; and hybrid approaches.

B. The New Approach for Source Code Similarity Detection

We have proposed a new hybrid approach for determining similarity between source codes that includes the use of data mining methods [4]. 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.

C. EMAx – a Source Code Analysis Tool

For the purpose of conducting the described structural source code analysis, we have created a software tool called EMAx [5]. 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 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.

D. The Resulting Model

The results presented in [4] and [6] show that the proposed approach can be used for source code similarity detection with satisfactory precision. Based on this conclusion, we have also proposed a new model which offers a way, given a large set of solutions, to determine which of the solutions that are still not graded (or that have been assigned a weak grade) should be (re)assessed manually, using information from solutions that have been assigned a solid grade (by a human evaluator or a CAA system). The model represents collaboration between a human and software based on data mining clustering methods.

According to the proposed algorithm for detecting candidates for reassessment, first we group the programs into clusters containing similar programs (using data mining methods). Clusters that do not contain programs that are assigned the maximum score by the CAA system are rejected – we assume that these clusters do not contain candidates for reassessment. The programs from the remaining clusters that are not graded with the maximum score should be subject to reassessment. Our first experiments showed decrease of the candidates for reassessment from 50% up to 80%, which effectively shortens the time for the task of reassessment.

E. The Need for Source Code Sets

In order to evaluate the successfulness and the usefulness of the proposed algorithm for detecting candidate source codes for reassessment, we have conducted experiments with some sets of programming codes (results presented in [6]). For deeper insight in the problem, we needed to experiment with many different sets of source codes, i.e. with a variety of program sets – solutions to different algorithmic problems. An inevitable step before starting the experiments was to collect appropriate source code sets. The focus of the following two sections of the paper is exactly on this important issue.

III. Systems Containing Source Code Sets

In this section we present three contest systems / web programming environments that gather programming codes from contestants / students. Each of these systems is used for different purposes, and by different sets of users.

A. MENDO

MENDO [7, 8] is a contest management system developed to support the organization of the Macedonian national competitions in informatics, organized by the Computer Society of Macedonia [9]. Because it contains an automated assessment feature, this system is currently used in some courses at our institution. MENDO employs the previously described type of grading and represents an example of a modern online contest management system. It has been successfully used for organization of the Macedonian national competitions in informatics for the last 4 years. Similar types of automated grading systems (for example, [10]) are used at the International Olympiad in Informatics [11].

Since we participate in the development of this system, we had an easy access to all the source codes submitted by students / participants and gathering them as sets of solutions to different tasks represented a fairly simple task.

B. E-Lab

E-Lab [12] is a CAA system developed at our institution as a web programming environment. This system has been used for 2 years in the first-year introductory programming courses. It also offers grading based on test cases, but is not suitable for grading programming solutions in informatics competitions.

Being a property of our institution, E-Lab provided us with easy access to all the source codes submitted by first-year students, and like with MENDO, it was quite easy to gather them as sets of solutions to different tasks.

C. Topcoder

Topcoder [13] is a company that uses crowdsourcing (a form of outsourcing) to produce high quality software. Everything is conducted via competitions. People compete, submit a solution and after the competition only the winners get paid. There are different types of competitions, ranging from bug tracking, design and development competitions, to architecture competitions, and finally the most popular ones – the algorithmic single round matches (SRM). Our research focuses only on SRMs.

SRMs are competitions held regularly 3 to 4 times a month. Each contest lasts 75 minutes. Participants are divided into two divisions, 1 and 2 (the former being the stronger and the later the weaker division), based upon a ranking obtained by competing in previous SRMs. Furthermore, the participants in each division are subdivided into so called "rooms" of up to 20 people. Each division is given 3 problems with increasing difficulty. After the coding phase of 75 minutes, each participant is given a preliminary score based on how fast he has solved the problems. Then, there is a challenge phase and a system testing phase. In the challenge phase, participants are given 15 minutes to review the solutions of all the problems submitted by the people in their room. If they think a solution is wrong (contains a bug), they may submit their own test case and test it against the author's solution. A correct "challenge" adds 50 points to the score and an incorrect one takes 25 away. Finally, the remaining solutions "still standing" after the challenge phase are tested against a test set, designed by a test crew. A solution which passes the system tests is generally considered correct. However, in very rare occasions, due to a wrong choice of test cases, a solution may pass all of them although it is not completely correct.

The Topcoder competitions take place in the "Topcoder Arena", which is a Java applet. After each competition, all the solutions that (at the very least) compiled are uploaded on the Topcoder website. Since we didn't have a direct access, we had to find a way to gather them from the website.

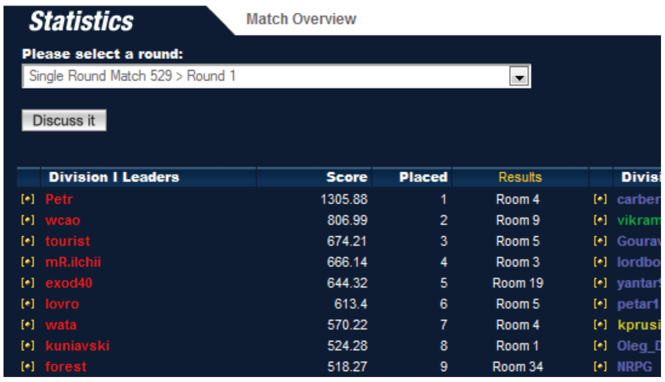
IV. REALIZATION OF WEB CRAWLER FOR GATHERING SOURCE CODE SETS

For the purposes of our research, we needed to build a web crawler in order to be able to extract codes from past Topcoder competitions. A web crawler is a program which scans a website and is able to go to its subsequent links much like a Windows user walks (crawls) through desktop folders. Our specific task was to pick a handful of different tasks (10 would suffice), but to pick only such tasks that had a large number of passing (correct) solutions so that we could build a large enough dataset of codes. For each problem, we needed to separate the solutions written in Java, C++ and C#. The project was built as 2 applications: one written in Java and another one in C++.

The Java application works as follows. It is provided a website URL as input by the C++ application. Then, it downloads the HTML of the given website and saves it into a text file. In addition, it uses cookies in order to log in, since Topcoder requires one to be logged in so that she can get access to the results.

The main part of the project is the C++ application. When run, it asks for an initial URL – the starting point for the crawler. This URL was handpicked for each particular problem. An example of an URL for one page from the Topcoder website is: "http://community.topcoder.com/stat?Contest=%2Fstat%3Fc%3Dround_overview%26er%3D5%26rd%3D14722&rd=14722&c=round_overview&er=1000".

The web page corresponding to this URL is shown in Fig. 1.



	Division I Leaders	Score	Placed	Results		Divisi
[•]	Petr	1305.88	1	Room 4	[•]	carber
[•]	wcao	806.99	2	Room 9	[•]	vikram
[•]	tourist	674.21	3	Room 5	[•]	Gourav
[•]	mR.ilchii	666.14	4	Room 3	[•]	lordbo
[•]	exod40	644.32	5	Room 19	[•]	yantar
[•]	lovro	613.4	6	Room 5	[•]	petar1
[•]	wata	570.22	7	Room 4	[•]	kprusi
[•]	kuniavski	524.28	8	Room 1	[•]	Oleg_D
[•]	forest	518.27	9	Room 34	[•]	NRPG

Figure 1: Web page from the Topcoder web site showing the results for a single SRM competition.

Fig. 1 presents a web page containing the results from a single SRM contest. The division 1 coders are shown on the left and the division 2 coders are on the right. The figure depicts only a small part of the listed contestants. Usually, there are 1000 more contestants below. The data are always presented in this format. By clicking on the yellow circle in brackets (placed before any name), we can see the results of the selected user for that competition. There, we are presented with links to that user's solutions, if they have compiled. Furthermore, there is a column which shows whether the particular solution passed or failed the system tests. For our research, we were only interested in the passing ones.

Before we started coding the web crawler, we had to go through the HTML of some of these pages manually and see the way data is formatted there so that we know how to build the crawler. Afterwards, for each of these pages we created a parser which isolates the URLs needed to continue crawling. For example, on the page presented in Fig. 1, the parser would isolate the URLs which represent the yellow dots beside each name and save them in a separate file. From here, the next parser needs only to repeat the procedure. Then we use the Java application to open each URL and save the HTML codes in text files. The C++ application would then parse each file and search for the URLs of the specific task we want. This step is conducted one more time, now saving the code of each solution in a separate file. The file extension is determined based on a basic code analysis. For example, all C++ codes typically begin with a #include pre-processor directive, so it is easy to determine that the extension for such files should be '.cpp'.

V. Conclusion and Future Work

In this paper we presented our wider research on improvement of the assessment of programming codes submitted by students as a solution to a particular problem. Further, we emphasized the need for using a web crawler and we discussed the creation process of such a crawler for the Topcoder system in details. The described web crawler has played a significant role in achieving and confirming the results of our research.

A future step in our research regarding the web crawler will be to create a more general crawler that will offer greater opportunities for gathering source codes from many different web systems/sites.

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*, ACM, New York, NY, 2010.

[2] D. Trusso Haley, P. Thomas, A. De Roeck, M. Petre, "Seeing the Whole Picture: Evaluating Automated Assessment Systems", in *ITALICS e-journal of the Learning and Teaching Subject Network for Information and Computer Science* (*LTSN-ICS*) 2007; 6(4): 203-24.

[3] C. Roy, J. Cordy, "A Survey on Software Clone Detection Research", School of Computing, Queen's University, Canada, 2007, http://research.cs.queensu.ca/TechReports/Reports/2007-541.pdf.

[4] 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, University Computing Centre, 2013.

[5] E. Stankov, M. Jovanov, A. Bojchevski, A. Madevska Bogdanova, "EMAx: Software for C++ Source Code Analysis", *Olympiads in Informatics, an International Journal*, 2013, vol. 7, pp. 123-131.

[6] E. Stankov, M. Jovanov, A. Madevska Bogdanova, M. Gusev, "A New Model for Semiautomatic Student Source Code Assessment", *Journal of Computing and Information Technology (CIT),* 2013, (in print)

[7] 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, Ohrid, Macedonia, 2010.

[8] The MENDO system, http://mendo.mk.

[9] Computer Society of Macedonia, http://www.cs.org.mk.

[10] S. Maggiolo, G. Mascellani, "Introducing CMS: A Contest Management System", *Olympiads in Informatics, an International Journal*, 2012, vol. 6, pp. 86-99.

[11] International Olympiad in Informatics, http://www.ioinformatics.org

[12] T. Delev, D. Gjorgjevikj, "E-Lab: Web Based System for Automatic Assessment of Programming Problems", in *ICT Innovations 2012, Web proceedings*, pp. 75-83, Ohrid, Macedonia, 2012.

[13] The Topcoder system, http://topcoder.com.