Journal of the Brazilian Computer Society, 2022, 28:1, doi: 10.5753/jbcs.2022.2744

Model for evaluation of multiple abilities programming problems in online massive environments

Fabiana Zaffalon Ferreira [**Federal University of Rio Grande** | ***fabinhazaffalon@gmail.com***]
Ricardo Lemos de Souza [**Federal University of Rio Grande** | ***rcrdsou@hotmail.com***]
Andre Prisco Vargas [**Federal University of Rio Grande** | ***prisco.c3@gmail.com***]
Davi de Lemos Teixeira [**Federal University of Rio Grande** | ***daviltcomp@gmail.com***]
Michel Neves dos Santos [**Federal University of Rio Grande** | ***nevess.michel@gmail.com***]
Wanderson de Oliveira Paes [**Federal University of Rio Grande** | ***wandersonpaeswop3@gmail.com***]
Rafael Augusto Penna dos Santos [**Federal University of Rio Grande** | ***rapennas@gmail.com***]
Neilor Tonin [**Integrated Regional University of Alto Uruguai and Missões** | ***neilor@uricer.edu.br***]
Paulo Jefferson Dias de Oliveira Evald [**Federal University of Rio Grande** | ***paulo.evald@gmail.com***]
Silvia Silva da Costa Botelho [**Federal University of Rio Grande** | ***silviacb.botelho@gmail.com***]

Center for Computational Sciences (C3), Federal University of Rio Grande, Rodovia RS-734, s/n, Rio Grande, RS, 96203-900, Brazil.

Received: 21 June 2022 • **Accepted:** 06 December 2022 • **Published:** 30 December 2022

Abstract Research indicates human being is endowed with multiple intelligences, skills, and abilities. In the context of education, many exercises demand multiple skills from students to successfully solve them in different areas of knowledge. In computer science, computer programming is one of the skills that involves the use of multiple skills for problem-solving, where problems can be solved in more than one way (*paths*). On massive environments for teaching programming, it is common for automatic assessment systems to observe only the final result of the student's interaction with the learning object, not identifying the individual interaction of multiple skills needed to solve the problem nor identifying a *solution path* adopted by the student. Many models were proposed based on Elo models, which use performance expectation, and Item Response Theory, but these models do not consider the various *paths* to solve problems. The objective of this work is to propose a model also based on performance expectation, which individually estimates multiple abilities of students in the context of massive *online* education, assuming problems have more than one solution, and there is access only to the final result (right or wrong). An experimental *setup* is proposed to validate the model, involving the use and analysis of the proposed model through an experiment in a database, named beecrowd, and a case study with programming students. Model results are satisfactory, since: i) it is possible to treat the student's abilities individually, as well as to follow the evolution of each ability over time; ii) it is possible to predict the *paths* adopted by them according to the student's abilities; iii) the model also shows positive results when integrated with a recommendation system, recommending problems compatible with the student's abilities.

Keywords: Model, Abilities, Problems, Programming

1 Introduction

With an increasing number of massive online environments for education, there are several challenges associated with these new teaching and learning processes. Among them, it is emphasized the assessment methods, where it is sought after to infer and represent the construction of skills and abilities acquired by students, not through a punctual assessment; but through assessments that follow the student's progress over time.

There are systems named Online Judge, which are massive systems that evaluate the algorithm source codes, commonly utilized on programming competitions and known for the large repository of problems that can be used to teach programming skills [Wasik *et al*., 2018]. These systems are characterized as automatic assessment systems, whose verification is centered in the output of submitted the script, according to the input data [Giordano *et al*., 2021]. Both, the input and output data, are standard problem information; therefore, if the algorithm is correct, the output data is correctly generated, considering a known input [Galasso and Moreira, 2014]. After the assessment process, the students receive an automatic feedback, notifying them if the submitted script returned a correct or wrong output.

It is very important to assess student progress and provide them a meaningful feedback to support their learning process, which is traditionally done by a professor [Gerdes *et al*., 2010]. However, in very large classes, to provide a relevant feedback in a short time is hard, and error-prone, as it is difficult to maintain consistent and fair judgments for all students [Gerdes *et al*., 2010]. Therefore, the programming exercises are generally assessed by the resulting program, and the assessment criteria vary from professor to professor, as well as among universities. Some professors adopt holistic assessment and others detail analytical assessment criteria [Ala-Mutka, 2005]. In the holistic assessment, an algorithm or program can receive a grade or concept, which can consider the student approval even if its code did not work well due

to compilation errors, because the professor observes and assesses the solution development. The same algorithm would be reproved in the analytical criteria, because it would not be compiled [Ala-Mutka, 2005].

There are studies that analyze the use of Online Judge systems as a pedagogical strategy for teaching programming and discuss their characteristics [Giordano *et al*., 2021]. The advantage is the immediate feedback of these systems, allowing the students to update their programs and submit them again to the system [Giordano *et al*., 2021; Francisco *et al*., 2018]. Pedagogically, Online Judge systems can be a stimulus their to the traditional programming teaching methods, due to instantaneous feedback provided to the students, exactly when they need, allowing them to proceed at own pace and promoting self-learning [Francisco *et al*., 2018].

On the other hand, the system's feedback does not indicate the correct use of programming techniques and resources, nor does it indicate where and why the compilation error occurred [Giordano *et al*., 2021]. It is important highlight that due to adoption of this automatic assessment, the students can create the habit of solving problems by trial and error, based only in the system test, missing the learning inherent in developing your own tests [Francisco *et al*., 2018]. Although automatic assessment systems have evolved, they are still limited in assessing whether educational goals have indeed been achieved. Thus, the main challenge of automatic evaluation systems is to evaluate programming exercises as close as possible to a professor's assessment, since human evaluation allows a complex perception of different nuances involved in the design of a solution [de Oliveira and Oliveira, 2015].

The use of Online Judge systems to support programming teaching has been extensively explored. In this sense, some interesting works were developed, such as: *BOCA* [de Campos and Ferreira, 2004], *Codeforces* [1], *Timus Online Judge* [2], *CodeBench* [3], *SPOJ Brasil* [4], *Indian CodeChef* [5], *beecrowd* [6], among others. In this kind of system, one of the challenges is to estimate the multiple abilities of students, because these online platforms did not allow monitor the student's interaction with the learning object, returning as feedback only a message about correctness of the submitted program.

The *beecrowd*[7] [Bez *et al*., 2021], is widely used in competitions. However, it also provides exercises aimed at learning programming, being used as a didactic tool for worldwide professors. In this virtual environment, users select the problems to solve; submit their solution through a source code, one or more times until success or withdrawal, and receive automatic feedback.

Research shows that humans are endowed with various intellectual skills, indicating that intelligence is something broader [Smole, 1999a]. Besides, competence can be defined as the global or practical domain of an everyday situation, and skill as domain of a specific operation or actions that meet one or more competences [Perrenoud, 1999]. Therefore, in the educational area, skill is a practical application of a certain competence to solve a complex situation [França, 2020]. In the present work, competence is considered as the ability to elaborate a problem solution and skills are the specific knowledge applied to solve it. Furthermore, competence can be compounded of several skills; however, a skill does not belong only to a competency, since it can contribute to diverse competencies [Brasil, 2005].

It is highlighted that many practical exercises, in the different areas of knowledge, require more than one skill to be solved. Students in the computing area, in addition to the programming skill itself, referring to the particularities of the programming language, also need mathematical skills, logical reasoning, text interpretation, among others [Moreira *et al*., 2018; Robins, 2010]. There are several studies that point out the different skills involved in programming problems. The role of Computational Thinking as an important contribution to problem-solving has often been discussed nowadays, not only in the computing area; but also in general and propaedeutic disciplines. Naturally, in computing, some skills of Computational Thinking can be identified in solving algorithms [Barr and Stephenson, 2011]. In many cases, the same problem can be solved in several ways [Falckembach and Araujo, 2013], and each way to solve a problem is composed of a set of skills. In fact, the students have distinct abilities [Falckembach and Araujo, 2013], consequently finding out different solutions.

To estimate the student's abilities, there are models based on performance expectations, whose basic principle is to update the abilities according to the expected result after the subject's interaction with the object. If a student with high ability solves correctly an easy problem, the results will not be surprising and the skill update will be small; otherwise, the update will be bigger [Pelánek, 2016]. Among the models based on performance expectation, stands out: the Item Response Theory (IRT) [Baker, 2001] and the Elo System Classification [Elo, 1978].

The IRT is a set of mathematical models that seeks to represent the probability of an individual answering correctly an item, depending on the item's parameters, such as: difficulty, discrimination and hit by chance, as well as the student's ability [Baker, 2001; de Andrade *et al*., 2000]. The probability of an individual giving a correct answer to an item can be expressed as follows: the greater the individual's ability, the greater the probability of being correct [de Andrade *et al*., 2000]. There are IRT models that depend on the item nature, number of population involved, and amount of skills to be estimated [Baker, 2001; de Andrade *et al*., 2000]. There are also Multidimensional IRT (MIRT) models, which include items that require more than one skill from student [Baker, 2001; de Andrade *et al*., 2000]. MIRT models can be compensatory or non-compensatory. In a compensatory model, it is assumed that a low skill can be compensated by a higher skill. On the other hand, non-compensatory MIRT models assume that the student must have each of the skills at relevant levels to answer correctly an item [Park *et al*., 2019].

The Elo classification system [Elo, 1978], originally used in international chess ranking evaluations, classify players through their game history, using a statistical classification

[1] Link: https://codeforces.com
[2] Link: https://acm.timus.ru
[3] Link: http://codebench.icomp.ufam.edu.br/
[4] Link: https://br.spoj.com/
[5] https://www.codechef.com/
[6] Link: https://www.beecrowd.com.br/judge/
[7] Until 2021, the beecrowd identity was URI Online Judge.

that calculates the competitor abilities. In education, Elo system establishes that a student is considered a player and the problem is considered his opponent [Pelánek, 2016; Prisco *et al*., 2018]. Both, the Elo model and the IRT system, estimate a student's abilities, using different estimation procedures. Generally speaking, IRT assumes that student's skills are constants, while Elo system has been implemented to allow changes in their skills over time, considering a single skill [Pelánek, 2016].

From Elo model and IRT system, some interesting models for student skill ability estimation were proposed. The model M-ERS (Multidimensional extension of the Elo Rating System) [Park *et al*., 2019], which is the Multidimensional Extension of the Elo Rating System, uses the MIRT model to track the history of student's abilities and item difficulty, in a continuous way. This model assumes that a single item may involve more than one skill. In this way, the authors extended the standard Elo model, which updates only a single skill, to allow a simultaneous update of several skills based on a compensatory MIRT model. Other interesting approach is the Mixed Compensation Multidimensional IRT [Moissinac and Vempaty, 2020], which incorporates the two MIRT models, compensatory and non-compensatory. This model assumes that the item can have multiple compensating and non-compensating abilities simultaneously, and can only rely on a subset of abilities, rather than all of them.

In the context of massive online education, the adoption of skills assessment techniques, based on expectations and performance methods, may not be successful in the case of learning objects that have different solutions, and that involve different sets of skills. The fact of partially observing the subject's interaction with the object in relation to each required skill, depending on the way in which the student solved the problem, makes it difficult to analyze the effective performance in relation to the expectation predicted by the model. Therefore, in this sense, the objective of this work is to propose a model, based on performance expectation, that individually estimates the multiple abilities of students in the context of massive online education, where there are not access to the source code nor other characteristics of the solution, such as compilation time, execution time, memory occupation, source code size are observed. Thereby, considering that the problems can be solved in different ways, the model estimates the probability of success in each way, according to the student's abilities.

The remainder of this work is given as follows: Section 2 discusses the online judges, challenges of teaching and learning algorithms, as well as competencies and abilities. Next, in Section 3, the proposed model is presented, followed by model validation in Section 4. The results are presented in Section 5, and the conclusion is given is Section 6.

2 Online Judge

Massive environments are online platforms where thousands of students interact with thousands of learning objects [Vargas *et al*., 2019]. Therefore, they cannot be compared with traditional courses. For computing area, there are platforms aimed at evaluating source codes of algorithms in the most diverse programming languages. These platforms are called Online Judges. Their goal is to provide a secure, reliable and continuous assessment based on algorithms that are submitted by users distributed all over the world. Many universities offer these systems to help their students in their preparation for competitive programming championships [Wasik *et al*., 2018]. They are also used by professors, for creation of virtual classrooms, or simply by students who want to train and/or solve the programming problems available on these platforms [Bez *et al*., 2014]. As was previously discussed, Online Judge systems provide automatic feedback, which is immediate, allowing students to redo and correct their mistakes and resubmit their solutions for reassessment. [Giordano *et al*., 2021; Francisco *et al*., 2018].

The platform *beecrowd* [Bez *et al*., 2014], from Brazil, currently has students from more than 240 countries registered. It contains a base of programming problems classified into categories and levels of difficulty. Furthermore, it allows the students to choose the problem they want to solve first. In addition, this system has an automatic rating system. There is a template for each problem that is compared to the user program's output data, informing if the program is completely correct or if there is any error percentage. The user receives feedback and can rewrite the code and resubmit it. If there is an error in the code, this process can be repeated until the code is accepted.

The platform *beecrowd* has seven feedback types [Bez *et al*., 2014], which are:

- *Accepted*: the problem was accepted without any errors;
- *Closed*: there was a problem connecting to the server and the submission was not received;
- *Compilation error*: some code structure was written wrong;
- *Presentation error*: the output data is not formatted as described in the problem statement;
- *Runtime error*: error in the program execution flow;
- *Time limit exceeded*: execution time exceeded that stipulated in the problem statement.
- *Wrong answer*: program executed normally; however, it presented incorrect outputs.

The *beecrowd* integrates a tool, called Academic, intended for teachers to follow the practice, through the history of submissions, and the evolution of students. It allows teachers to have control over exercises, by creating lists with exercises into the platform Selivon *et al*. [2015]. Besides, teachers have access to the source codes of each submission, in addition to the percentage of errors when the student receives feedback, *Presentation error* or *Wrong answer*.

2.1 Challenges of teaching and learning algorithms

The mental process for learning algorithms often represents an obstacle for students beginning in the computing area [Falckembach and Araujo, 2013]. For many students, the challenge begins in the initial phase of learning, when it is necessary to understand and apply some abstract programming concepts [Gomes and Mendes, 2007; Gomes *et al*., 2008]. The challenges encountered in learning algorithms have been

studied for a long time. Such studies point out that the study methods adopted are not considered appropriate for learning, as programming requires practical and intensive study, and it involves a lot of understanding, reflection, and problem solving. Therefore, just attending classes and studying the content in books is not enough [Gomes and Mendes, 2007; Gomes *et al*., 2008]. One factor that can cause resistance to learning is the methodology adopted in teaching algorithms, which often ends up being the same for all students, as it is very difficult for the professor to provide personalized teaching to each student [Falckembach and Araujo, 2013].

One of the causes for the difficulties of many beginning programming students is the lack of generic problem-solving skills [Gomes and Mendes, 2007]. Students do not know how to create algorithms, mainly because they do not know how to solve problems. Problem solving requires multiple skills that students often lack [Gomes and Mendes, 2007]. Thereby, the evaluation of the student's progress associated with the feedback, usually provided by the professor, are pointed out as support for learning programming [Gerdes *et al*., 2010]. This would be the ideal scenario, but it is not always possible to do so, since in large classes, it is difficult to provide quick feedback. Furthermore, due to the short term and large classes, it is possible that there is a failure in the human evaluation [Gerdes *et al*., 2010].

Some professors adopt automatic assessment systems as support and assistance in their tasks [Gerdes *et al*., 2010]. The time required for the assessment is shorter and the professor can focus on the quality of the exercises and the teaching process [Wasik *et al*., 2018]. Other professors adopt holistic assessment, where an algorithm or program can receive a grade or concept considered good or approved, even if it fails in some analytical categories, such as compilation error, because the professor observes and evaluates the solution development [Ala-Mutka, 2005]. On the other hand, in an analytical observation, the same algorithm will reprove, because the program would not compile [Ala-Mutka, 2005].

2.2 Competencies and Abilities

The concept of competence began to be discussed in education in the 1990s. Due to the changes that have taken place in the educational environment, there is a dynamic based on the concepts of competence and ability [Cardoso and Hora, 2013]. Therefore, the National Curriculum Guidelines (NCG) and the National Curriculum Parameters (NCP) highlight the importance and need to no longer focus teaching and learning only on content; but on the development of skills and abilities, allowing the student to elaborate new concepts from the ones that have been [Cardoso and Hora, 2013]. It is highlighted that competence is problem solving ability itself; while skills are the use of resources to solve certain problems. A competency is made up of several skills, but a skill does not belong to a particular competency, since the same skill can contribute to different competencies [Brasil, 2005].

Believing there is no single and equal intelligence for everyone, Howard Gardner proposed in the 80s the theory of multiple intelligence based on the idea that people have different abilities to perform different activities and such activities require some kind of intelligence, but not necessarily the same [Gardner, 2011; Smole, 1999b]. In this sense, such concepts of skills and competences addressed in this work can be used worldwide.

2.2.1 Skills used in the computer programs development

Teaching programming aims to enable students to develop computerized systems capable of solving real-world problems [Pimentel *et al*., 2003; Gomes *et al*., 2008]. Therefore, computer programming competency is complex and context dependent, which requires multiple skills [Pea and Kurland, 1984].

There is a skill set involved that goes beyond just mastering the syntax of the programming language [Gomes *et al*., 2008]. Other factors are often mentioned as a pre-requisite for learning programming [Pea and Kurland, 1984], such as:

- Mathematical ability;
- Logical reasoning skills (a student may have basic knowledge and skills relevant to programming and not connect them to the programming domain, nor transfer knowledge acquired in programming to other domains);
- Conditional reasoning skills (working with conditional statements is an important part of programming, because they guide an operation of loops, tests, input checking, logical connectives, negation, conjunction and disjunction predicates, and other programming functions);
- Procedural reasoning skill (it involves the ability to construct a series of logically ordered instructions);
- Temporal reasoning skills (it involves the ability to execute subroutines or procedures, and understand when each should be executed, ensuring that a counter does not exceed a certain value until another operation is performed).

Gomes and Mendes (2007) assume that students do not know how to solve problems and understand that one of the causes is the multiple skills required to solve them. In In the work, the authors point out:

- Understanding the problem: students often try to solve a problem without fully understanding it;
- Relate knowledge: establish analogies with problems already solved, and transfer knowledge to new problems. Many students base their solutions on unrelated problems, leading to incorrect solutions;
- Lack of persistence: Students tend to give up on solving problems if they do not solve it quickly;
- Mathematical and logical knowledge: many students have difficulties or little mathematical knowledge, in addition to difficulties in transforming a textual problem into a mathematical problem, not identifying the formulas that solve it.

To develop an algorithm, it is necessary to interpret the information contained in the problem statement and have specific knowledge to plan the solution strategy. Naturally, the same problem can be solved in several ways that involve one

or more strategies. Strategies involve skills such as reasoning and deductive logical thinking; besides, allowing them to exercise their reasoning gradually [Falckembach and Araujo, 2013].

2.2.2 Computational Thinking

The role of Computational Thinking (CT) has been discussed and applied in all disciplines, signaling how computer science can contribute to solving problems [Wing, 2006]. Computational Thinking is the process of formulating and solving a problem using the fundamental concepts of computer science and can help in solving problems in the most diverse areas [Wing, 2006]. The CT proposal is to apply the skills used in the development of computer programs as a methodology for solving general problems. There is no consensus in the literature on CT skills; however, there are 3 pillars that underlie the CT [Wing, 2006], they are: abstraction (problem formulation), automation (solution expression) and analysis (solution execution and evaluation).

According to these pillars, the concepts of CT were organized in a structured model, with the purpose of identifying the main CT concepts and providing examples of how they can be incorporated into activities in various disciplines. From this study, CT elements related to computer science were identified, and it can be also identified in the source codes [Barr and Stephenson, 2011; de Souza *et al*., 2020]. In this study, the following skills were selected:

- Data Analysis: write a program to solve basic statistical calculations;
- Data Representation: use data structures in vectors, lists, queues, tables, and so on;
- Abstraction: use repetition structures, conditionals, recursion, and others. In addition to using procedures to encapsulate a set of data that will be repeated for the same purpose.

These skills are easier to identify even when you do not have access to the source code, which is why they were selected. The Problem Decomposition skill is difficult to identify because a problem can be solved with or without the use of functions.

2.3 Item Response Theory

The IRT is an approach that has gradually been inserted into education, because it is considered an important instrument in the evaluation process. This model allows the measurement of individual characteristics, which are difficult to measure directly [Baker, 2001; de Andrade *et al*., 2000]. For skill estimation, θ, the IRT is based on statistical methods and mathematical models that consider individual responses and items properties [Baker, 2001; de Andrade *et al*., 2000]. Therefore, the greater the student's skill, the greater will be the probability of item success [Baker, 2001].

There are several IRT models, which depend on various factors [Baker, 2001; de Andrade *et al*., 2000]. For dichotomous items, there are 3 models that differ from each other by the nature of the parameters observed to describe the item [Baker, 2001; de Andrade *et al*., 2000]. Are they:

- 1-Parameter Logistic Model or Rasch Model (only observes item difficulty);
- 2-Parameter Logistic Model (observes difficulty and item discrimination);
- 3-Parameter Logistic Model (observes the difficulty, item discrimination, and the probability of a random hit).

These models consider the test to be a one-dimensional instrument that implies the existence or predominance of only one skill, which does not apply in many practical situations in which many exercises, in different areas of knowledge, require more than one skill to be solved. As an example, we can cite a mathematical test that requires text interpretation before requiring mathematical development, and, in this case, it is a two-dimensional test, because it requires two skills [Nojosa, 2002]. Research has shown that MIRT is better adapted to real assessment situations than the one-dimensional models, because in the teaching-learning process, subject's responses are determined by more than one skill [Pasquali, 2018].

As aforementioned, MIRT models can be separated into two classes: compensatory and non-compensatory. The non-compensatory MIRT model assumes that dimensions of knowledge are independent of each other, and compensation is not possible. Thus, it is assumed that the student must have each of the relevant skills to correctly answer an item. In this model, each dimension's logistic function is treated as an independent probability [Reckase, 2006], as follows,

$$P_{ij} = P(Y_{ij} = 1) = \prod_{m=1}^{M} \frac{e^{(\alpha_{jm}(\theta_{im} - \beta_j))}}{1 + e^{(\alpha_{jm}(\theta_{im} - \beta_j))'}}, \quad (1)$$

where Y_{ij} is the individual's response i to the item j; $P(Y_{ij} = 1)$ is the correct answer probability; α_{jm} is the item discrimination parameter j in the dimension m; θ_{im} is the individual's ability i in the dimension m and β_j is a scalar that indicates the difficulty of the item j.

A model is said to be compensatory when the probability of item success is maintained or increased even if one of the skills is low, being this one compensated by another, higher skill [Nojosa, 2002]. The compensatory model [Reckase, 2006] is represented by

$$P_{ij} = P(Y_{ij} = 1) = \frac{e^{(\sum_{m=1}^{M} \alpha_{jm}(\theta_{im} - \beta_j))}}{1 + e^{(\sum_{m=1}^{M} \alpha_{jm}(\theta_{im} - \beta_j))'}}. \quad (2)$$

2.4 Elo Rating System

Initially, the Elo classification system was proposed to analyze and rank the performance of chess players [Elo, 1978]. Elo is a real scalar value that represents a player's skill level. Through statistical methods, each player receives an initial Elo θ_i and, as you participate in the games, Elo is updated according to the results.

Elo model is based on expectation and outcome; the expected probability that the player will win the match is given by the logistic function with respect to the difference in the

estimated ratings [Pelánek, 2016]. It can be formulated as follows,

$$P(R_{ij} = 1) = \frac{1}{1 + 10^{\frac{\theta_j - \theta_i}{400}}} \tag{3}$$

where $R = \{0,1\}$ is the result set of a game: 1 (win) and 0 (lose). Given a match between the players i and j, with Elo θ_i and θ_j, respectively. After the game ends, new Elos are calculated according to the results expectations, the previous Elos and a constant k. The higher the k, the greater the Elo change [Pelánek, 2016]. This relation is given by

$$\theta_i = \theta_i + k(R_{ij} - P(R_{ij} = 1)) \tag{4}$$

When applied in education, Elo establishes that a student is considered a player and the problem is considered his opponent [Pelánek, 2016]. Thereby, updates of student skills and problem difficulties happen on a continuous basis, at the end of each resolution [Pelánek, 2016]. The difference between Elo applications when used in education, if compared to its employment in games, lies in the asymmetry between students and education items. For most problems (or learning objects), the difficulty is expected to be approximately constant. On the other hand, changes in student skills are expected, which is the goal of educational systems [Pelánek, 2016].

In educational applications, there is also a discrepancy between correct and incorrect answers. The response to an item is not only evidence of the student's knowledge, but also an opportunity for learning. Therefore, it is necessary to run different updates for correct and incorrect answers [Pelánek, 2016]. It can be performed using different constants for correct and incorrect answers, instead of using a single constant K.

The use of the Elo model offers relevant advantages, such as: simplicity for using it in the online environments and implementation in educational systems, in addition to presenting a low number of parameters to adjust [Pelánek, 2016]. However, as a restriction, it is intended to track only a single skill [Park *et al*., 2019].

2.4.1 Multidimensional Elo

In the classical model, Elo is a scalar value for each student and for each learning object (item/problem). The extended model of the Elo metric, in order to make it multidimensional, considers that each dimension is a skill that the student must have at some level [Prisco *et al*., 2018]. The student's skills are represented by

$$\overrightarrow{\theta_s} = (S_1, S_2, ..., S_N) \tag{5}$$

where S_i is the Elo in the skill i.

Each L learning object has its demand level represented by $\overrightarrow{\sigma_l}$ and by relevancy $\overrightarrow{m_l}$. The set of learning objects can be described by

$$L = \{(\overrightarrow{\sigma_1}, \overrightarrow{m_1}), (\overrightarrow{\sigma_2}, \overrightarrow{m_2}), ..., (\overrightarrow{\sigma_n}, \overrightarrow{m_n})\} \tag{6}$$

where n is the total number of learning objects; σ_l is the demand level, relevance m is a real number, between 0 and 1, that indicates how important a skill is for interaction with a learning object. Considering student interaction i with learning object j, the adaptation to the model, for each skill s, is given by

$$\begin{aligned} \theta_{i_s} &= \theta_{i_s} + m_{j_s} k(R_{ij} - P(R_{ij} = 1)) \\ \sigma_{j_s} &= \sigma_{j_s} + m_{j_s} k(R_{ji} - P(R_{ji} = 1)). \end{aligned} \tag{7}$$

Each interaction is a tuple $I = (S, L, R)$, where $R = \{0, 1\}$, with 1 indicating that the student got the problem right or 0 indicating that the student got it wrong.

2.5 Model M-ERS

The model M-ERS presents an approach that incorporates the compensatory MIRT model into the Elo model, to track estimates of ability parameters [Park *et al*., 2019]. Rather than assuming a one-dimensional trace of item responses, the approach assumes that a single item may involve more than one skill. In this way, the authors extended the standard Elo, which updates, in an evolutionary way, a single skill, allowing a simultaneous update of m distinct skills.

The difference between the observed performance Y_{ij} and the expected performance P_{ij}, based on the compensatory MIRT model, shown in (8), is used on (9) to update skill parameters after each item response.

$$P_{ij} = P(Y_{ij} = 1) = \frac{e^{\left(\sum_{m=1}^{M} \alpha_{jm}\theta_{im} - \beta_j\right)}}{1 + e^{\left(\sum_{m=1}^{M} \alpha_{jm}\theta_{im} - \beta_j\right)}}, \tag{8}$$

$$\begin{aligned} \hat{\theta}_{im(t)} &= \hat{\theta}_{im(t-1)} + D_{m(t)} K\{Y_{ij(t)} - P_{ij(t)}\} \\ \hat{\beta}_{j(t)} &= \hat{\beta}_{j(t-1)} - D_{m(t)} K\{Y_{ij(t)} - P_{ij(t)}\} \end{aligned} \tag{9}$$

where being $D_{m(t)}$ is a weight to specify whether the skill m is indicated by the item given in the t-th step. For the skill present in the item, $D_{m(t)}$ assumes a value equal to 1. Otherwise, the weight assumes values between 0 and 1. Moreover, K decreases linearly, between 0.4 and 0.1, as a function of the total number of items answered.

2.6 Mixed Compensation Multidimensional Item Response Theory

Mixed Compensation Multidimensional IRT (MCMIRT) incorporates the two MIRT models, the compensatory and the non-compensatory [Moissinac and Vempaty, 2020]. The authors understand that adopting only one of the MIRT models ends up restricting its use, as the items can have compensatory and non-compensatory dimensions, simultaneously. Also, items may depend on only a subset of dimensions, rather than all.

MCMIRT depends on a graph representing the compensatory and non-compensatory dimensions of knowledge and their relationships. Each graph node is a knowledge dimension, and each undirected edge represents the relationship between the two knowledge dimensions. Therefore, given an item i, a subgraph is formed when several dimensions of

knowledge are covered and compensated for each other. Figure 1 shows an item i that covers the dimensions B, C, D, and E. Dimensions D and E are compensatory, and constitute the subgraph S_3. In addition, dimensions B and C are not compensatory to each other. Therefore, they constitute two subgraphs S_1 and S_2 [Moissinac and Vempaty, 2020].

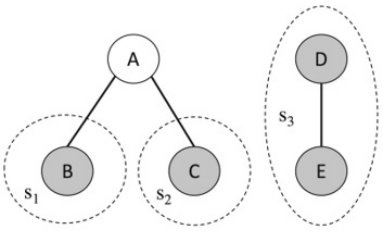

Figure 1. An example of a knowledge graph

This model, assumes that there is compensation within the subgraph, and a non-compensation between the subgraphs. Thus, the probability of success is estimated as

$$P(u_i = 1|\theta, \Delta_i) = c_i + (1 - c_i) \underbrace{\prod_{s_m \in S_i} \underbrace{\frac{e^{\sum_k a_{i,k}(\theta_k - b_{i,k}) I^k_{i,s_m}}}{1 + e^{\sum_k a_{i,k}(\theta_k - b_{i,k}) I^k_{i,s_m}}}}_{compensatory}}_{no-compensatory} \quad (10)$$

where θ is the skill vector; $\Delta_i = \{a_i, b_i, c_i\}$, a_i and b_i are vectors of discrimination and difficulty of item i for each dimension, c_i is the random hit; $S_i = \{s_1, ..., s_m\}$ is the set of subgraphs created by coverage of item i; I_{i,s_m} is a vector indicating the dimension number k (if the knowledge dimension does not belong to the subgraph, the indicator is 1; otherwise it is 0).

Although the literature presents relevant reviews in the evaluation of algorithms and skills involved in computer programming, skills assessment models still face challenges in the computing area. Such models do not consider that programming problems can be solved in more than one way. As each student has their skills that can be developed more than another student; therefore, each one can solve a problem according to their skills, achieving the correct response in distinct forms.

3 The proposed model

For the proposed model, SMAS (Student's Multiple Abilities and Skills), the following assumption is assumed: programming problems have more than one way to be solved, called *paths*, and each of the paths has a set of skills necessary to solve it correctly.

3.1 Representation of problem solving paths

The skills belonging to each set are not compensatory among themselves; that is, it is necessary for the student to have all the skills according to the problem relevance. The *paths* are mutually compensatory, since the student will choose a certain *path* to solve the problem, according to their abilities. Thus, even if the student has some lower skills, if the *path* chosen does not require such skills, it is possible to succeed in the solution.

The paths and skill relationships of SMAS, are represented by graphs, as shown in Figure 2. Each graph node is a skill, and each undirected edge represents the skill relationship. Given a problem j, a subgraph represents a *path* (s_n) of solution in which the skills are not compensatory; that is, they are all necessary for that *path*. Model SMAS, assumes there is compensation between subgraphs and no compensation within the subgraph.

Figure 2 presents an example of a problem j which can be solved using 3 skills, A, B and C. To solve the problem there are 2 possible *paths* (s_1 and s_2): for *path* s_1 are required abilities A and B, and for *path* s_2 are required abilities B and C. The student will solve the problem using either the *path* s_1 or s_2, according to abilities which has. The D skill is not needed to solve this problem.

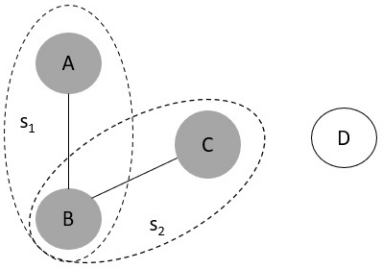

Figure 2. Example problem with two paths to solution

Thereby, each problem j has a set of *paths* to the solution, $j = \{s_1, s_2, .., s_n\}$, each set s is composed by n skills that have difficulty β and relevance α, $s = \{(\beta_1, \alpha_1), (\beta_2, \alpha_2), ..., (\beta_n, \alpha_n)\}$. Relevance is how important a given skill is for the correct resolution of the problem, given by a real value between 0 and 1. Naturally, a relevance with a value equal to 0 means that the skill is not needed to solve that specific problem.

As all skills are needed on each *path* s, for each *path*, it is calculated the probability of success, through the adapted non-compensatory MIRT, as follows,

$$P_{is} = \prod_{n \in s}^{N} \frac{e^{(\alpha_{ns}(\theta_{in} - \beta_{ns}))}}{1 + e^{(\alpha_{ns}(\theta_{in} - \beta_{ns}))'}}, \quad (11)$$

where P_{is} is the probability of student i to be successful on problem solving with *path* s; α_{ns} is the relevance of skill n in *path* s; β_{ns} is the difficulty of skill n on *path* s and θ_{in} is the n skill of student i .

3.2 Path taken to the solution

With the probability of each *path*, it is assumed that the student will choose that *path* that has the highest probability of being correct. According to Rossler [Rossler, 2004], the task solution by individuals, based on spontaneity, tends to follow the law of least effort, which requires less consumption of energy, time and thought. Moreover, according to Azevedo and Formiga [Azevedo and Formiga, 2013], doing the most with the least effort is the basic law of nature that permeates the

universe. Thus, the probability of the question being correct assumes the highest probability among the *paths* ($P_{ij} = P_{is}$), as well as problem parameters: difficulties and relevance of *path* skills most likely.

3.3 Model update

Updates of student skills and problem difficulties occur simultaneously according to the solution result: correct or incorrect answers Pelánek [2016]. It is known that in massive systems there can be many interactions of students until they succeed in solving the problem. In addition, problems with many incorrect submissions can lead to a significant increase in the difficulty of the problems, which may not match the real difficulty level of such problem.

The difference between the observed performance Y_{ij} and the expected performance P_{ij} is used to update skill parameters after each item response [Prisco *et al.*, 2018]. Therefore, n-th skill of individual i and problem j are updated according to

$$\begin{aligned} \theta_{in(t)} &= \theta_{in(t-1)} + \alpha_{jn} K\{(R_{ij} = 1) - P_{ij}\} \\ \beta_{jn(t)} &= \beta_{jn(t-1)} + \alpha_{jn} K_p\{(R_{ji} = 0) - P_{ji}\} \end{aligned} \quad (12)$$

where β_{jn} and α_{jn} are the difficulty and relevance of the skill n of the problem j, both of the *path* with the highest probability of success, respectively; K is a constant equal to 32[8], which defines how much the estimate can be affected by the difference between the current and the expected response; Y_{ij} is student i's answer to the problem j: 1 for correct or 0 for error.

When updating the problem's difficulties, the variable K is replaced by K_p (a proportional K) [Prisco *et al.*, 2018], described as

$$K_p = \frac{32}{10}\alpha_{jn}. \quad (13)$$

This replacement avoids excessively high updates in the problem parameters. According to Pelánek [2016], it is not expected many variations in the difficulty of learning objects; on contrary to student skills. Figure 3 shows an overview of the methodology for applying the model SMAS.

4 Model Validation

An experiment and a case study were performed to validate the proposed SMAS model. The experiment was carried out using the database provided by the platform *beecrowd*. In this database, the SMAS and Elo models were applied, and, for each model, a simulation of problem recommendations was made based on the students abilities. The purpose of this test was to analyze the evolution of the student's skills in each model, and to observe the recommendations made by the models. The comparison of the SMAS model could be made with any model studied in this work. However, we chose to compare with Elo, as it is a consolidated model in the literature.

[8]empirically adopted value as presented in [Pelánek, 2016]

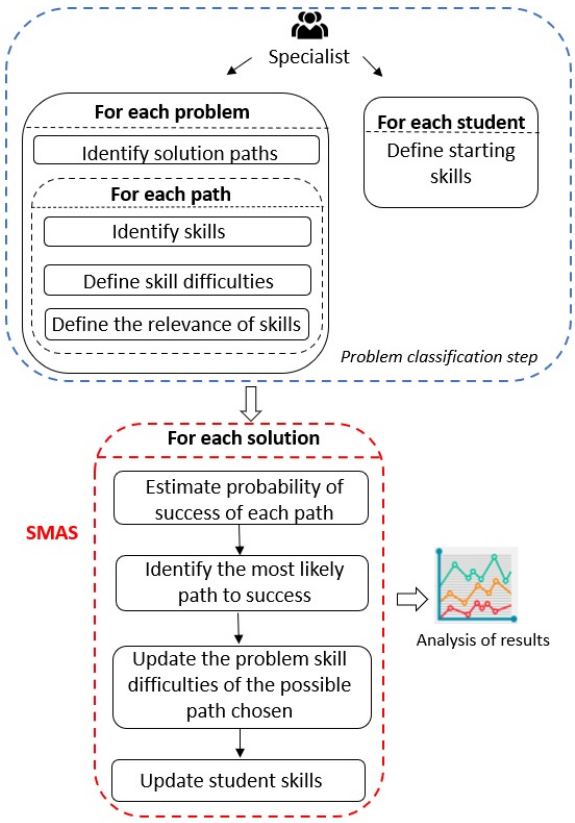

Figure 3. Application of model

Subsequently, a case study was carried out in Computer Science area classes, where students solved problems of the *beecrowd* platform in virtual rooms, created by the professor. The objective of this case study was to verify if the paths chosen by the students to solve the problems were the same paths indicated by the SMAS model.

4.1 Repository Problem Analysis

An expert analyzed a set of 200 problems from the *beecrowd* platform [Paes, 2022]. Each problem was analyzed in order to identify all possible *paths* to solve it. In each *path*, the skills involved were associated and each skill received a difficulty value and a relevance value, which varies between 0 and 1. Information about adopted methodology for problem analysis can be found in [Paes, 2022].

4.2 Experiment using the database

The objective was to observe the skill performance of each user who submitted solutions to the platform's problems. The proposed model was applied to estimate the multiple abilities of users. Moreover, a recommendation system was simulated in the base, where, at each submission, a set of problems compatible with the user's abilities was generated.

The data provided by *beecrowd* platform are: *submission date and time, user id, problem id* and *answer (1 - correct or 0 - wrong)*. The base consisted of 1.162 programming problems, 62.976 users, and 1.048.576 submissions. In this database, a filter was applied, selecting only the submissions made to the problems that were analyzed by the specialist. In this way, the database was composed of 753.113 submissions, 200 problems, and 61.240 users.

Figure 4 shows the procedure to perform the experiment. Data from students, problems and submissions were collected and stored in a database, in chronological order of submission, in which each row was considered a user interaction with the problem.

For each problem, the *paths* for solution were defined and, for each *path*, the skills were identified, as well as its difficulty and relevance values. Besides, initial values of 1100 were assigned to each student skill. It was necessary due to cold start problem, which occurs because the system does not know the skill level of a new student, on their first interaction (submission). It is highlighted that this value can be randomly defined, as it will be adapted by the model as students submit their solutions [Vargas *et al*., 2019].

The SMAS model estimates the success probabilities of each identified *path*, through the equation (11), and identifies the *path* with the highest chance of success. Next, it simulates each submission made in chronological order through the response of each submission. After the estimates, the new values of the user's skill and the problem difficulties, through in the equation (12), are stored in the database. In this way, there is a history of the skills of each user and the difficulty of each problem.

A simulation of a recommendation system was made, to observe the user's self-recommendation, since the database provided is of users who chose themselves the problems to solve. In this simulation, the recommendation zone was created, where all problems compatible with the user's abilities were inserted. Problems into this zone cannot be too difficult (low probability of success), nor too easy (high probability of success) [Vargas *et al*., 2019]. Therefore, problems with a probability between 40% and 60% of success chance were included in the recommendation zone. Due to the cold start, simulations of the recommendations were made from the 31th submission of each user, to calibrate the student's skill values and problem difficulties.

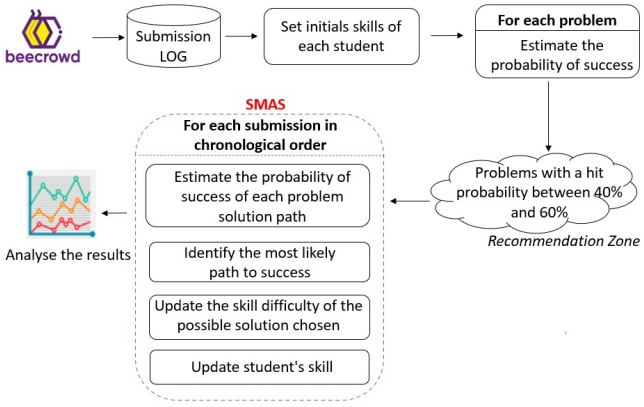

Figure 4. Methodology of the experiment

A second experiment was carried out using the same database, applying the Elo model, where the objective of it was to compare the results generated by the two models, through the students abilities. In this experiment, there was also a simulation of a recommendation system, starting from the 31th submission of each user, where the problems compatible with the user's Elo were inserted. The parameters were the same in both experiments: User Elos were initialised at 1100, and problems with success probability between 40% and 60% were included in the recommendation zone.

4.3 Case Study: Validation of *Paths*

The objective of the case study is to validate the SMAS model in relation to the *paths* of solving the problems. On the *beecrowd* platform, virtual classrooms were created for students enrolled in programming courses. In total, 54 students participated in the case study, including 10 students from higher education and 44 technical education students.

The classe's Professors selected 42 problems made available by the *beecrowd* platform. Of these problems, in 36 there were two possible *paths* of solution; and in 6 problems there were three *paths* of solution. Most problems are categorized as Beginner, as most students are at the beginning of the course. Even so, some problems considered more complex were included, which involve knowledge of arrays and data structures.

Students registered on the platform and submitted solutions to the proposed problems within 30 days. There were 2,613 submissions made in 42 problems. After the deadline, the data of the submissions of the classes were tabulated in chronological order of submission. The data are: date and time of submission, problem id, answer (hit or miss). With this information, the SMAS model was applied.

Each source code was analyzed by an expert who identified the *path* adopted to solve the problems. After this analysis, a comparison was made between the *paths* chosen by the students and the *paths* indicated by the SMAS model, with the purpose of verifying whether through the model it is possible to predict the *paths* adopted by the students.

5 Results

5.1 Experiment using the database

After applying the SMAS and Elo models in the same database, the performance and choices of the users who showed the greatest abilities were observed. These users were taken as a reference for the analysis of problem choices versus problems contained in the recommendation zones, as it is understood that users with better performance (higher success rate in solving the chosen problems) have a greater understanding of their abilities, which enables them to choose challenging problems that develop their skills, which is why they are considered good self-recommenders.

The experimental validation of the model was carried out by comparing the problems solved by users with greater skills versus the recommended problems, according to the skills estimated by the SMAS and Elo models. In both models, the same heuristic was applied: recommend problems with a probability between 40% and 60% of the student being successful in their solution.

From 19,866 submissions using the SMAS model, 33% (6,479) of the problems solved by students were in the recommendation zone. Considering the Elo model, only 20%

(3,962) of the submissions were related to problems in the recommendation zone.

In the submissions made by users, there are many solutions without success, which are the cases in which users submit their solutions and the platform gives an error feedback; then, users continue resubmitting solutions until they receive a positive feedback from the platform. Disregarding these unsuccessful attempts, in 46% of the recommendations simulated by SMAS, users solved problems that would be recommended, and were successful in the solution. In the simulation with the Elo model, only 26% of success was obtained in the responses.

Among users who gained skills, the average percentage of correct answers for problems in the recommendation zone was: 71.40% for the SMAS model, and 50.15% for the Elo model. Comparing the recommendations simulated by the models, the SMAS achieved a better representation of the user's abilities, as it came closer to the self-recommendations made by the users.

The skill evolution graphs of the two groups of users were analyzed. The first group presented better performances and greater abilities in both models. On the other hand, the second group showed lower performance in the evolution of skills. Due to the large number of users in the database, we selected 3 users in each group, to analyze their evolution in detail.

First group: in the SMAS model, these users had values above 1300 in the 3 skills (Data Analysis, Data Representation and Abstraction), and Elo values varied between 1264 and 1300, according to Table 1. This table presents the values of the Data Analysis (Analysis column), Data Representation (Representation column) and Abstraction (Abstraction column) skills, while the Elo column presents the user's Elo value.

Table 1. Abilities and Elo of the users of the first group

User	Analysis	Representation	Abstraction	Elo
1	1477	1316	1439	1264
2	1424	1414	1456	1494
3	1408	1392	1442	1300

In addition, Table 2 presents the submission data from students of the first group. In the column Submissions, the valid number of submissions from each user are shown. However, the first 30 submissions were used to calibrate user's skill and Elo values; therefore, they were disregarded to simulate recommendations. Besides, the Success and Errors columns refer to the number of successful and unsuccessful submissions, respectively.

Table 2. Data from user submissions in the first group

User	Submissions	Success	Errors
1	178	146	32
2	77	75	2
3	116	95	21

Complementary, Table 3 shows information about problems recommended by models. In the columns SMAS and Elo, there are the number of problems in the recommendation zone that the user solved. Furthermore, the SMAS Success and Elo Success columns contain the number of problems that were present in the model's recommendation zone and that were successfully solved by users.

Table 3. Data from simulated recommendations to users in the first group

User	SMAS	SMAS Success	Elo	Elo Success
1	40	36	1	1
2	23	22	0	0
3	43	35	1	1

From data analysis, it was observed that, for these users, the SMAS model was closer to their choices than the Elo model, since the number of problems contained in the SMAS recommendation zone that were solved by the platform users is higher than the Elo model. Also, users solved more than 80% of problems in the recommendation zone correctly. Moreover, users who presented the reduced performance (second group) were also analyzed; that is, those who were unsuccessful in their choices and, consequently, decreased their skills and Elo, as shown in Table 4.

Table 4. Abilities of the users of the second group

User	Analysis	Representation	Abstraction	Elo
4	562	596	596	823
5	568	681	733	782
6	648	728	749	895

Table 5 shows information regarding the submissions of the users of the second group; and Table 6 presents the recommendations simulated by the models. As in the first group, the first 30 submissions were used for skills calibration.

Table 5. Submissions data from users in the second group

User	Submissions	Success	Errors
4	132	18	114
5	122	9	113
6	188	39	149

Table 6. Submissions data and simulated recommendations to users of second group

User	SMAS	SMAS Success	Elo	Elo Success
4	5	2	68	10
5	5	2	6	2
6	3	2	105	22

The behavior of the second group users was to choose problems that, in most submissions, were not present in the recommendation zone of the SMAS model, where they achieved success in more than 40%. In the Elo model, there were more problems solved that were present in the recommendation zone, but with a low hit rate, on average 22%. Although both models are based on performance expectations, the SMAS model estimates each subject's ability according to the skills required to solve the problems, considering several *paths* to solve them, while the Elo model considers only one skill. It is believed that this difference between the models is responsible for the discrepancy in relation to the observed results.

Figure 5 shows the skill evolution of the user 1, according to the SMAS model; and the Figure 6 presents the skill evolution considering the Elo model.

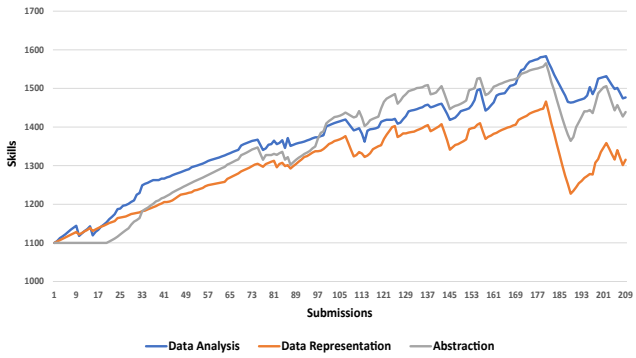

Figure 5. Skill evolution of the user 1 by the SMAS Model

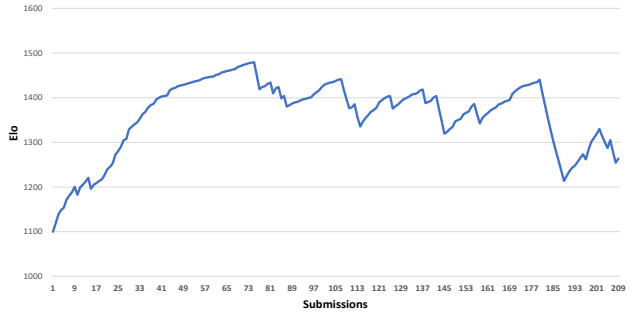

Figure 6. Skill evolution of the user 1 by the ELO Model

According to Figure 5, it is possible to observe the evolution of student 1's individual skills; and thus, identify which skills require more attention, as well as the most developed abilities. Note that all skills have increased their values, but it can be said that the lowest skill is *Data Representation*. In the Elo model, shown in Figure 6, only the evolution of the single skill is visualized. Thus, in the Elo model, it is possible to know how the student's overall learning progress is; but, it is not possible to infer whether any skill is lower than another, given the limitations of the model.

Individual skills information is important to help teachers recognize students abilities; then, they can guide them in providing study materials tailored to each student's needs. This information is also important for educational systems that recommend learning objects, or that perform automatic assessment of multi-skill exercises.

5.2 Case Study

Students submitted 2,613 solutions to 42 programming problems registered in a virtual room on the *beecrowd* platform, where 569 submissions were correct.

Among all student solutions, in 1,700 submissions, the SMAS model predicted the path chosen by the students. In other words, the approximate success rate was 65.05%. Considering only successful submissions, among the 569 correct solutions, the model had an accuracy rate of approximately 69.42%. The graphics of the Figures 7 and 8 present these data.

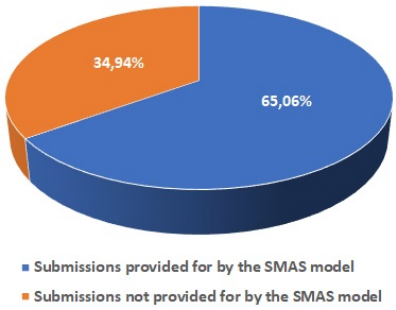

Figure 7. SMAS model success and error rates

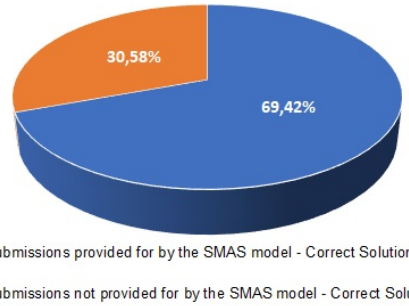

Figure 8. SMAS model success and error rates with success

Although the numbers are not considered very high, the SMAS model modeled the student's abilities in order to identify, in most submissions, the *path* that the students chose to solve the problems. The model does not set a path indication, since at each submission the values of the student's abilities and problem difficulties vary according to the submitted solution. Therefore, at each submission, new estimations are performed by the model. It was noticed, through the analysis of the source codes, that most students, when adopting a certain *path*, follow the same strategy until they get the solution right.

There were 393 cases in which students submitted more than one solution to the same problem, until they were successful in the solution, or giving up the problem. Furthermore, in 320 cases, the students adopted the same solution path; that is, they did not try to change their strategy to solve the problem. For model indicating adequately the problems, it is important that the problems are carefully analyzed by specialists during the attribution of the relevance and difficulties of each problem, because the model is based on these data.

6 Conclusion

This paper presented a model that estimates the multiple abilities of students who solve programming exercises. This model assumes that: programming problems have more than one way (*paths*) to be solved; each path involves multiple skills, and students solve the problems whose path is most likely to be correct. The proposed model was applied to a database provided by the *beecrowd* platform, and compared with the Elo model, where was simulated the recommendation of programming problems, registered on the platform, according to the estimated skills. The objective of this comparison was to observe the evolution of the student's skills, as well as to verify the behavior of both models when faced with

the recommendation of learning objects. Between the models, the SMAS simulated the recommendation of a greater number of problems that students chose on the platform. Among the problems solved by users, the SMAS model recommender suggested more problems chosen by users in the recommendation zone. It is believed that the SMAS model came closer to the choices made by users, because the model estimated the possible way to solve the problems taking into account the individual skills of each user involved in the problem solution.

Regarding ability evolution, the proposed SMAS model allows to identify the progress of the three abilities of the student: analysis, representation and abstraction. Such identification supports teachers and education systems in improving material suggestions that reinforce student weak skills. In the Elo model, this detailed analysis is not possible, because it considers only one progress marker, related to overall abilities. Furthermore, the SMAS model considers the different ways to solve programming problems, and identifies the one most likely for each user, according to their abilities. With the case study carried out, it was possible to observe that the model pointed out, in most submissions, the paths chosen to solve the problems. The success rate obtained was satisfactory.

The following contributions of the proposed model stand out:

- Identification and modeling of multiple student's abilities;
- Monitoring the evolution of student's abilities;
- Considering different ways (paths) to solve programming problems;
- Identification of the abilities involved in each problem-solving path;
- Prediction of the paths adopted by students in light of their abilities.

Results shown that among users who gained skills, the average percentage of correct answers for problems in the recommendation zone was: 71.40% for the SMAS model, and 50.15% for the Elo model. Therefore, the multiple skills model, proposed in this work, contributes positively to the personalization of programming teaching, since it provides a possibility of observing the evolution of each student skill and, thus, indicating more favorable exercises to develop their weaker skills.

However, it is important to pay special attention to problem annotations (identification of *paths* and skills, attribution of difficulty, and relevance) for the model achieving better results, since SMAS model uses this information, given the student's abilities to indicate the most likely *path* and estimate abilities. If these data are not accurately identified, skill estimation and suggestion of the *paths* may not give satisfactory results.

Declarations

Acknowledgements

The authors would like to thank the beecrowd plataform for providing access to its repository, as well as the Graduate Program in Science Education from Federal University of Rio Grande (FURG) and Instituto Federal de Educação Ciência e Tecnologia Sul-rio-grandense (IFSUL).

Authors' Contributions

All authors contributed to the final manuscript.

Competing interests

The authors declare that they have no competing interests.

References

Ala-Mutka, K. M. (2005). A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102. DOI: 10.1080/08993400500150747.

Azevedo, D. T. C. d. and Formiga, V. R. A. (2013). As competências da pedagogia: um estudo de caso na escola municipal de ensino fundamental Centenário Presidente João Pessoa. Undergraduate Project (Pedagogy), Centro de Educação, Universidade Federal da Paraíba. Paraíba. p. 48.

Baker, F. (2001). *The basics of Item Response Theory*. ERIC, Washington, USA.

Barr, V. and Stephenson, C. (2011). Bringing Computational Thinking to K-12: What is Involved and What is the Role of the Computer Science Education Community? *Acm Inroads*, 2(1):48–54. DOI: 10.1145/1929887.1929905.

Bez, J. L., Tonin, N., and Rodegheri, P. (2014). URI Online Judge Academic: A tool for algorithms and programming classes. In *2014 9th International Conference on Computer Science Education*, pages 149–152. DOI: 10.1109/ICCSE.2014.6926445.

Bez, J. L., Tonin, N., and Rodegheri, P. (2021). beecrowd Judge. Available in: https://www.beecrowd.com.br/judge. Access date: 22/03/2022.

Brasil (2005). Exame Nacional do Ensino Médio (ENEM): Fundamentação Teórico Metodológica. *Ministério da Educação. Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira*. Brasília/DF.

Cardoso, M. and Hora, D. M. (2013). Competências e habilidades: alguns desafios para a formação de professores. *Jornada do HISTEDBR*, v. 11.

de Andrade, D. F., Tavares, H. R., and da Cunha Valle, R. (2000). *Teoria da Resposta ao Item: conceitos e aplicações*. ABE - Associação Brasileira de Estatística, São Paulo.

de Campos, C. P. and Ferreira, C. E. (2004). Boca: um sistema de apoio a competiçãões de programaçãõ. In *Workshop de Educação em Computação*, pages 1–11. Sociedade Brasileira de Computação.

de Oliveira, M. and Oliveira, E. (2015). Abordagens, práticas e desafios da avaliação automática de exercícios de programação. In *Anais do IV Workshop de Desafios da Computação aplicada à Educação*, pages 131–140. SBC. DOI: 10.5753/desafie.2015.10048.

de Souza, R. L., Ferreira, F. Z., and da Costa Botelho, S. S. (2020). Proposta para avaliação de códigos fonte

com TF-IDF. In *Anais do XXXI Simpósio Brasileiro de Informática na Educação*, pages 112–121. SBC. DOI: 10.5753/cbie.sbie.2020.112.

Elo, A. E. (1978). *The rating of chessplayers, past and present*. Arco Pub., New York, USA.

Falckembach, G. and Araujo, F. (2013). Aprendizagem de algoritmos: dificuldades na resolução de problemas. *Anais do Congresso Sul Brasileiro de Computação*, 2:1–7.

Francisco, R. E., Ambrósio, A. P. L., Junior, C. X. P., and Fernandes, M. A. (2018). Juiz online no ensino de cs1-lições aprendidas e proposta de uma ferramenta. *Revista Brasileira de Informática na Educação*, 26(3):163–179. DOI: 10.5753/rbie.2018.26.03.163.

França, L. (2020). Competências e habilidades no ensino: o que são e como aplicá-las? Plataforma Educacional. Available in: https://www.somospar.com.br/competencias-e-habilidades. Access date: 20/11/2020.

Galasso, R. H. and Moreira, B. G. (2014). Integração do ambiente boca com o ambiente moodle para avaliação automática de algoritmos. In *Anais de Computer on the Beach*, pages 22–31. DOI: 10.14210/cotb.v0n0.pp.22-31.

Gardner, H. (2011). *Frames of mind: The theory of multiple intelligences*. Hachette Uk, New York, USA.

Gerdes, A., Jeuring, J. T., and Heeren, B. J. (2010). Using strategies for assessment of programming exercises. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 441–445. DOI: 10.1145/1734263.1734412.

Giordano, C. V., de Lira, L. N., Langhi, C., and Feitosa, M. D. (2021). Tecnologia de apoio ao ensino e aprendizagem de programação em graduações tecnológicas profissionais: Juiz on-line. *Boletim Técnico do Senac*, 47(2):127–140. DOI: 10.26849/bts.v47i2.886.

Gomes, A., Areias, C., Henriques, J., and Mendes, A. J. (2008). Aprendizagem de programação de computadores: dificuldades e ferramentas de suporte. *Revista Portuguesa de Pedagogia*, pages 161–179. DOI: 10.14195/1647-8614_42-2_9.

Gomes, A. and Mendes, A. J. (2007). Learning to program-difficulties and solutions. In *International Conference on Engineering Education*, pages 1–5.

Moissinac, B. and Vempaty, A. (2020). Mixed compensation multidimensional item response theory. In *International Conference on Intelligent Tutoring Systems*, pages 132–141, Cham. Springer International Publishing. DOI: 10.1007/978-3-030-49663-0_17.

Moreira, G., Holanda, W., Coutinho, J., and Chagas, F. (2018). Desafios na aprendizagem de programação introdutória em cursos de TI da UFERSA, campus Pau dos Ferros: um estudo exploratório. In *Anais do Encontro de Computação do Oeste Potiguar ECOP/UFERSA*, pages 90–96.

Nojosa, R. T. (2002). Teoria da Resposta ao Item (TRI): modelos multidimensionais. *Estudos em Avaliação Educacional*, 1(25):123–166. DOI: 10.18222/eae02520022193.

Paes, W. d. O. (2022). Habilidades necessárias para resolução de problemas de programação aplicados em sistemas de recomendação. Undergraduate Project (Computer Engineering), Universidade Federal do Rio Grande. Rio Grande. p.45.

Park, J. Y., Cornillie, F., Maas, H. L. v. d., and Noortgate, W. V. D. (2019). A multidimensional irt approach for dynamically monitoring ability growth in computerized practice environments. *Frontiers in Psychology*, 10:1–10. DOI: 10.3389/fpsyg.2019.00620.

Pasquali, L. (2018). *TRI–Teoria de resposta ao item: Teoria, procedimentos e aplicações*. Editora Appris.

Pea, R. D. and Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New ideas in psychology*, 2(2):137–168. DOI: 10.1016/0732-118X(84)90018-7.

Pelánek, R. (2016). Applications of the Elo rating system in adaptive educational systems. *Computers & Education*, 98:169–179. DOI: 10.1016/j.compedu.2016.03.017.

Perrenoud, P. (1999). *Construir as competências desde a escola*. Artmed, Porto Alegre.

Pimentel, E. P., de França, V. F., Noronha, R. V., and Omar, N. (2003). Avaliação contínua da aprendizagem, das competências e habilidades em programação de computadores. In *Anais do Workshop de Informática na Escola*, pages 533–544, Campinas. DOI: 10.5753/cbie.wie.2003.533-544.

Prisco, A., Penna, R., Botelho, S., Tonin, N., and Bez, J. L. (2018). A multidimensional elo model for matching learning objects. In *2018 IEEE Frontiers in Education Conference (FIE)*, pages 1–9. DOI: 10.1109/FIE.2018.8658847.

Reckase, M. D. (2006). Multidimensional item response theory. *Handbook of statistics*, 26:607–642. DOI: 10.1016/S0169-7161(06)26018-8.

Robins, A. (2010). Learning edge momentum: A new account of outcomes in cs1. *Computer Science Education*, 20(1):37–71. DOI: 10.1080/08993401003612167.

Rossler, J. H. (2004). O desenvolvimento do psiquismo na vida cotidiana: aproximações entre a psicologia de alexis n. leontiev e a teoria da vida cotidiana de agnes heller. *Cadernos Cedes*, 24:100–116. DOI: 10.1590/S0101-32622004000100007.

Selivon, M., Bezerra, J., and Tonin, N. (2015). Uri online judge academic: integração e consolidação da ferramenta no processo de ensino/aprendizagem. In *Anais do XXIII Workshop sobre Educação em Computação*, pages 188–195. SBC. DOI: 10.5753/wei.2015.10235.

Smole, K. (1999a). *Múltiplas Inteligências na Prática Escolar*. Ministério da Educação, Secretaria de Educação a Distância, Brasília.

Smole, K. C. S. (1999b). Múltiplas inteligências na prática escolar. *Brasília: Ministério da Educação, Secretaria de Educação a Distância*, page 80.

Vargas, A. P., dos Santos, R. A. P., Bez, J., Tonin, N., and da Costa Botelho, S. S. (2019). Um modelo de mediação pedagógica para ambientes massivos. *RENOTE*, 17(1):93–102. DOI: 10.22456/1679-1916.95711.

Wasik, S., Antczak, M., Badura, J., Laskowski, A., and Sternal, T. (2018). A survey on online judge systems and their applications. *ACM Computing Surveys (CSUR)*, 51(1):1–34. DOI: 10.1145/3143560.

Wing, J. (2006). Computational thinking. *Communications of the ACM*, 49(3):33–35. DOI: 10.1145/1118178.1118215.