Mining an Online Judge System to Support Introductory Computer Programming Teaching

Rodrigo Elias Francisco
Instituto Federal Goiano
Campus Morrinhos
Morrinhos – GO – Brazil
+55 (64) 34137900
rodrigo.francisco@ifgoiano.edu.br

Ana Paula Ambrosio
Instituto de Informatica
Universidade Federal de Goiás
Goiânia – GO – Brazil
+55 (62) 35211181
apaula@inf.ufg.br

ABSTRACT

Computer programming is an activity which requires a set of cognitive processes that naturally develop through practice, writing algorithmic solutions. Students learn a lot from their mistakes, but for this they need feedback on their workouts. Marking students' work outs is very time consuming, which often limits a teacher's capacity to offer close guidance individually. The PROBOCA project aims to build a tool, based on the BOCA online judge, suited for the purpose of learning computer programming by practice. In addition to a problem database organized by theme and difficulty, the system provides functionalities to support the teacher in the classroom. One of the main endeavors is to develop a procedure for estimating the degree of difficulty of a certain problem. This "nominal" parameter may then be compared to the difficulty level as perceived by each student. The result is a valuable indicator of those students that are experiencing challenges. This paper presents the preliminary specification of PROBOCA´s architecture and functional requirements along with its current state of development.

Keywords

Online Judge; Computer Programming Education.

1. INTRODUCTION

The learning process of Computer Programming (CP) usually involves practicing the resolution to many problems. Students write code to implement algorithms that meet exercise requirements and teachers should review those codes and present their feedback to the students. As this is a time consuming task, some teachers are installing and using online judge systems as automatic reviewing and scoring tools.

Online judge refers to software tools originally designed for programming competitions. They usually run on a server, and contestants access it online. Basically, its role is to present the contestant teams with a list of problems, to which they should respond by uploading program codes that satisfy the criteria of each problem. The tool then evaluates the answer code by using a set of predefined inputs and comparing the program results to predefined outputs. If the output of the answer code exactly matches the expected output for the corresponding input, the answer code is considered correct. Otherwise it is considered incorrect. No indication of where the program went wrong is given. Although helpful as a teaching assistant, these tools were not designed for use in a classroom and therefore lack some features that are important for academic management.

The BOCA software [1] is an online judge system used in programming marathons in Brazil. It is freely available for institutions to download and install. This particular system allows teachers to register problems and to track their students' work. However this system is neither easy to handle nor has an exercise database (DB), needed to facilitate the generation of problem lists.

This project proposes to extend the BOCA online judge to make it more suitable for use in introductory programming teaching. The resulting system, called PROBOCA, complements the online judge functionality with features that improve its ease-of-use, enhancing teacher productivity and course effectiveness.

One of the introduced features provides automatic classification of problem difficulty, based on submitted solutions and students' evaluation of degree of difficulty encountered in solving a given problem. This will aid teachers while composing the exercise lists, allowing them to better gauge the list complexity. It also lets students organize the order of problems to solve, tackling the easier problems first before turning to more complex ones.

Another additional feature is the production of student reports based on the submitted solutions and the students' behavior while accessing the tool. Student evaluation of program difficulty, number of submissions for the same exercise, time between submissions, order of submissions, among other information gathered by the tool, reveal information about student behavior and his ease in solving the proposed problems.

2. EXERCISES ON PROGRAMMING EDUCATION

Aiming at automatic correction of program code and student monitoring and evaluation during the programming process, several initiatives have been developed.

Within this line of research, the study of Chaves et al [2] aims to explore resources of online judges looking to integrate them into the Moodle system. The goal is to provide a Virtual Learning Environment (VLE) with automatic evaluation feature, allowing the teacher to monitor students´ problem solving. The authors defined an architecture containing a module for integration with online judges (MOJO). The Integration Module integrated the VLE structure to URI Online Judge [3] and SPOJ Brazil [4].

In [5], the authors have developed a prototype intended to be an educational online judge. They argue that online judge tools are suited to competition and have few educational features. They also criticize the way online judges provide feedback to students only indicating whether the answer is right/wrong or if there were errors at compilation/runtime. Their project, called JOnline, aims to add the following didactic features: Presenting tips in

Portuguese to fix compilation errors found in the submitted source code; presenting the test cases that generate erroneous results; organization of the problems by topic; difficulty level deduced from a poll conducted by the system and a resource for collaborative programming that allows two students to co-write one common code.

Automatic generation of exercise lists based on user defined criteria requires the classification of problems according to these criteria. In [6], the authors conclude that there is a strong relationship between the difficulty of a problem and the total number of lines of code and amount of flow control in the program (IF, WHILE, FOR ...). New students have difficulty in reading and interpreting problem statements. This problem has been related to the difficulty in dealing with abstraction [9].

3. BOCA

Designed for use in programming marathons, the BOCA online judge system has vocabulary and requirements contextualized by Competition and Teams. The main interest in this system is that it was designed to enable its users, the competitors, to interact with a set of problems. Figure 1 shows a use case diagram with the different functions provided by BOCA. Its software allows registration of problems for a given competition. That is done by submitting a PDF file stating the problem to be tackled and the files containing the input and output data sets for the corresponding test cases. The registered problems are associated with a single competition. If needed, to reuse the problems for another competition, the files must be inserted again. BOCA does not include a database to store the problems.

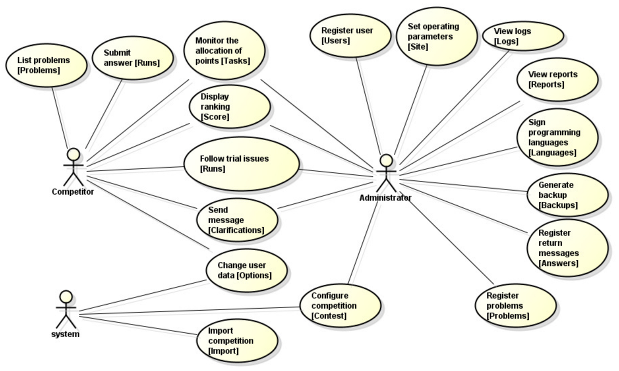

List problems [Problems]
Submit answer [Runs]
Monitor the allocation of points [Tasks]
Register user [Users]
Set operating parameters [Site]
View logs [Logs]
View reports [Reports]
Display ranking [Score]
Sign programming languages [Languages]
Follow trial issues [Runs]
Competitor
Administrator
Generate backup [Backups]
Send message [Clarifications]
Register return messages [Answers]
Change user data [Options]
Configure competition [Contest]
Register problems [Problems]
system
Import competition [Import]

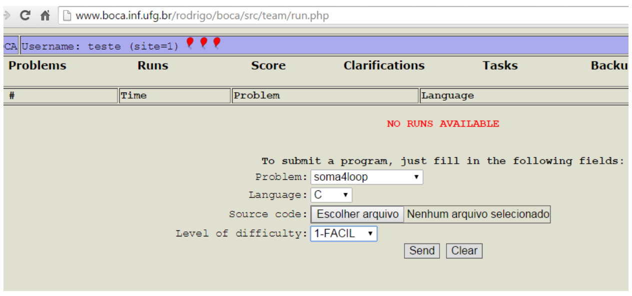
Figure 1. BOCA´s use case diagram

In the system, colored balloons represent the points gained by the competitors. This feature is part of the programming marathons context where correctly answering a problem yields a balloon to its team. Besides allowing teams to monitor their submissions and the related results, BOCA also provides a screen displaying the teams ranking including the team's overall score, and a list of the solved exercises using balloons to identify those successfully done. It also produces information on students' interaction with the problems containing the time spent and the number of resolution attempts.

BOCA has been used in CP introductory classes at our institute for some years, with good results. Using this system enhances the practice of problem solving by providing automatic feedback to students. We have observed that the number of problems solved by students using BOCA is significantly higher when compared to traditional exercise lists. The burden on the teacher is significantly reduced and the students get feedback. Teachers are then able to focus on helping the students that fail to solve the problems even after several attempts. This strategy allows students to tackle a larger number of exercises, thus increasing their potential to master cognitive skills required for programming. However, in spite of these advantages, some drawbacks were revealed.

Looking at the process of reviewing students' answers two issues were identified. First, the system assesses a submitted program by comparing its output, as generated by the student's code in response to a given input data set, to the registered expected output. For the two outcomes to be identical, exact formatting of the program's output is required. At the beginning students incurred in many errors just for using a slightly different format. This situation marks the problem's result as wrong whereas the program's logic maybe right, causing frustration among students as the system does not point out where the error is. Students, however, managed to adapt and began paying more attention to output formatting. Second, there is no cross-answers plagiarism identification, in other words no control over cheats. It was observed that some students simply copied their colleagues' program and submitted them as their own, thus considered to have solved the problem without any real comprehension of the solution.

Figure 2 shows the BOCA code submission screen. It was modified to include the "level of difficulty" selection where the students evaluate how hard it was to solve that problem they are submitting.



```
Username: teste (site=1)

#    Time    Problem    Language

NO RUNS AVAILABLE

To submit a program, just fill in the following fields:
           Problem: soma4loop
          Language: C
       Source code: Escolher arquivo  Nenhum arquivo selecionado
Level of difficulty: 1-FACIL
```

Figure 2. BOCA's submission functionality answers

Regarding the class management functionality, it was noted that each installed instance of BOCA supports one active competition at a time, meaning a single class per instance. Thus, if a teacher wants to have several competitions running at the same time i.e. different groups doing the same or different sets of exercises, he must install multiple instances of BOCA. Each instance is independent. So even if two competitions are composed of the same problems (e.g. for different classes), these problems must be separately registered for each instance. As each competition is independent, the system does not group the results of the different competitions. This feature would be interesting in case of multiple exercise lists delivered to the same class. It should also be noted that the system is not trivial to install and manage, which ends up discouraging the teachers from adopting it. On top of that, the teacher needs to add each student to the system, which proves to be quite tedious, especially for large classes. To overcome this drawback, teachers have implemented programs that automatically generate registration ids and corresponding passwords based on student university registration number. As BOCA does not allow password modification, students often know other students password as they are generated following certain patterns. This facilitates copying other students' solutions.

This shows that, although BOCA has several advantages, there are problems that hinder the use of the system in the classroom. To solve this, beyond what has already been presented, we aim to tackle the following requirements: automatic generation of lists of problems based on subjects and level of difficulty, measurement of the student experience with problem solving by subject, and rank problems by levels of difficulty. We consider the last item is important for didactic purposes, not finding a technique or algorithm to use, this is the focus of this work.

4. PROBOCA

From an architectural viewpoint, the proposed system builds upon BOCA that is considered an internal component and offers its structure, PHP source code and PostgreSQL DB to be reused.

In order to avoid further complication of the development process, the "Competitions and Teams" context is kept, with some adaptations. The terms "*Competition*" and "*Team*" are used to loosely indicate *exercise list* and *student* respectively. BOCA´s user interface allows for such extrapolation which is already in practice by the teachers who use this system in their classroom.

Adapting BOCA to support CP teaching brought the need to introduce new functional requirements, as presented in the use case diagram in Figure 3.

PROBOCA required some changes to the BOCA DB structure. It was necessary to modify the internal tables in order to link the stored problems to given course's syllabus and certain difficulty levels as well as to include more detailed data about the students. Furthermore, the original competition-based structure will be altered to adapt it to the concept of multiple exercise lists.

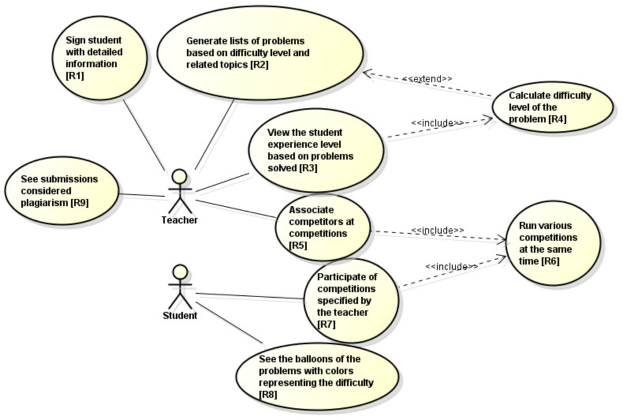
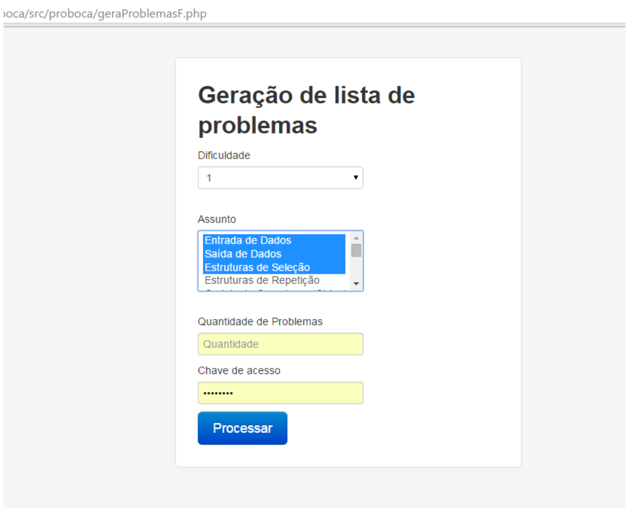

Sign student with detailed information [R1]
Generate lists of problems based on difficulty level and related topics [R2]
<<extend>>
Calculate difficulty level of the problem [R4]
<<include>>
View the student experience level based on problems solved [R3]
See submissions considered plagiarism [R9]
Teacher
Associate competitors at competitions [R5]
<<include>>
Run various competitions at the same time [R6]
<<include>>
Student
Participate of competitions specified by the teacher [R7]
See the balloons of the problems with colors representing the difficulty [R8]

Figure 3. Use Case Diagram with New Requirements

Following requirement R1, student registration is now done by the student himself using a token which is handed out by the teacher. Besides saving a password, other information is collected during registration that permits class wide statistical analysis.

R2 was intended to simplify the task of competition generation, which is performed by the teacher. To achieve this goal, a database of problems was implemented. Currently, to generate a competition, as shown in figure 4, the user indicates three parameters, namely a difficulty level (1-easy; 2-medium; 3-difficult); the desired component syllabus´ elements for the given problem set and finally the quantity of problems to compose the list. The system then analyzes which problems best fit the user-defined parameters and generates a competition list, from among the problems available in its DB. The difficulty level for each problem is estimated automatically by the system based on data obtained from solutions submitted by students and other parameters as described in section 6.



ɔoca/src/proboca/geraProblemasF.php

Geração de lista de problemas

Dificuldade
1

Assunto
Entrada de Dados
Saída de Dados
Estruturas de Seleção
Estruturas de Repetição

Quantidade de Problemas
Quantidade

Chave de acesso
•••••••••

Processar

Figure 4. Generate problem lists\competitions

R3's purpose is to provide teachers with a report of every student´s experience level based on his interaction with the system. Using the collected data, it should be possible to present information about a student's experience regarding the syllabus elements addressed in each problem, thus allowing the teacher to detect where students show more difficulty. This requirement depends on R4, which measures the difficulty level of a given problem. A simple mechanism to measure the degree of difficulty of a given problem was developed and tested.

For R3 to be reliable, it is necessary that a plagiarism identification strategy (R9) be also implemented in the system.

R5 is aimed at associating a student to multiple competitions without the need to register each time. Thus, the data from different competitions can be associated to the student, allowing the integration of results obtained for the different lists presented to a given class. This requirement has a dependency upon R6, which aims to adapt BOCA to judge problems from several competitions. R7 is also related to R5, with the goal of allowing a student to participate in several competitions - analog to answering several problem lists.

R8 aims to display the problems' balloons, color coded to indicate the problem's difficulty level. Currently, balloon colors are registered by the system administrator along with the problems and have no relation to difficulty or content, unless the classification and corresponding color attribution is done by the teacher when uploading the problems.

5. IMPLEMENTATION

The functionalities of detailing the student information and providing automatic generation of exercise lists are already implemented and tested. Requirement R4, estimating the difficulty level of a given problem, is currently in progress.

Within this requirement, the first executed task was to create 74 exercises using BOCA´s problem-registration functionality. These exercises cover the syllabus of the "Introduction to programming" course (CS1) and range from debutant programming instructions (Hello World style) up to coding with matrixes. The inserted problems populated an initial database to be used for testing, but additional problems are expected to be easily included. An estimated level of difficulty for each problem was supplied by the teacher inserting the problem in the database.

To submit an answer, the student uploads one code file per problem solved. Along with the file upload, the student is asked to evaluate the problem´s difficulty, by indicating one of three choices: Easy, Medium or Difficult, based on his experience when solving the problem (Figure 4).

Success was obtained in computing several parameters that are needed to calculate problem difficulty. They include different measures, such as counting the number of repetition and selection structures used in the solution and the number of topics involved in the problem. Problem topics were defined using the introductory programming course syllabus. They include, input/output; attribution; selection; repetition; vectors; strings; matrices and functions. In this list, topics appear in the order they are taught and therefore in increasing level of complexity for novel students. Since programming is incremental, more complex topics usually base upon lesser complex topics. Several tests have been conducted to define and validate the mathematical function that will calculate the difficulty of the problem, but this is still an open issue.

Based on the calculated difficulty and the topics involved in the problem, colored balloons are associated to the problem. Different colors represent different topics, and within each topic, the level of difficulty is represented by the intensity of the color. For example, blue represents problems whose most complex topic is selection. Light blue balloons are associated to easy problems using selection. Medium blue balloons are associated to medium difficulty problems and dark blue balloons are associated to problems that demand more from students.

The task of providing teachers with a report on student achievements (R3) has not been tackled yet. Ideas in this sense include: (1) comparing student's perceived problem difficulty and mean problem difficulty. If students perceive problems as harder or easier than the mean this could indicate that they are at a lesser or higher level of programming competency; (2) comparing successive submission of the same problem. This may show if students adopt a trial-and-error approach; (3) mean time taken to solve problems; (4) mean number of submission per problem; (5) score when compared to the other students; among others.

6. Estimating Problem Difficulty

In a sense, "difficulty" expresses the lack of ability, or amount of skill/effort needed to accomplish something. Obviously, the perception of difficulty is an individual matter that is related to many aspects, including the time at which the question was posed. Nonetheless, this work investigates the possibility of characterizing a programming problem in a way such that calculated parameters may correlate to a determined "Difficulty-level" as expressed by students.

In their work [6], Alvarez and Scott present the program control flow and number of lines of code as variables that correlate to the difficulty of a problem. The experiments undertaken in the current study corroborate this information.

Also, other works [7, 8] show the need to deal with abstraction to solve problems. A highly abstract problem is one that implies a greater level of generalization. Thus, a problem that involves many different topics can become more abstract and hence more difficult. It is fairly safe to assume that in order to measure the difficulty of a problem, it is necessary to make a detailed analysis of the topics that are addressed within the problem. That proposition helps in formulating the hypothesis below.

6.1 Problem Difficulty Mechanism

On one hand, the survey of the student´s opinion of the submitted problem's difficulty provided the first estimate. For each of the 74 registered exercises, this information was stored, along with the respective answer in the DB. Although information was collected for all students, statistics were calculated only considering those that completed at least 95% of the exercises (70 out of 74). This excluded students that dropped out the course in order to prevent their partial answers from skewing the results. After filtering, the mean difficulty "Mean_Dif" was then calculated for each exercise based on the answers of the resulting 62 students. "Median_Dif" and "Mode_Dif" were also calculated.

In addition, C source code was written to correctly workout each of the registered problems. A PHP program was also developed in order to analyze the programs´ internal structures and to extract some measures from each program. The measures include counting:

- Lines of code, represented by variable N_LC;
- Repetition structures used in the solution, N_RP;
- Selection structures, N_SL;
- Edges in a graph that represents the algorithm, N_EDG;
- Edges in a graph that represent repetition structures in the algorithm, N_EDG_RP;
- Height of a tree, with each sub-block of internally nested code representing a node, MX_HT and
- Number of topics involved in the problem, N_TPC. This last count was obtained manually.

To verify if a combination of variables obtained better results, the following formulae were also tested against the Mean_Dif variable to verify their correlation. Table 2 shows the results. Mean_Dif correlated positively with all measured variables, being the best correlation associated to f4, $r = .76$, $p = .000$, that improves on the individual correlations with N_TPC (number of topics) and MX_HT (maximum tree height).

Table 1. Correlation between measured variables and student evaluation

		Median_Dif	Mean_Dif	Mode_Dif
N_RP (Repetitions)	*Pearson Correlation* *Sig. (2-tailed)* *N*	.49 .000 74	.52 .000 74	.44 .000 74
N_SL (Selections)	*Pearson Correlation* *Sig. (2-tailed)* *N*	.16 .171 74	.34 .003 74	.19 .108 74
N_LC (Lines of Code)	*Pearson Correlation* *Sig. (2-tailed)* *N*	.44 .000 74	.62 .000 74	.45 .000 74
N_TPC (Topics)	*Pearson Correlation* *Sig. (2-tailed)* *N*	.57 .000 74	**.69** .000 74	.59 .000 74
N_EDG (Edges)	*Pearson Correlation* *Sig. (2-tailed)* *N*	.42 .000 74	.58 .000 74	.41 .000 74
MX_HT (Tree Height)	*Pearson Correlation* *Sig. (2-tailed)* *N*	.52 .000 74	**.67** .000 74	.56 .000 74
N_EDG_RP (Rep. Edges)	*Pearson Correlation* *Sig. (2-tailed)* *N*	.53 .000 74	.60 .000 74	.51 .000 74

$$f1 = \frac{N\_RP+N\_SL+N\_LC+N\_TPC+N\_EDG+MX\_HT+N\_EDG\_RP}{7}$$

$$f2 = N\_EDG \,.\, 0.3 + MX\_HT \,.\, 1.4 + N\_EDG\_RP \,.\, 1.4 + N\_LC \,.\, 0.2$$

$$f3 = \frac{N\_TPC+MX\_HT+N\_LC}{3} + \frac{N\_EDG\_RP+N\_EDG+N\_RP + N\_SL}{8}$$

$$f4 = \frac{N\_TPC + MX\_HT}{2}$$

Table 2 shows the results. Mean_Dif correlated positively with all measured variables, being the best correlation associated to f4, r = .76, p= .000, that improves on the individual correlations with N_TPC and MX_HT.

Table 2. Correlation of student perceived difficulty and developed formulae

		Mean_Dif
f1	*Pearson Correlation* *Sig. (2-tailed)* *N*	.66 .000 74
f2	*Pearson Correlation* *Sig. (2-tailed)* *N*	.68 .000 74
f3	*Pearson Correlation* *Sig. (2-tailed)* *N*	.67 .000 74
f 4	*Pearson Correlation* *Sig. (2-tailed)* *N*	**.76** .000 74

We also verified correlations between teacher evaluations of the problems' difficulty. For this we asked five teachers to read each problem and attribute a level of difficulty in the range 1-5. Since we had only five evaluations we chose to work with the median difficulty obtained (Median_Prof_Dif).

Correlating this variable to the measured variables we obtained the results presented in table 3. Best correlations were obtained with N_RP and N_EDG_RP, *r* = .62, *p* = .000, both related to the number of repetition structures found in the solution codes.

Table 3. Correlation between measured variables and teacher evaluation

		Median_Prof_Dif
N_RP (Repetitions)	*Pearson Correlation* *Sig. (2-tailed)* *N*	**.62** .000 74
N_SL (Selections)	*Pearson Correlation* *Sig. (2-tailed)* *N*	.20 .093 74
N_LC (Lines of Code)	*Pearson Correlation* *Sig. (2-tailed)* *N*	.56 .000 74
N_TPC (Topics)	*Pearson Correlation* *Sig. (2-tailed)* *N*	.50 .000 74
N_EDG (Edges)	*Pearson Correlation* *Sig. (2-tailed)* *N*	.53 .000 74
MX_HT (Tree Height)	*Pearson Correlation* *Sig. (2-tailed)* *N*	.50 .000 74
N_EDG_RP (Rep. Edges)	*Pearson Correlation* *Sig. (2-tailed)* *N*	**.62** .000 74

Table 4 correlates the teacher defined difficulty with the proposed formulae defined above. Positive correlation was found with all formulae, being the best correlation associated to f2, *r* = .63, *p*=.000. Furthermore, a positive correlation was found between the teacher defined difficulty and mean student perceived difficulty, *r* = .69, *p* = .000.

Table 4. Correlation of teacher defined difficulty and developed formulae

		Mean_Dif
f1	*Pearson Correlation* *Sig. (2-tailed)* *N*	.59 .000 74
f2	*Pearson Correlation* *Sig. (2-tailed)* *N*	**.63** .000 74
f3	*r** *Pearson Correlation* *Sig. (2-tailed)* *N*	.59 .000 74
f4	*Pearson Correlation* *Sig. (2-tailed)* *N*	.55 .000 74

Although student perceived difficulty and teacher defined difficulty are correlated, there are differences that can be verified by the correlations found with the measured variables and proposed formulae. This could be explained by the fact that students evaluate difficulty based on the knowledge they have when developing the solution. As they do not know what comes ahead, they cannot base their evaluation on the overall knowledge of programming. Teachers on the other hand, have this overall view of the domain and evaluate difficulty accordingly. It must be observed that the teachers that did the evaluation are new to teaching and have not yet acquired a more critical understanding of the difficulties students encounter when learning to program. After developing algorithmic reasoning, people tend to forget how they thought before, and find that many concepts are obvious when in fact, for beginners, they are not.

7. CONCLUSION

PROBOCA is a system under development whose goal is to adapt the BOCA software for use in teaching programming. While BOCA itself was developed for programming marathons, it is already in use, as a support tool, in programming introductory courses. As a learning aid, BOCA has several limitations, especially relative to class administration and student monitoring. In addition, as a system specifically developed for competitions, it lacks mechanisms that facilitate the creation of exercise lists, such as a question bank, and analysis of student performance.

PROBOCA supports the persistence of problems registered by teachers in the database and provides greater access to students' related information. Unlike other "Online Judge" systems that are available exclusively online and are managed by their creators, PROBOCA can be downloaded and installed by the teacher, giving the teacher control over the problems stored in the database. Since teachers are responsible for introducing the problems, this solution has the additional advantage that it is language free, i.e., it is not limited to teachers and students that speak the language in which the problem specifications were written as is the case of online systems administered by third parties.

In addition to implementing an environment that facilitates the use of BOCA in teaching programming, PROBOCA also aims to provide teachers and students with information that will help students in their learning process. One of the important features of PROBOCA is an automatic evaluation of problem difficulty. This gives students direction in the path to follow when choosing which exercises to solve first, allowing them to solve easier exercises before more complex ones, diminishing student frustration at not solving problems. It also allows teachers to better gauge the level of difficulty of exercise lists. As shown by the collected data, teacher and student evaluation regarding problem difficulty do not match, and this may lead to distortions when teaching programming. Future work could include developing a system that will automatically suggest problems to students based on their performance and calculated problem difficulty.

This work shows the approach being taken to calculate the difficulty of the problems. This approach differs from others by treating the algorithms submitted by students in the form of graphs and trees to identify properties that could be correlated with the difficulty of problems. For this, data mining using correlations was undertaken.

Part of the system has already been implemented, and has been successfully used in the classroom. The success of these tasks shows the feasibility of the project and encourages further work.

REFERENCES

[1] BOCA. *BOCA Online Contest Administrator*. Available in <http://www.ime. usp.br/~cassio/boca/> Access on March 20th, 2015

[2] Chaves, J. O. et al. *Uma Ferramenta de Auxílio ao Processo de Ensino Aprendizagem para Disciplinas de Programação de Computadores (2013).* TISE 2013

[3] URI. *URI Online Judge*. Available in <http:// www.urionlinejudge.com.br/> Access on May 20th, 2014.

[4] SPOJ. *SPOJ Brasil*. Available in <http://br.spoj.com/embed/info/> Access on May 20th, 2014.

[5] Santos, J. C. S., Ribeiro, A. R. L. *JOnline - proposta preliminar de um juiz online didático para o ensino de programação (2007).* SBIE 2011.

[6] Alvarez, A. and Scott, T. A. (2010). *Using student surveys in determining the difficulty of programming assignments.* Journal of Computing Sciences in Colleges, 26(2):157–163.

[7] Gomes, A. et al. (2008). *Aprendizagem de programação de computadores: dificuldades e ferramentas de suporte. Revista Portuguesa de Pedagogia*. 42(2).

[8] Mendonça, A. et al. *Difficulties in solving ill-defined problems: A case study with introductory computer programming students*. In Frontiers in Education Conference, 2009. FIE'09. 39th IEEE, pages 1–6. IEEE.

[9] Mendonça, A., de Oliveira, C., Guerrero, D., and Costa, E. (2009). *Difficulties in solving ill-defined problems: A case study with introductory computer programming students*. In Frontiers in Education Conference, 2009. FIE'09. 39th IEEE, pages 1–6. IEEE.