Model for rapid assessment of engagement in academic courses using genetic algorithms

Ion Alexandru POPESCU, Nicolae BOLD

Pitesti University Center, University of Science and Technology Politehnica Bucharest

alexionpopescu@gmail.com, bold_nicolae@yahoo.com

Abstract: *This paper presents a model for rapid assessment of fundamental notions from an academic course and its implementation using the Python language. The model has two types of users: professor and student. The professor can enter test items along with answer options, and students must respond in a limited time to a test generated using genetic algorithms. The items are stored in a database to be used as a starting point for the test generator. To avoid the similarity of the tests, the items and answer options are permuted, thus obtaining results that correctly and impartially reflect the engagement in the course.*

Keywords: algorithm, learning, test, modelling, database, interfaces.

1. Introduction

In the current conditions, great emphasis is placed on students' attention to courses. Thus, considering the means of communication such as mobile phones, tablets, laptops that can become means of disrupting attention to courses, the problem arises of motivating students to actively participate, engage in the course and acquire minimal knowledge. In this sense, rapid tests can be a solution to solve this problem.

In order to spend as little time as possible in order to design and implement time these tests, a rapid mechanism is also needed for this. Testing can be done either through mobile devices or through other devices that have access to the Internet. In addition, the way the tests are composed must consider the limited response time. This problem can be solved by using items with multiple-choice answers.

The model that we will present in the following sections contains a mechanism for entering items and answer options into a database along with a list of courses in which they can be used (courses are numbered with consecutive numbers: 1, 2, 3, ...). Another component of the model is the selection of items to obtain a test by the professor. The created test is then transformed with genetic algorithms into as many tests as there are students by permuting the items and

https://doi.org/10.58503/icvl-v20y202526

within each item by permuting the answers. In this way, the correctness of the testing and the obtaining of objective results are ensured. The last component is the testing that involves the students and the testing mechanism. The results are stored in a database and automatically sent by email to the students, and the teacher is sent various statistics related to these results.

These quick tests can also be integrated into more complex platforms such as those presented in (Early Education Nation, 2024) and (Homeschool Solutions, 2024). Also, the generation process can take more into account pedagogical aspects, such as Bloom's Taxonomy (Rahim et al., 2017) or other academic criteria (Yildirim, 2010) and may be applied on several items or educational domains (Kim et al., 2021). In order to assess better the literature extent of the subject, a short quantitative search was made using the database Dimensions.ai. A map of the most used and relevant terms was obtained, shown in Figure 1.

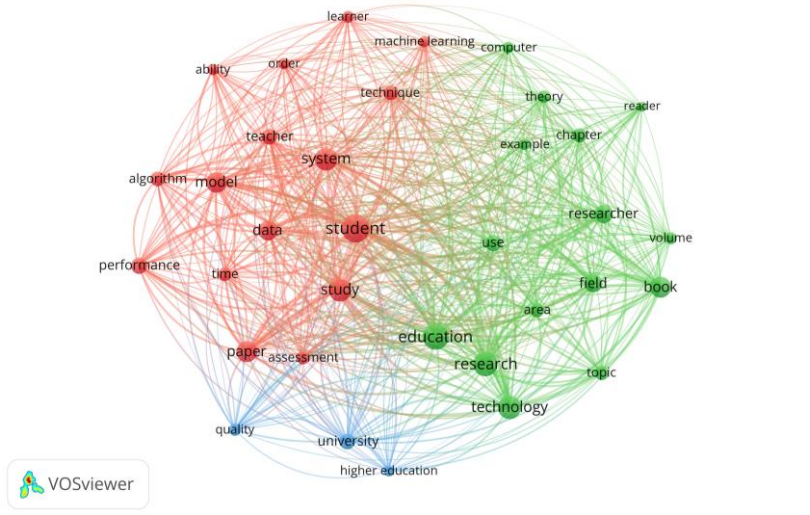

Figure 1. The term map obtained in the literature research

These terms represent key concepts associated with the generation of educational tests using genetic algorithms, and are classified by the number of occurrences and relevance score. Terms such as "student" (408 occurrences), "education" (349), and "research" (298) suggest that the educational field and research in this context are central aspects. At the same time, technical terms such as "algorithm" (114), "machine learning" (64), and "model" (215) indicate a close connection with advanced computational methods. Interestingly, terms with a high relevance score, such as "reader" (3.3968) and "volume" (2.8065), may reflect important factors in the analysis of test generation, perhaps in the context of educational resources or user interaction. The presence of terms such as assessment and performance indicate an emphasis on the evaluation and quality of the generated tests, which suggests that genetic algorithms are used not only for automation, but also for the optimization of the educational process.

2. Model presentation

The model will have three main components:

- Items

- Eval

- Verif

The first component **Items** presented in figure 1, contains a mechanism for the insertion of items within a database. The items have several components, such as the statement, the answer choices (a common item format such as single-answer Multiple-Choice Question – MCQ – was used, with 4 choices and one of them being the correct answer) and a list of courses that can be used for assessment (1, 2, 3, …), as well as a mechanism for selecting from the database the items preferred by the professor for the assessment and building the assessment test specific to the current course. By using a genetic algorithm for selecting the items and permutting the answer choices, a number of tests equal to the maximum number of students who can be present at the course is created.

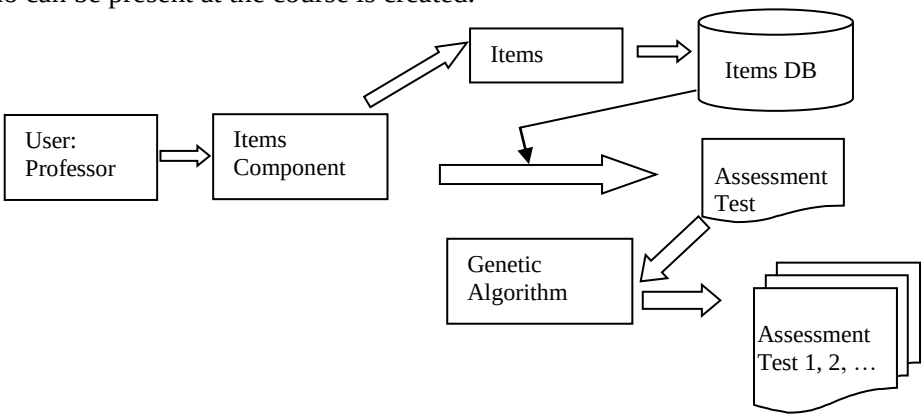

Figure 2. The Items component in the model

The second component, **Eval**, is the assessment component, shown in Figure 3, in which the student logs in and receives the test, selects the choices they consider correct and submits them to a database with answers.

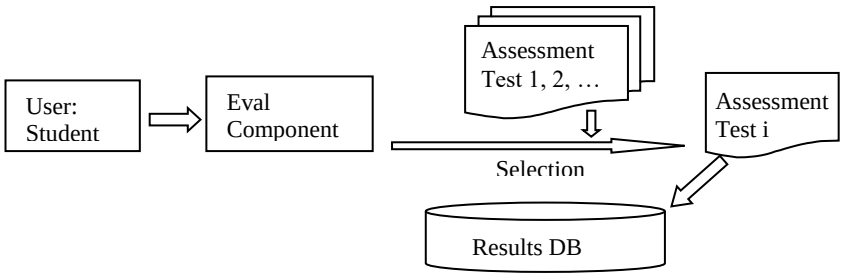

Figure 3. Eval component in the model

The third component, **Verif**, is managed by the professor, shown in Figure 4, to transmit the answers to the students and to produce various useful statistics for the courses to follow.

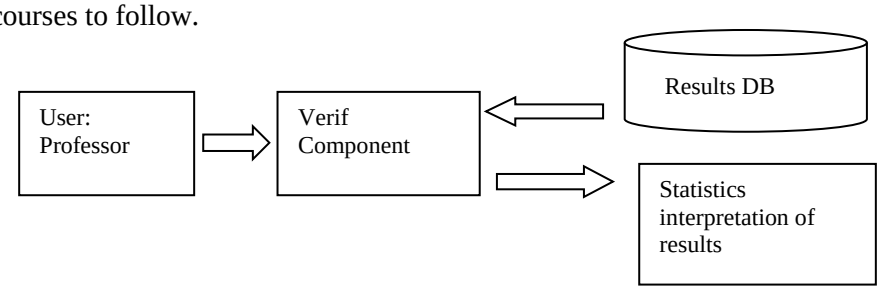

Figure 4. Verif component in the model

The three components of the presented model can be integrated into a web or mobile platform to obtain a complex application with many e-learning features, such as those in (Popescu et al., 2023), (Hamidi et al., 2011) and (Popescu et al., 2021a).

3. Implementation

In order to assess the efficiency of the model, several aspects of the model were implemented using a Python-based implementation, especially related to the assessment test generation. For the other aspects of the model, such as Eval and Verif, a separate implementation was used.

In order to determine the efficiency of the Items component, a Python implementation was made, with a number of 1000 (n = 1000) of items established in the database. The numeric measurement of the fastness and the variety characteristics of the algorithm was determined using a specific form of the fitness function. The form of the fitness function is shown in Equation (1).

$$f = w_1 \times E_T + w_2 \times M_T \quad (1)$$

where:

- E_T is the total entropy of the test, where the entropy measures the total measurement of the permutation of the choices of all items. The entropy of each item is the total number of inversions made to the choices reported to the total number of the possible inversions of all choices of the item. This entropy is calculated based on the permutation of the choices and using permutation-based computations. Thus, given an item q_i and its set of choices V = {v_1, v_2, …, v_l}, with l being the total number of choices, an initial order of choices O = (v_1, v_2, …, v_l) and a permutation of the order P = ($v_{\pi(1)}$, $v_{\pi(2)}$, …, $v_{\pi(l)}$), where π is a permutation of the indices {1, 2, …, l}, an inversion is considered made when two choices (v_j, v_k) exist such $j < k$ and $\pi(j) > \pi(k)$. Thus, the number of inversions made for an item q_i (I_i) and the total number of possible inversions (I_{max}) can be determined with the next relations:

$$I_i = \sum_{1 \le j \le k \le v} 1(\pi(j) \neq \pi(k)) \quad \text{and} \quad I_{max} = \frac{l \times (l-1)}{2},$$

where *1(condition)* is an indicator function which has the value 1 if the condition is true and 0 otherwise. This part of the fitness function determines the tests whose items are the most different from the initial items extracted from the database after the choices of the test items are permutted (item choice variation):

$$E_T = \frac{1}{m}\sum_{i=1}^{m} E_{q_i}, \qquad E_{q_i} = \frac{I_i}{I_{max}}$$

For example, for an item with $l = 4$ choices with the initial order O = (A, B, C, D) and the permutation P = (C, A, D, B), the original indexation would be A = 1, B = 2, C = 3 and D = 4 and the permutation is indexed as P = (3, 1, 4, 2). Each pair is then verified (e.g., (A, C) is considered inversion because C is after A in the permutation). The number of inversions I_i and I_{max} are then computed using the described relations, resulting a total entropy of the item being equal to 4 / 6 (0.67). Another version of inversions may be considered when a choice is set on a different index then the original one. This case will be treated in future research papers.

- M_T is the normalized value of the difference between the value of the calculated total time of the test and the time desired and given by the user (T_G). This part of the fitness function determines the closest tests to a given time by the user (time restriction).

$$M_T = 1 - \frac{|T_G - T|}{T_G}, \qquad T = \sum_{i=1}^{m} t_{q_i}$$

- w_1 and w_2 are the weights of the two parts of the fitness function and help the user to obtain tests giving greater importance either to item choice variation or time restriction, w_1, w_2 in [0,1]. The weights are used to give equal or different importances to the two parts of the fitness function (e.g., if the user desires the generation of a test with items which would have a greater degree of permutation, the choices being more disordered than the original order, then w_1 would have a greater value; if the user desires a test that would have a solving time closer to a given one, then w_2 would have a greater value).

Using this function, the user can obtain tests considering the two main important characteristics of the test: choice variety and total solving time. Using this form of the fitness function, the implementation was built and several trials for the parameters of the generation mechanism of the component Items were made. Firstly, the interface of the implementation of the Items components is shown in Figure 5.

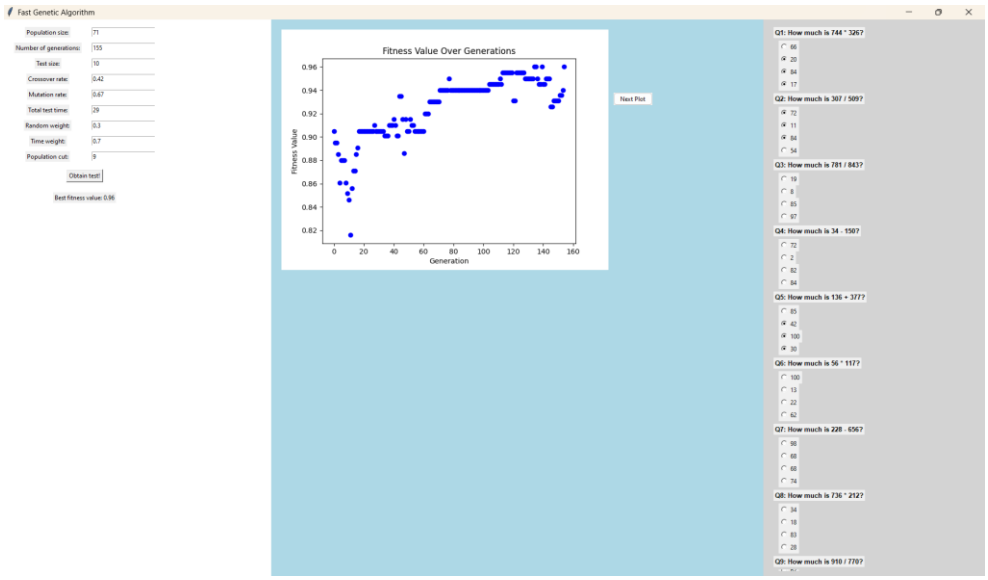

Figure 5. The implementation of the Items component

There were four characteristic that were studied for the efficiency of the generation mechanism: (1) convergence; (2) robustness; (3) performance for various parameter values; and (4) fitness distribution.

Firstly, the convergence of the algorithm was determined. For that, the genetic algorithm was run for a random set of parameters:

- the size of initial population (NP) is 71;
- the number of generations (NG) is 155;
- the number of items in the test (m) is 10;
- the crossover rate (rc) is 0.42;
- the mutation rate (rm) is 0.67;
- the total test time (TG) is 29 minutes;
- the values of w_1 and w_2, as they are explained above, are 0.3 and 0.7; these values have the meaning that the user desires a test that would respect to a greater extent the desired solving time opposing to the entropy (disorder) of the choices of the items that form the test; the choice of the value was randomly set;
- the number of the best individuals selected at each generation (S) is 9.

For a single run of the genetic algorithm, the best fitness value for this is 0.96, while fitness values are comprised between 0 and 1. The values of the fitness function were calculated after each generation, in order to determine the convergence of the algorithm. In this matter, the obtained values are presented in the next figure.

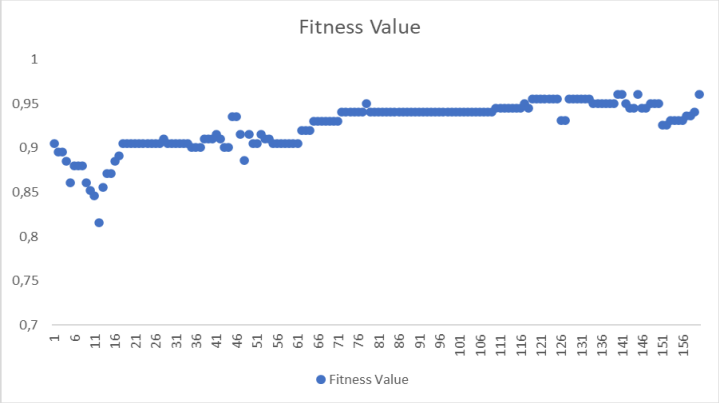

Figure 6. The convergence of the genetic algorithm

We can observe that the value of the fitness grows with each generation, with a minimun of 0.81 and a maximum of 0.96, reached at the final generation, meaning that the algorithm approaches a global optimum. The convergence rate for the fitness values is approximately 0.004. This indicates that the changes between consecutive fitness values are quite small, suggesting that the algorithm is approaching a point of stability and converging towards an optimal solution.

In order to determine robustness, we have calculated the statistical indicators after a set number (100) of genetic algorithm runs. For each run, the best chromosome obtained after the final generation was selected. The statistical values obtained for the 100 runs of the genetic algorithm provide an overview of its performance. The average fitness of 0.9395 indicates that the algorithm tends to reach consistently good quality solutions, with an overall performance that is close to optimal. The standard deviation of 0.0198 suggests a relatively small variation between the fitness obtained in each run, which indicates a stable and efficient convergence of the algorithm. It does not produce large fluctuations in performance, which may be a sign that the algorithm stabilizes quickly towards an optimal solution. The minimum fitness of 0.8609 and the maximum fitness of 0.9650 show that there is some variation, but that the solutions obtained are generally very close to the maximum possible value. In conclusion, these indicators suggest that the genetic algorithm used is efficient and convergent, with consistent performances and close to ideal solutions.

In order to determine the behaviour of the algorithm for specific values of the parameters, we have established specific values of these parameters, while a selected one was changed progressively. For the general setup, the values of the parameters were established to be NP = 50, NG = 50, m = 10, rc = rm = 0.5, TG = 30, w1 = w2 = 0.5 and S = 10. The fitness values obtained for progressive (with a growth rate g) values of selected parameters were obtained. The selected parameters were NP (with g = 10, 50 runs), NG (with g = 10, 100 runs), m (with g = 1, 100 runs) and w1 (with g = 0.01, 100 runs). The values obtained for the given setup are shown in the next figures.

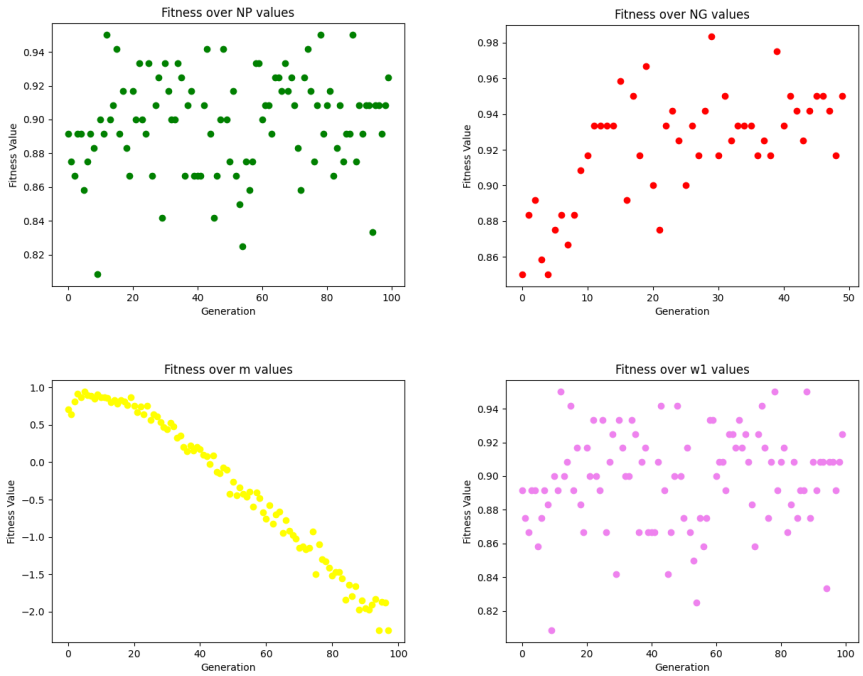

Figure 7. Values of fitness function for different values of NP, NG, m and w_1

The dependance of the algorithm has a strong connection with the parameters. The correlations are established as follows:

- The correlation coefficient of 0.1265 suggests a very weak positive correlation between NP and f. That is, there is a positive relationship between these two variables, but it is very weak and probably not statistically significant;
- NG and f have a positive correlation of 0.6297. This means that as NG increases, f tends to increase as well, but the relationship is moderate. A correlation of 1 would indicate a perfect linear relationship, so this suggests a moderate association between the two variables;
- m represents the number of questions in a test, and f represents the fitness value. A correlation coefficient of -0.8889 suggests a strong inverse relationship between the two variables. That is, as m (the number of questions) increases, f (the fitness value) decreases in a fairly consistent manner. The decrease may be also determined as the time (TG) was set to the double of questions, while the number of items increases;
- The correlation coefficient of -0.1819 indicates a weak negative correlation between w1 and f. That is, there is a negative relationship between these two variables, but this relationship is quite weak and

probably not statistically significant. This may mean that this suggests that, in general, response permutations that receive a higher weight do not lead to better genetic algorithm performance.

As for the Eval and Verif components, the implementation captures the mechanism of test solving and assessment performance. Screenshots of implementations related to these two components are shown in the next figure.

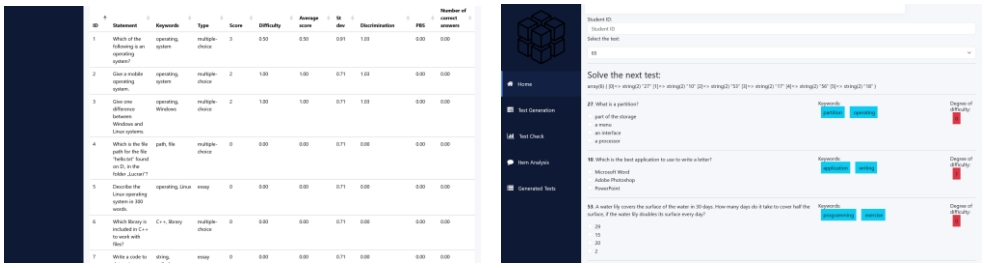

Figure 8. Implementation of Eval and Verif

The model's effectiveness is constrained by the lack of empirical validation for fitness function parameters, limiting adaptability across educational contexts. It also relies on predefined question structures, restricting flexibility for open-ended assessments. Additionally, scalability challenges may arise in large-scale applications, and comparisons with existing AI-based test generation tools are needed.

In conclusion, the model appears to be very consistent and performant, especially related to Items component and is achieving good results with little variation between runs. The average fitness of 0.9395 is very good, and the small standard deviation suggests that the algorithm is robust and does not generate very poor solutions. However, there is room for possible improvements, such as optimization to achieve higher fitness values in each run.

4. Conclusions and future work

Future research should refine the fitness function, integrate the model into e-learning platforms and enhance scalability in large-scale educational assessments. Comparative studies with machine learning-based approaches would further assess its effectiveness in adaptive testing environments. One key direction is refining the fitness function by incorporating dynamic weighting mechanisms based on learner profiles and real-time performance data. Lastly, the three components of the presented model can be integrated into a web or mobile platform to obtain a complex application with many e-learning features, such as those in (Popescu et al., 2023), (Hamidi et al., 2011) and (Popescu et al., 2021a).

REFERENCES

Bradley, V. M. (2021) Learning management system (lms) use with online instruction. *International Journal of Technology in Education*. 4 (1), 68–92.

Digital Science. (2018) *Dimensions [Software]*. *https://app.dimensions.ai*. Accessed on (DATE), under licence agreement.

Early Education Nation (2024) *How to Plan a Lesson or Activity - Activity Planner*, https://support.lillio.com/s/article/director-Creating-a-lesson-on-the-Activity-Planner [Accessed 2th January 2025].

Hamidi, F., Meshkat, M., Rezaee, M. & Jafari, M. (2011) Information Technology in Education. *Procedia Computer Science*. 3, 369-373. doi: 10.1016/j.procs.2010.12.062.

Homeschool Solutions (2024) *Creating a Lesson Plan with Repeating Pattern*, http://help.homeschooltracker.com/LP_Intervals.aspx, [Accessed 2th January 2025].

Kim, D., Choi, H. & Jung, G. (2021) Diagnostic assessment generation via combinatorial search. *arXiv*, [preprint] arXiv:2112.11188. [Accessed 2th January 2025].

Popescu, D. A., Cristea, D. M., Bold, N. (2023) A Transfer Learning Approach Interaction in an Academic Consortium. *22nd ICWL 2023: Sydney, NSW, Australia, 2023.* pp. 204-2019.

Popescu, D. A., Gosoiu, C. I. & Nijloveanu, D. (2021a) Learning Testing Model using Test Generators and Mobile Applications. *L2D@ WSDM*. pp. 41-48.

Popescu, D. A., Stanciu, G. C. & Nijloveanu, D. (2020) Application of Genetic Algorithm in the Generation of Exam Tests. *9-th International Workshop on Soft Computing Applications (SOFA)*. pp. 145-152.

Popescu, D. A., Stanciu, G. C. & Nijloveanu, D. (2021b) *Evaluation Test Generator Using a List of Keywords*. ITS 2021: pp. 481-489.

Popescu, D. A., Bold, N. & Popescu, I. A. (2016) The Generation of Tests of Knowledge Check Using Genetic Algorithms. *9-th International Workshop on Soft Computing Applications (SOFA)*. pp. 28-35.

Rahim, T. N. T. A., Aziz, Z. A., Rauf, R. H. A. & Shamsudin, N. (2017) Automated exam question generator using genetic algorithm. *2017 IEEE Conference on e-Learning, e-Management and e-Services (IC3e), Miri, Malaysia, 2017*. pp. 12-17. doi: 10.1109/IC3e.2017.8409231.

Seman, L. O., Hausmann, R. & Bezerra, E. A. (2018) On the students' perceptions of the knowledge formation when submitted to a project-based learning environment using web applications. *Computers Education*. 117, 16–30. doi: 10.1016/j.compedu.2017.10.0

Yildirim, M. (2010) A genetic algorithm for generating test from a question bank. *Comput. Appl. Eng. Educ*. 18, 298-305. doi: 10.1002/cae.20260.