A multidimensional ELO model for matching learning objects

André Prisco, Rafael Penna, Evandro Junior and Silvia Botelho
Federal University of Rio Grande (FURG)
Email: {andre.prisco,rafaelpenna,evandro,silviacb}@furg.br
Neilor Tonin and Jean Bez
Integrated Regional University
Email: {neilor,bez}@urionlinejudge.com.br

***Abstract*—This research-to-practice full paper proposals a metric of multiple skills for learning of programming students. This kind of system often need to diagnose the student's skill level. In the same way it needs to know the level of difficulty learning objects in its database. Such information makes it possible to make an appropriate match between student and the learning object. To model such tasks, we have adapted the ELO technique to apply a matchmaking process similar to that used in choosing opponents in chess tournaments or online matches. We used as a case study a virtual learning environment which has a repository with programming problems and the users interaction log. In this work we propose an extension to the traditional ELO model. In the classical model, ELO is a scalar value for each student and for each learning object. The extended model considers ELO as a multidimensional quantity, where each dimension is a skill in solving programming problems. The enumeration of the skills was made using the literature as well as statistical data of relevance of the attributes. The results are presented in this work.**

***Keywords*—ELO, multidimensional, matchmaking, learning objects.**

I. Introduction

Higher education in computing is composed of heterogeneous classes, with students with different educational backgrounds. These, present different profiles, with their different skills.

Technology can be useful to enable the perception of such skills and offer different pedagogical approaches, each one seeking to be more appropriate to each student.

The objective of this work is to present a metric of multiple abilities for programming. However, it is not our objective to apply this metric to create rankings, but to provide the means to match learning objects and students, considering for each student his/her most developed skills and those that need improvement.

For this purpose, we use the models applied in games to integrate them into learning models. Matchmaking systems in electronic games aim to find the best opponent for a match [1], [2]. For a player who wants to improve his/her performance, that is, to effect his/her learning in the game, the best matches are those in which he/she can improve his technique. Weaker opponents are easier to win, but offer little learning opportunity, while too strong opponents can be demotivating to the player, who does not have the support to appropriate much more advanced techniques. Systems then seek to find opponents that enhance learning.

We have adapted one of these techniques, ELO, to infer, rather than an adversary, more appropriate programming challenges for computer students. We expect this model to be the basis for a future recommendation system that considers students' learning and motivation.

We used as a case study a virtual learning environment that has a repository with programming problems. This platform works through a submission system, where students send algorithmic solutions to the presented problems, receiving automatic feedback. We consider each problem with its feedback mechanism as a learning object and the submissions as the student's interactions with such learning object. The platform is used in classrooms and contests, with thousands of students and millions of submissions registered. It offers the opportunity to compare different cases and behaviors through the record of each student on the platform.

In the next sections, we present pedagogical models related to programming teaching, our adapted ELO model, the methodology for the experiment and finally we discuss the results.

II. Pedagogical Models in Teaching Programming

Initially, in this section, we present some concepts that we will use in the course of this work, related to the genetic epistemology of Piaget [3], [4]. Subsequently, we discussed pedagogical models used in programming teaching.

The focus of our study is in the Piaget's post-formal stage, which is the stage that studies the adult learning [5], [6]. In [7], some concepts related to this cognitive approach were presented, such as Schemas, Assimilation, Accommodation, etc. Among the concepts coined by Piaget, we present as main for this work, the cognitive conflict. It occurs when the mind is faced with a serie of problems presented in order to respect its equilibration [3]. At this moment, the person can put in a state of cognitive motivation in learning, that is, the curiosity itself or the "annoyance" by not assimilating the problem, it arouses the interest in its solution. Such motivation may even be independent of external elements, such as school assessments [4]. Perhaps one of the most challenging tasks in the learning process is to use technology to gather all the information we have about each student so that we can present ways that cooperate with this state of motivation.

978-1-5386-1174-6/18/$31.00 ©2018 IEEE

Such concepts can be understood from the perspective of programming teaching. Problem-solving skills are fundamental to programming learning. A teaching model based only on passing information on computing is less effective than a model that includes the manipulation of knowledge through programming problems [8], [9].

A student-centered approach involves choosing problems that motivate the student and catalyze the learning process. For this, we use cognitive theories, more specifically genetic epistemology, as the basis for our model of problem choice.

The practice of exercises is an important didactic tool because it allows the student to manipulate the concepts to be learned through the programming challenges. However, the simple practice of any exercise on the subject does not guarantee such learning.

Using the theory of Equilibration [3], we understand that the exercise is effective when it proposes a challenge, that is, it causes gaps in the schemes previously constructed by the student. When faced with such gaps, the student creates new structures accommodating what has been learned in the challenge to previous schemes. As discussed in [9] , new structures may reflect new concepts that require previous concepts and generalizations of concepts already worked in some aspect.

The continuous process of challenge and overcoming is for the student a trigger of motivation that does not depend on external rewards, such as assessments or punishments. The very sense of mastery over the object to be learned is the catalyst for such motivation [4], [10]. On the other hand, the student's interaction with exercises that do not challenge him can be more detrimental than beneficial. These do not promote learning and discourage the energy that the student spends on them. Exercises that are more difficult than their skill means that the student cannot create new structures because they do not have enough previous schemes to assimilate the new challenge [3], [4].

Therefore, a computational model that works on students' perception of learning is an important step towards the construction of tools for personalization and improvement of computer teaching. In addition, technology has enabled the emergence of new types of learning objects with new approaches to teaching. The next section presents a specific type of learning objects that is used in our model.

III. Learning Objects with Automatic Evaluation

There are several definitions of learning objects. Downes [11] defines learning objects as educational resources that can be used in learning supported by technology. They are modular units that can constitute lessons and courses. A learning object can be based on a text, a simulation, a website, an image, a video, an application, or any other resource that can be used in learning.

Our model uses a specific type of learning object, which we call Learning Objects with Automatic Embedded Evaluation. This Learning Object has, in addition to the content, information so that an automatic evaluation of the student's interaction can be made, that is, a feedback if the student was successful in solving the problem or not.

We need to point out that by creating this type of learning object, the author plans on the skills that students will develop in interacting with them. Problem-making is a process that occurs only for didactic purposes, designed to develop specific skills, not necessarily to solve a real problem. All the parts that make up the problem, for example, are designed to be interesting and challenging for students.

Some examples of this type of learning objects are: multiple-choice exercises, quizzes and programming contests. In programming contests the students, individually or through teams, try to solve programming problems. The learning objects present texts and images describing the problem, classification by level of difficulty and subject, among other information. They also have templates that are used to verify whether the user interaction was successful or not. The learning object gives a positive feedback if the user solves the problem. If not, it gives a negative feedback.

An important feature of these learning objects is that they can be used on large scale by a very large number of users, producing amounts of data to be analyzed.

Among the many existing learning objects, finding the right one for each student profile is a great challenge. The next section presents classification metrics in order to establish profiles and to find matching between student and Learning Object.

IV. Ranking Metrics in Games and Cognitive Theory

This section presents some metrics for classifying students and learning objects. In general, these metrics are used to establish student profiles and find the most appropriate learning objects by personalizing teaching.

A tutoring system to assist students in learning programming is presented in [12]. Protus was designed and implemented as a mentoring system, able to recommend useful and interesting materials to students based on their different knowledge, preferences, learning purposes and other characteristics. The system recognizes the student's learning style and allows him to take different paths in the development of learning activities. The AprioriAll [13] algorithm is adopted to extract behavioral patterns from the record of activities done by students and recommend the most appropriate learning paths.

The research of [14] presents an adaptive classification mechanism for personalizing of teaching using repositories on the Internet. The system uses approaches based on the preferences and interests of the "neighbors" to classify the degree of relevance of the learning objects according to the intention of a user.

The metrics designed for games can be useful in the task of relating activities and students. Matchmaking is the task that game promoters and coaches make to choose the best opponent for a player [15]. In this case, the goal is not to choose the opponent that most pleases, but the one that enhances the skill gain and minimizes the risk of demotivation. In the case of the method created by [15], players are chosen for a match in order to have a higher gameplay, taking into account factors such as language, skill level and location.

One of the most used models for this task was developed for chess tournaments and today it is used in sports and electronic games. ELO aims to quantify players through their game histories. ELO is a statistical ranking system that can calculate relative skill level values for competitors or machines in competitive games [16]. Briefly, the ELO system works as follows:

- Each player has an associated skill value (ELO);
- At each match, the player's ELO values are updated;
- Before the match occurs, there is a calculated probability (based on the ELOs of the two players) of player 1 to beat player 2 (and vice versa);
- After the match, the ELOs of both players are changed according to their current values and the result of the match;
- After the game, if the player most likely to win wins, the ELOs of both players undergo minor changes;
- After the game, if the player less likely to win wins, the ELOs of both are significantly changed;

Next, we present the formal definition used in our work.

R=$\{0,1\}$ is the set of results of a game. 1 for win and 0 for loss. Given a match between player i and player j, with an ELO θ_i e θ_j, respectively, the expectation that i wins j ($R_{i,j} = 1$) is given by the equation:

$$P(R_{ij} = 1) = \frac{1}{1 + 10^{\frac{\theta_j - \theta_i}{400}}} \tag{1}$$

At the end of the match, the new ELOs are calculated according to the expectation of the results, the previous ELOs and a constant *k*. The greater *k*, the more important is the match in the change of the ELOs. Observe the equation below:

$$\theta_i = \theta_i + k(R_{ij} - P(R_{ij} = 1)) \tag{2}$$

Further information on the formalization of ELO can be found in [17].

For this work, we present the hypothesis that the process of improving in a game is an equilibration process. By choosing matching players, ELO emerges as a facilitating mechanism for the equilibration process. Players have a scheme formed to deal with the game. When faced with more difficult players with adequate ELOs, they do not have formed schemes and need to go through the stages of assimilation and accommodation, so that they improve some skills (increasing their ELOs), finally reaching a new state of equilibrium. In these situations, we believe that subjects are motivated and learning through the manipulation of appropriate objects (matches with a good level of challenge) and of the cognitive conflict provoked.

Thus, we can observe in this type of game a Piaget learning process. So, the metrics of game performance can be an indicator for the evaluation of learning.

We used an adapted ELO model. While the original model deals with the relationship of a player to another player, our approach adapts the metric to relate the student to the problem with which one interacts. Genetic epistemology is relational, that is, it considers learning as a relation of the individual to the object to be learned. In this way, assigning ELOs to problems as well as students and then comparing them is an elegant way to modify the metrics to fit our model.

We consider each interaction with a learning object a game, a "duel" between the player and the problem. As discussed by Piaget, the learning process allows to modify and manipulate the object to be learned, in the same way that modifies the learner himself. This concept is modeled from the variation of students ELOs and learning objects ELOs. At each start of interaction with positive feedback indicates that the student "won the game" while negative feedback indicates that the "problem has won". After the evaluation, the ELOs are then updated. Of course, learning objects that require more of the students will have their ELOs raised. More experienced students will also have larger ELOs.

This approach has also been adopted in other works in the literature. In [18] the authors present several ELO applications in education and compare their use with IRT [1] [19] as a student skill metric. The authors state that ELO is a versatile and simple approach, which allows the creation of different models. In [18] is presented a system that measures with ELO geography knowledge, as well as other similar initiatives for mathematics, reading and grammar.

In [7] programming problems are evaluated in a similar way, based on the log of an online system of contests of programming problems.

These approaches consider that students and learning objects constitute a unidimensional model, assuming that the theme of the proposed challenges can be reduced to a skill level. In the case of programming contests, it is assumed that the programming is formed by one great concept with levels of maturity.

In this work, we assume that the relationship between the student and the learning objects comprises a set of skills that may be relatively independent or orthogonal. In this context, a learning object may require higher levels in one skill and more basic levels in others. It may not even require some skill. In the same way, a student has different levels for each skill. To do this, we extend the ELO metric to make it multidimensional, in which each dimension is a skill or concept that the student must have for himself at some level. Some previous studies have proposed multivariate approaches. A formal multivariate ELO model for psychometric analysis is presented in [20]. The following is our approach.

A. Multidimensional ELO Model

Each student s has his or her skills represented by $\overrightarrow{\theta_s} = (Skill_1, Skill_2, ..., Skill_N)$, where $Skill_i$ is his/her ELO on skill i. Each Learning Object l has its level of requirement represented by $\overrightarrow{\sigma_l}$ and by relevance $\overrightarrow{m_l}$, such that the set of learning objects can be represented by $L = \{(\overrightarrow{\sigma_1}, \overrightarrow{m_1}), (\overrightarrow{\sigma_2}, \overrightarrow{m_2}), ..., (\overrightarrow{\sigma_n}, \overrightarrow{m_n})\}$, where n is the number of learning objects.

[1] Item Response Theory

Relevance is a real number between 0 and 1 that indicates how important a skill is for interaction with a learning object. Regardless of the difficulty level, relevance equals to 1 indicates that the skill is critical for a positive feedback, while relevance equals to 0 indicates that skill is not required. Intermediate values indicate that the use of a particular skill or concept is positive but optional. It also indicates that it can be used but it's not the central challenge of the learning object.

Each interaction is a tuple I = (S, L, R), where R = $\{0,1\}$, with 1 indicating that the student had a positive feedback or 0 indicating negative feedback after the interaction. Given the interaction of student i with learning object j, the adaptation to the model can be represented by the equations below. For each Skill s:

$$\theta_{i_s} = \theta_{i_s} + m_{j_s} k(R_{ij} - P(R_{ij} = 1))$$

$$\sigma_{j_s} = \sigma_{j_s} + m_{j_s} k(R_{ji} - P(R_{ji} = 1))$$

In the next section we present the experiment using the multidimensional model.

V. Methodology

In this section, we present the methodology used to process and analyze data from a repository of programming learning objects.

Currently, the chosen repository has about 1500 learning objects (programming problems), thousands of users and about 10 million submissions (feedback of students' interactions with learning objects). The main users are students who submit their solutions to programming problems as an activity evaluated in their courses, by students who want to practice independently and by students who train for programming contests. The students are free to choose the problems they want to solve. The system is available online [2]. It is possible to consult the history of each user and there is a ranking with those with more accepted submissions. Good ranking users have more than 1000 submissions in their history. Frequent users have, in general, at least 100 submissions.

The system does not provide a recommendation tool. Students can freely choose which problems they wish to submit a solution to. The same way as presented in [7], skills analyze and metrics are based on log submissions. The problem information and submission records I are available in log.

TABLE I. Submission database sample - [7]

User Id	Problem Id	Submission Time	Feedback	Language
7xxxx	1134	"2016-05-05 01:10:09"	Accepted	C
5xxxx	1154	"2016-01-04 20:41:39"	Accepted	Java
4xxx	1002	"2015-02-03 20:57:46"	Wrong Answer	C++

A. Skills definition

It is not an easy task to define a set of skills for computer science. The area is very wide and applied concepts are interrelated. In [21], [22], computational thinking presents some skills that are relevant in solving computational problems and in developing computer systems. The main concepts are logical/mathematical thinking, problem decomposition, pattern detection, abstraction, algorithms, problem evaluation and data representation.

Important skills and concepts in programming learning and how they are related are presented in [9]. In this work, the concepts are related from the point of view of the learning process. Concepts are hierarchically related. Some concepts are generalizations of several others already learned. Others are specifications. The concepts are also presented as small steps while others have more important jumps.

Sijin Joseph [23] proposes a matrix of programming skills. Among the skills are the appropriation of the use of data structures, analysis of algorithms, programming logic, problem decomposition and testability of a solution. For each skill, indicators are presented that seek to insert the student or professional into one of the 4 maturity levels defined. Raphael Poss discusses in [24] a metric for programming skills using guides similar to European Union models for competence in foreign languages [3].

We use the above models making adaptations considering the group of system users, computer science curricula, and the style of problems at the repository. The skills have a level of specificity suitable for analysis and are concrete concepts enough to be related to the problems. Below we present the 8 skills adopted:

- Programming logic: covers programming and logic knowledge to solve problems from the simplest problems, sequential problems, those involving logical operators, conditionals and loops. Patterns, abstraction and knowledge of the language are related to this concept.
- Problem integration: covers the knowledge needed to interact with the evaluation tool. Even the simplest problem requires a minimum of knowledge to read the input and display the output as required. Some problems may require more sophisticated outputs. It is common submissions that result in errors not because of programming or logic problems, but because they do not present the data correctly. In addition to the format, some problems require an understanding of the numerical data of the problem, both in data entry and output.
- Advanced Algorithm: covers concepts that go beyond basic programming logic. It is necessary to know programming techniques such as greedy methods, dynamic programming, divide and conquer, search and ordering methods, among others. The student should know how to analyze the complexity of the algorithm that should develop. It is normal wrong submissions for "timelimit" when the solution is correct but is not processed in the time required by the problem. Therefore, the main factor for this ability is the appropriation of the computational resource concept and the algorithm analysis.
- Linear Structures: stacks, queues, lists, are often applied to the problems repository. In some problems,

[2] www.urionlinejudge.com.br/

[3] $http://www.coe.int/t/dg4/linguistic/cadre1_en.asp$

the relevance is high, since the main goal of the problem is to test the student's mastery of some structure. In others, the student must use some structure as a way to the solution. In other cases, the application of structures is desirable, but it is possible to solve the problem without its application. In the latter case, the relevance is lower. In most cases, arrays are used for data entry storage.

- Strings: textual data processing. Although this could be clustered into linear structures, there are a range of problems involving specific features of word processing, substring, search and others that specifically involve such structure.
- Nonlinear Structures: domain and application of graphs and trees. Many problems in the database require knowledge in applying such structures, as well as knowing which ones are most efficient and appropriate in each situation. Most problems that require more advanced programming techniques also require mastery of non-linear structures.
- Mathematics: We consider this skill as the domain of knowledge in algebra and geometry and how to apply such concepts in problem-solving.
- Modularization: ability to divide a problem into subproblems to enable or facilitate its solution. It presents in problems like divide and conquer, recursive structures, processing of strings among others.

B. Analysis of the repository problems

In the problem repository, each of these is related to a list of concepts and algorithms linked to its solution. The terms are filled out by the authors of each problem. With these terms, and knowing the skills proposed, three experts made the relationship between problems and skills. Each problem received a relevance rate for each of the 8 skills. The relevance values range from 0 to 1 in intervals of 0.1.

C. Applying the methodology

The methodology, presented in Figure 1, is similar to that applied in [7] and [18]. Data about students, learning objects (problems) and interactions (submissions) are collected and stored in a database. Initial ELOs are assigned to each student and each problem. The level of relevance to each skill is also assigned to each problem.

We analyzed the submissions of 74245 students in the period from February 2012 to July 2016. A total of 4.5 million submissions on 1071 problems.

Subsequently, the submissions were analyzed in chronological order and used for calculating the update of both the ELO of the problems and of the students. Each submission is analyzed as a match and the ELO in each dimension is updated. Finally, the results are analyzed in our database. Some analyzes are presented in the following section.

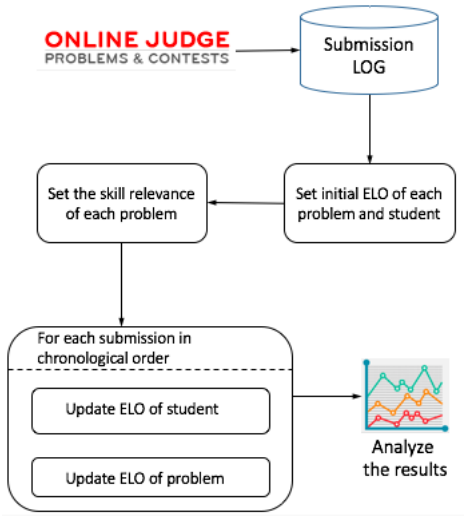

Fig. 1. Methodology adopted for system log analysis

VI. Analysis of Results and Conclusion

After the experiment application, we analyzed in a preliminary way the results obtained. For this work, we decided to analyze the skills evolution graphs of 4 computer engineering students. These students are frequent users of the tool for more than 2 years and submitted their programs in a period prior to our project, therefore, they were not influenced by our analysis.

We try to establish 4 students profiles and understand their skills curves:

- Profile 1: students with learning difficulties, without prior knowledge in programming.
- Profile 2: students without prior programming knowledge and with satisfactory learning throughout the course.
- Profile 3: students with prior programming knowledge and with great learning evolution throughout the course.
- Profile 4: advanced students who participate in contests.

Note that the number of submissions is important but not critical to getting ELO [16], [17]. The Model is flexible in relation to the number of interactions with learning objects, that is, a higher number of submissions does not necessarily imply a higher ELO or a higher quality student. Analyzes beyond ELO may study the correlation between number of submissions and student motivation.

A. Profile 1

This profile represents, in a general way, those students who did not know to program when they started to use the system and who also usually present a great difficulty in learning programming. Figure 2 is about a student who, for 3 years, made the first year of the computer engineering course and is currently in the second year. Note that in the area of the graph marked by region *A*, the student attempts to solve only problems that require *logical* skill. In this period, the

student's ELO decrease, since he/she is learning the basics of programming and still does not know how to solve the problems.

In a second moment (region *B*), the student continues to make problems that require the logical skill, but also that require skills to deal with the system. Note that while ELOs in these skills continue to decrease, there are small regions of skill enhancement. This seems to indicate that the student presents difficulty in learning programming logic. In this area, there are two new skills (*strings* and *linear structures*), however, they remain practically constant. This seems to indicate that few problems made in that period have relevance in these skills.

In the *C* area, new skills emerge, but the most important is the ELO increase in logic skill, which indicates that the student, after a long time, actually began to learn the basics of programming.

Finally, *D* area seems to indicate the moment that the student reaches the second year of the course, where he begins to work on strings, linear and nonlinear structures, and advanced programming. Even though subtly, the student shows an ELO increase in practically all skills.

B. Profile 2

This profile represents those students who did not know how to program when they started using the system, but who seem to improve satisfactorily in programming learning. Figure 3 shows the evolution of a student who started computer engineering without knowing how to program and finished the course in a standard way (in 5 years).

We can note that area *A* shows the period in which the student is beginning to learn programming, doing problems that only require the *logical* skill. In this period, the student's skill decreases because he is not yet able to solve most of the problems he tries.

In a second time, area *B* of the graph, the student begin to understand the basic concepts of programming and how to deal with the system. We can see an increase in these skills as well as the raise of new skills. Among the new skills worked, we can see that *linear structures* appear with a small increase, which is natural, since, after the basic concepts of programming are learned, this is the next subject to work.

In area *C* of the graph, the student begins to make problems that require *nonlinear structures* and *advanced algorithms* skills, which probably indicates the students began the second year of the course. These problems are often more difficult, which decrease the student's skill in this period.

Finally, the area *D* of the graph shows student's ELO raise in practically all skills. This seems to indicate the student passing through in the final years of the course, where it acquired maturity and improved in all skills related to programming.

C. Profile 3

Profile 3 comprises students with programming knowledge background and with a great evolution of learning throughout the course.

In this profile, we can see that the evolution of the programming skills of a student who learned how to program before to starts the computer engineering course. The graph is presented in Figure 4.

In area *A* of the graph, we can see that, when starting to use the system, the student does not present many difficulties, increasing several skills from the beginning.

In area *B*, the student seems to focus on solving problems that deal with two problematic skills (*modularization* and *nonlinear structures*), increasing his ELO. Only *advanced algorithms* and *math* skills are not worked in this period.

In area *C*, the student appears to begin working with problems that involve multiple skills, which are more difficult and which lower their ELOs at first.

Finally, in area *D*, the student presents a consistent evolution, ending with all skills very close, which indicates an excellent formation.

D. Profile 4

The fourth profile represents advanced students who participate of programming contests throughout the course. Figure 5 shows the evolution of a student with a particular characteristic: he chooses the problems well. When choosing appropriate problems, that is, not so easy and not so difficult, according to their skills, their ELOs vary little, but consistently.

Area *A* of the graph shows the student's period of adaptation to the system, since his/her *logical* ability increases and few skills appear.

In area *B* of the graph, the student begins to work on multiple skills to solve problems. Note that there is little variation, but a consistent increase.

Area *C* of the graph represents the period of the course in which the student began to participate in programming contests events. By participating in this type of event, in his graduation, the student is faced with more complex and difficult problems. Note that at the end of area *C*, the student has his ELOs decreased, so far that he seems to find problems compatible with his ability (beginning of area *D*).

In area *D*, the student progresses again consistently, finishing with all abilities practically at the same level.

E. Average Behavior

To expand the analysis of the results, we decided to present an average of students according to the ranking of the submission tool (Figures 6 and 7) and to observe if the average behavior of these students corresponds or has some similarity to our chosen individual profiles. We were somewhat satisfied to see that the average behavior of the best placed in the ranking of the contests of the system (Figure 7) present similarity to our profile 4. The average behavior of students with learning difficulties (Figure 6) is similar to profile 1. This is an indicator that students with similar profiles have similar ELO behavior as well. Graphs can be used for clustering students in later analyzes. We emphasize that in this analysis the students of the individual profiles (profiles 4 and 1) are not part of the analysis of the average behavior, to avoid some bias in the analysis.

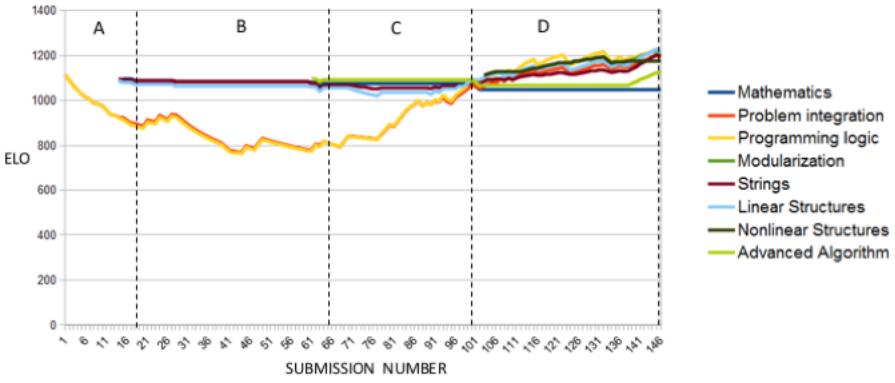

Fig. 2. Student skills of profile 1

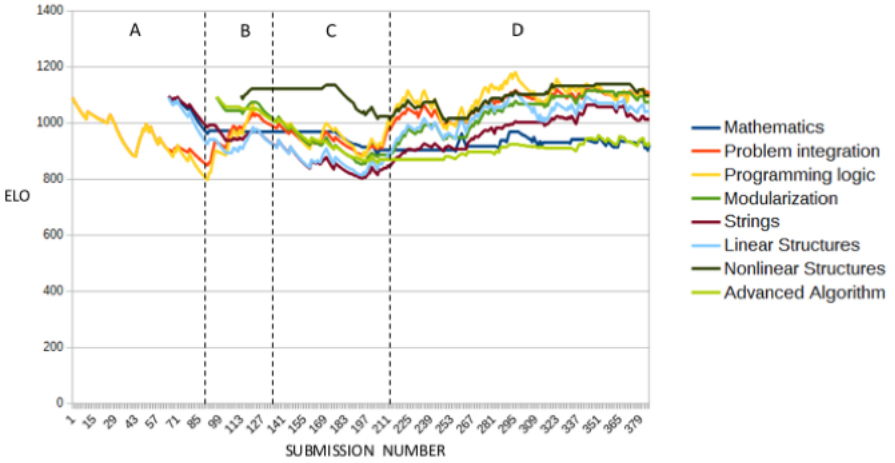

Fig. 3. Student skills of profile 2

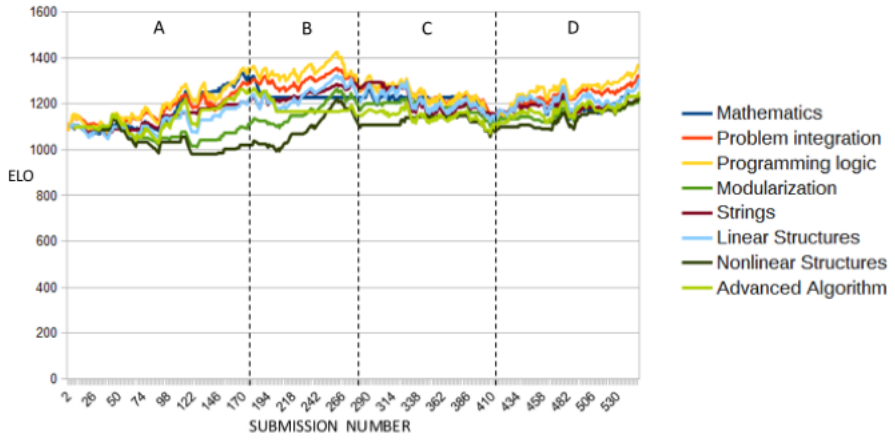

Fig. 4. Student skills of profile 3

Finally, it should be emphasized that the understanding of the different profiles of students has much to contribute to the personalization of teaching. We believe that by using ELO together with the multi-skills approach, we have taken a step towards greater customization of programming teaching. Other analysis will be done on future work, as well as comparison with other methodologies of data analysis in education (*Learning Analytics*). We believe that the results of this work will bring useful information to teachers and students in the engineering learning process.

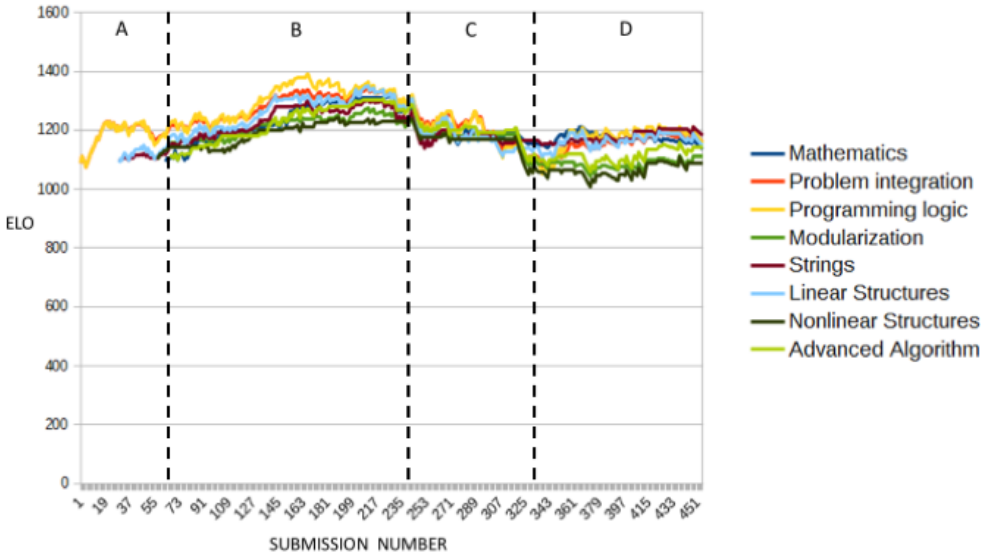

Fig. 5. Student skills of profile 4

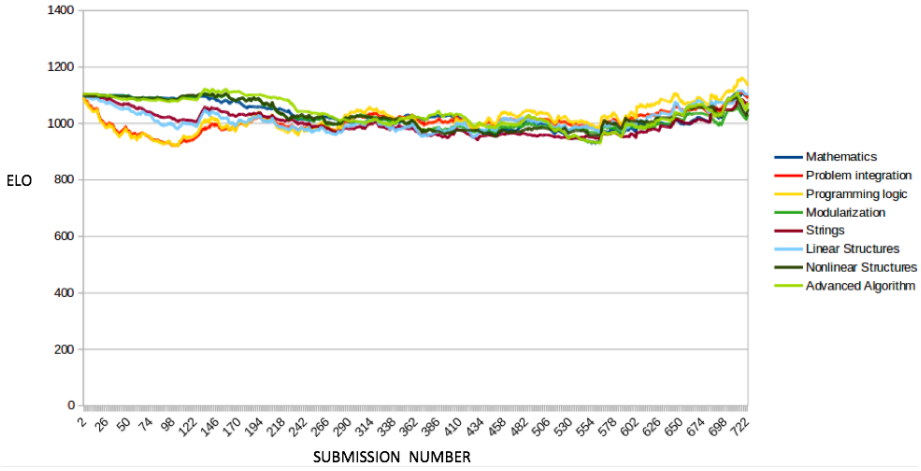

Fig. 6. Average ELO sequence of users with low performance at environment

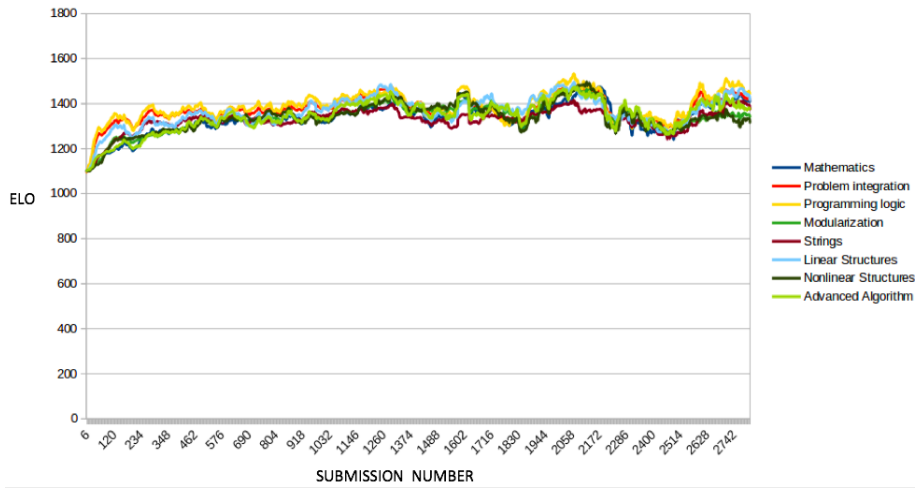

Fig. 7. Average ELO sequence of users with best ranking at environment

REFERENCES

[1] Z. Chen, S. Xue, J. Kolen, N. Aghdaie, K. A. Zaman, Y. Sun, and M. Seif El-Nasr, "Eomm: An engagement optimized matchmaking framework," in *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017, pp. 1143–1150.

[2] J. Huang, E. Yan, G. Cheung, N. Nagappan, and T. Zimmermann, "Master maker: Understanding gaming skill through practice and habit from gameplay behavior," *Topics in cognitive science*, vol. 9, no. 2, pp. 437–466, 2017.

[3] J. Piaget, "Intellectual evolution from adolescence to adulthood," *Human development*, vol. 15, no. 1, pp. 1–12, 1972.

[4] G. R. Lefrançois, *Theories of human learning: What the professor said*. Cengage Learning, 2012.

[5] P. K. Arlin, "Cognitive development in adulthood: A fifth stage?" *Developmental psychology*, vol. 11, no. 5, p. 602, 1975.

[6] C. C. Knight and R. E. Sutton, "Neo-piagetian theory and research: enhancing pedagogical practice for educators of adults," *London Review of Education*, vol. 2, no. 1, pp. 47–60, 2004.

[7] A. Prisco, R. dos Santos, S. Botelho, N. Tonin, and J. Bez, "Using information technology for personalizing the computer science teaching," in *2017 IEEE Frontiers in Education Conference (FIE)*. IEEE, 2017, pp. 1–7.

[8] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson, "A survey of literature on the teaching of introductory programming," *ACM SIGCSE Bulletin*, vol. 39, no. 4, pp. 204–223, 2007.

[9] D. B. Palumbo, "Programming language/problem-solving research: A review of relevant issues," *Review of educational research*, vol. 60, no. 1, pp. 65–89, 1990.

[10] R. M. Ryan and E. L. Deci, "Intrinsic and extrinsic motivations: Classic definitions and new directions," *Contemporary educational psychology*, vol. 25, no. 1, pp. 54–67, 2000.

[11] S. Downes, "Learning objects: Resources for learning worldwide," in *Online education using learning objects*. Routledge, 2012, pp. 47–57.

[12] A. Klašnja-Milićević, B. Vesin, M. Ivanović, and Z. Budimac, "E-learning personalization based on hybrid recommendation strategy and learning style identification," *Computers & Education*, vol. 56, no. 3, pp. 885–899, 2011.

[13] W. T. H. Pi-lian, "Web log mining by an improved aprioriall algorithm," *Engineering and Technology,* , vol. 4, no. 2005, pp. 97–100, 2005.

[14] K. H. Tsai, T. K. Chiu, M. C. Lee, and T. I. Wang, "A learning objects recommendation model based on the preference and ontological approaches," in *Advanced Learning Technologies, 2006. Sixth International Conference on*. IEEE, 2006, pp. 36–40.

[15] J. Davis, M. Feldman, and B. Johansen, "System and method for combining automatic opponent matching for computer gaming with chat room searchers," Aug. 8 2003, uS Patent App. 10/637,048.

[16] A. E. Elo, *The rating of chessplayers, past and present*. Arco Pub., 1978.

[17] A. Elo, "New uscf rating system," *Chess Life*, vol. 16, pp. 160–161, 1961.

[18] R. Pelánek, "Applications of the elo rating system in adaptive educational systems," *Computers Education*, vol. 98, pp. 169 – 179, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S036013151630080X

[19] F. Drasgow, M. V. Levine, S. Tsien, B. Williams, and A. D. Mead, "Fitting polytomous item response theory models to multiple-choice tests," *Applied Psychological Measurement*, vol. 19, no. 2, pp. 143–166, 1995.

[20] P. Doebler, M. Alavash, and C. Giessing, "Adaptive experiments with a multivariate elo-type algorithm," *Behavior Research Methods*, vol. 47, no. 2, pp. 384–394, Jun 2015. [Online]. Available: https://doi.org/10.3758/s13428-014-0478-7

[21] B. Project, "Computational thinking," aug 2014. [Online]. Available: http://www.computingatschool.org.uk

[22] J. M. Wing, "Computational thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, 2006.

[23] S. Joseph, "Programmer competency matrix," 2014. [Online]. Available: http://sijinjoseph.com/programmer-competency-matrix/

[24] R. 'kena' Poss, "How good are you at programming?—a cefr-like approach to mea- sure programming proficiency," jul 2014. [Online]. Available: http://science.raphael.poss.name/ programming-levels.html