Improvement of the Teaching Process Using the Genetic Algorithm

Goran Šimić[1(✉)], Aleksandar Jevremović[2], and Danilo Strugarević[3]

[1] School of Electrical and Computer Engineering at Academy of Technical and Art Applied Studies, 11000 Belgrade, Republic of Serbia
goran.simic@viser.edu.rs

[2] Faculty of Informatics and Computing, Singidunum University, 11000 Belgrade, Republic of Serbia
ajevremovic@ieee.org

[3] Academy of Applied Preschool Teaching and Health Studies, 37000 Kruševac, Republic of Serbia
strugarevic@avmss.edu.rs

Abstract. Teaching is a process that requires permanent observation and improvement. With the rapid development of e-learning, there was a need to review, improve and optimize the process of evaluating students' performance. The main objective of this study is to develop and implement the procedure for automatized generation of assessment tests. For this purpose, the proposed solution includes the application of a genetic algorithm. The procedure is developed on a sample of 29 existing test cases of "Internet programming" course. The optimization resulted in better selection and reduced number of questions used for evaluation of students' performance. The results of the study and the topicality of the issue point to the need for further research, both in the aspect of students' performance assessment, as well as in other aspects of higher education.

Keywords: Genetic algorithm · e-learning · Conceptual framework · Students' performance · Higher education

1 Introduction

The main objective of this research is to optimize the process of evaluating students' performance by using contemporary e-learning tools. To obtain reliable tests, the teachers are forced to produce a lot of questions of different types and difficulty, and permanently to improve them, at least before examination periods. Over time, the questions usually have been reused, and consequently, their reliability becomes questionable. Based on experience, it happens where there are large number of students and after using the same questions in multiple tests. The motivation of this study is to develop and implement the procedure for automatized generation of assessment tests based on optimized number of selected questions with reliability derived from historical data. For this purpose, the proposed solution includes the application of a genetic algorithm.

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2025
Published by Springer Nature Switzerland AG 2025. All Rights Reserved
D. Perakovic and L. Knapcikova (Eds.): FABULOUS 2024, LNICST 596, pp. 80–90, 2025.
https://doi.org/10.1007/978-3-031-72393-3_7

Based on literature analysis [1], there are many applications of artificial intelligence (hereinafter AI) in contemporary e-learning systems. They are organized in four areas: intelligent tutoring, adaptive systems and personalization, profiling and prediction, and assessment and evaluation. New developments in AI based educational assessment are more focused on improving assessment efficacy and validity [2].

Genetic algorithm (hereinafter GA) is commonly used in artificial intelligence as an optimization technique suitable for problems where approximations represent the satisfactory solutions. GA represents the subtype of evolutionary algorithms that are not often used, although existing implementations indicate the high efficiency in solving of many specific problems [3]. The same is with the implementation of GA in education. There are few theoretical works that consider how GA can be used for making improvements in educational processes [4, 5]. In one article there are comparative analysis (based on the review of 231 research articles) which suggest the various methods (classification, clustering, Artificial Intelligence methodologies like Expert Systems, Fuzzy Systems, Big Data Analytics, Text Mining, Sentiment Analysis, Pattern Mining, Process Mining, Graph Mining, etc.) for measuring of students' performances [6]. Some research is related to identifying at–risk students of computer science based on statistical analysis of their results [7]. On the other hand, there is a study with a more comprehensive approach. It considers the numerous quantitative factors such as theoretical, mathematical, practical, departmental, and other environmental marks to find the factor that has highest impact on student performances. The study uses genetic algorithm for this purpose [8]. However, there are no considerations how GA efficiency can be used for improving the teaching process.

2 Methodology of the Research

The research method was chosen based on its goal and purpose. For now, there is not a clearly defined, and mandatory method that should be applied in the process of evaluating students' knowledge. On the other hand, there is a need for refinement and optimization in this regard. Therefore, this research is designed as the development of a conceptual framework, and includes argumentation, explanation, and generation of an implementation procedure. A conceptual framework in research presented in this paper is used to understand a research problem (knowledge evaluation) and guide the development and analysis of the research. Bearing in mind that the application of the GA in the optimization of the evaluation of students' knowledge is in the development phase, the application of the conceptual approach can be considered optimal.

The procedure design is intended to optimize the teaching process related to the assessment of students' performance at School of Electrical and Computer Engineering (Academy of Technical and Art Applied Studies in Belgrade, Republic of Serbia). The procedure is developed by analyzing the sample of 120 students (absolvents) that participated in five test sessions (~20 questions per session) under the *Internet programming* course. The genetic algorithm is used as an optimization technique.

3 Problem Description

In contemporary e-learning systems, the evaluation of students' performances represents the activities that can provide feedback information for both students and teachers. The students can improve their learning efforts while the teachers can improve the quality of learning content and students' guidance. The e-learning platform enables the lecturer to create a question bank for the course. This bank consists of question categories, while each question belongs to a specific category (each question category is usually related to the specific learning topic). In the presented research, the first part of *Internet programming* course contains self-test questions grouped into six categories: *Basic concepts and terminology, HTML controls, JavaScript* (hereinafter JS) *basics, JS data types, JS collections and JS functions.*

Each question is of *Cloze* type [9] – it consists of embedded text fields in which the students should fill their answers (see Fig. 1). The presented question consists of three text fields. There are arbitrary number of text fields in the questions. The more fields the question consists of, the higher complexity it has.

Complete the JS code to replace the content of the 2nd paragraph (*"Paragraph 2"*) with *"Paragraph X"*:

```
<!DOCTYPE html>
<html>
  <body>
    <p>Paragraf 1</p>
    <p>Paragraf 2</p>
    <p>Paragraf 3</p>
    <script>
      document.[getElementsByTagName] ✓ ("p")[ [1] ✓ ].[innerHTML] ✓ = "Paragraf X";
    </script>
  </body>
</html>
```

Fig. 1. Question sample used for tests in *Internet programming* course.

Self-test results are useful for both students and teachers. Next illustration shows the self-test report for the teacher (see Fig. 2). The self-test consists of 22 questions from six categories (colored rectangles). In the presented example the maximum test result is 10 (100%) and question's share in the result (hereinafter *QF* - question factor) depends on both question and test complexity. In the presented test, full number of text fields (embedded answers) in all questions is 42 (maximum result is 10), that implies the question with one text field has QF = 0.24 (10/42), while question with four text field has QF = 0.95 (40/42).

It can be concluded that there can be different question factors for the same question in different tests. It can be calculated as follows (Eq. 1).

$$QF_{xa} = \frac{N_{tfx}}{N_{tfa}} M_{\mathrm{max}a}, \quad (1)$$

where QF_{xa} is a question factor for question x in test a, N_{tfx} is a number of text fields (embedded answers) in question x, N_{tfa} is a full number of text fields in test, and M_{maxa} represents the maximum mark for test a.

Grade/10.00	Q. 1 /0.24	Q. 2 /0.24	Q. 3 /0.24	Q. 4 /0.24	Q. 5 /0.24	Q. 6 /0.24	Q. 7 /0.24	Q. 8 /0.24	Q. 9 /0.24	Q. 10 /0.24	Q. 11 /0.71	Q. 12 /0.95	Q. 13 /0.24	Q. 14 /0.95	Q. 15 /0.48	Q. 16 /0.24	Q. 17 /0.24	Q. 18 /0.24	Q. 19 /0.71	Q. 20 /0.95	Q. 21 /0.95	Q. 22 /0.95
7,60	0,21	0,00	0,24	0,24	0,00	0,00	0,24	0,24	0,24	0,24	0,48	0,95	0,00	0,71	0,48	0,24	0,00	0,00	0,71	0,95	0,71	0,71
7,14	0,24	0,24	0,24	0,24	0,24	0,24	0,24	0,24	0,24	0,24	0,71	0,71	-	0,71	0,00	0,00	-	0,00	0,48	0,48	0,71	0,95
3,10	0,24	-	-	0,24	0,00	0,00	0,24	0,00	0,00	0,00	0,48	0,24	-	0,24	0,24	-	-	0,24	0,48	0,00	0,24	0,24
9,52	0,24	0,24	0,24	0,24	0,24	0,24	0,00	0,24	0,24	0,24	0,71	0,95	0,24	0,95	0,48	0,24	0,24	0,24	0,71	0,95	0,71	0,95
6,67	0,24	0,00	0,24	0,00	0,24	0,24	0,24	0,24	0,24	0,24	0,48	0,48	0,00	0,71	0,00	0,00	0,00	0,24	0,71	0,71	0,48	0,95
7,14	0,24	0,24	0,24	0,24	0,24	0,24	0,24	0,00	0,24	0,24	0,24	0,71	0,00	0,71	0,48	0,00	0,24	0,00	0,71	0,71	0,48	0,71
8,10	0,24	0,00	0,00	0,24	0,00	0,24	0,24	0,00	0,00	0,00	0,71	0,71	0,24	0,71	0,48	0,24	0,24	0,24	0,71	0,95	0,95	0,95
5,71	0,24	0,00	0,00	0,24	0,24	0,00	0,24	0,00	0,24	0,24	0,24	0,48	-	0,71	0,24	0,00	0,00	0,00	0,00	0,95	0,71	0,95
0,24	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	-	0,00	-	0,24	0,00	0,00	0,00	0,00	-	-	-

Internet concepts & terminology (Q. 1–3); HTML controls (Q. 4–10); JavaScript basics (Q. 11–13); JavaScript data types (Q. 14–16); JavaScript collections (Q. 17–19); JavaScript functions (Q. 20–22)

Fig. 2. The fragment of the teacher's test report

The *Cloze* type of question provides high flexibility of assessment. During question design, the teacher can provide more tolerance for students' answers (in the picture above, correct answers are checked by green signs, while partially correct answers are checked by yellow ones). Next illustration (see Fig. 3) demonstrates the *Cloze* questions flexibility: left text field (green box) is designed to accept both *getElementsByTagName* and *querySelectorAll* answers as correct ones. Expected string is case sensitive (*SHORTANSWER_C*). There are other options for making the system behavior during exams more flexible. For instance, answer can be partially accepted (<100%) if student enters only first part of the identifier (function, or type name), in case insensitive manner. This practice has proven to be useful considering modern software tools that help users with different *IntelliSense* features (e.g. code completion, parameter info, member lists, etc.).

Complete the JS code to replace the content of the 2nd paragraph (*"Paragraph 2"*) with *"Paragraph X"*:

```
<!DOCTYPE html>
<html>
  <body>
    <p>Paragraf 1</p>
    <p>Paragraf 2</p>
    <p>Paragraf 3</p>
    <script>
      document.[getElementsByTagName] ("p")[ 1 ].[innerHTML] = "Paragraf X";
    </script>
  </body>
</html>
```

```
<script>
   document.{1:SHORTANSWER_C:%100%getElementsByTagName~%100%querySelectorAll}("p")
</script>
```

Fig. 3. Fragment of JavaScript source code for *Cloze* question type

These results allow the teacher to improve the teaching process through detailed insight into the individual student's performances and, by analyzing the results of questions of specific category (the teacher can assess the quality of learning materials and tests). If there are many students in the group, a lot of questions in the question bank and a lot of performed tests, these activities can be time consuming, because teachers often give up on making quality improvements of the teaching process.

4 Proposed Solution

The problems described in the previous section have multidimensional complexity (e.g. students, topics and learning content, categories and questions used in different tests, time series of results, etc.). On the other hand, the tests are estimated as very important teaching activities as they obtain the feedback information necessary for making further e-learning improvements. Therefore, the presented research is focused on the optimization of tests based on the previously collected results and the test structure.

GA is selected as optimization AI (artificial intelligence) technique for fulfilling the main objective of the presented research. The described problem is recognized as a type of *knapsack* optimization problem [10]. GA is characterized by its structure (building blocks) and functionality (behavior). Very basic concepts are population, chromosomes, and genes. The population is a subset of all possible solutions, while the chromosome represents one of them. Chromosomes consist of genes. One gene holds the information relevant for problem solving. One test case (test with student results) represents the initial population for processing by GA (see Fig. 4). Each test record (e.g., TR1, TR2, etc.) is represented by a chromosome which consists of questions (e.g., Q1, Q2, etc.) – genes, and each gene consists of an allele – the student's result.

Fig. 4. Research problem (test case) structured for GA processing.

GA functionality can be described by pseudocode:

1. *Create initial population (initialization).*
2. *Repeat until the goal is not achieved:*
   a. *Perform fitness function on individuals from the population* (*evaluation*).
   b. *Choose the survivors* (*selection*).
   c. *Perform the genetical operations (recombination/mutation).*
   d. *Create a new generation of offsprings (g + 1 population)*

Practically, GA evaluates all chromosomes (test case record) in one population and chooses some of them (*survivors*) that best fit the problem solution. The fitness function [11] performs this task. Further, GA combines survivors' genes (questions) producing

the new generation of chromosomes (*offsprings*). GA repeats this sequence of actions (a, b, and c) until the ending conditions are not fulfilled. There are two main ending conditions: minor changes (convergency) of fitness values and limited (predefined) number of generations. Finally, GA processing will result in the specified number of test cases that can be used for future testing purposes.

The fitness function proposed in the solution is based on question factors (QF). The fitness value for question Q_x for test a can be calculated as follows (see Eq. 2).

$$fitness(Q_{xa}) = QF_{xa} / \left(\frac{\sum_{i=1}^{N} (QF_i / QR_{iavg})}{N} \right), \tag{2}$$

where QF_{xa} is a question factor of question x in test a that consists of N questions, QF_i is a question factor of i-th question in test a, $QR_{i\,avg}$ is a modifier based on QF_i and average of student results of i-th question. This way, the fitness value for each question in the selected population is normalized by a fraction of its factor QF and average of sums of fractions of factors and modifiers of all other questions in the considered test case. Consequently, fitness value for one question strongly depends on question factors of all other questions from the same population.

The proposed procedure includes several activities that can be split into two groups: data preparation and data processing (Fig. 5). Data preparation starts with selection of question category. This way only questions that belong to specific subject matter (see Sect. 3) are included in the GA processing. Further, the system loads the results records of the test cases that include the specified category. The purpose of the next activity (extraction and saving data in the new format) is data cleaning and preparation for reasoning. For instance, there are different number of questions from specific category included in the test case (different number of genes in chromosome can cause the unpredictable results), also there are unanswered questions in the records (they represent the information noise). Data processing starts with the formation of the initial population; prepared data are loaded into GA working memory. GA engine parameters should also be prepared: maximum number of generations, GA engine behavior regarding the trends

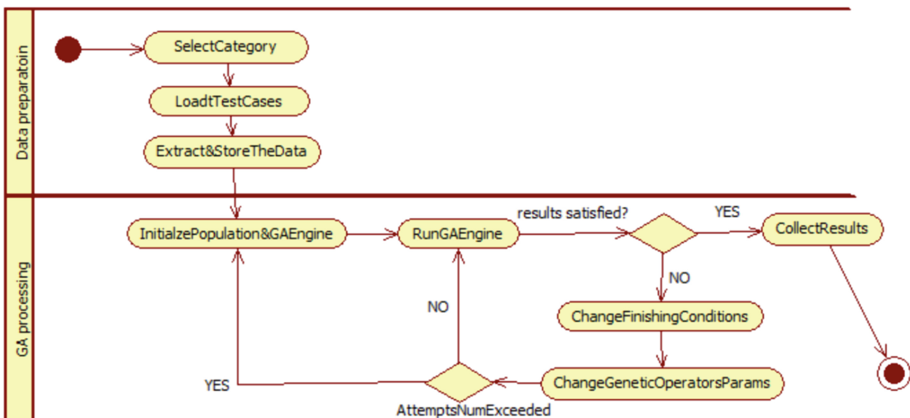

Fig. 5. Proposed procedure (activity diagram)

of fitness values (prevention of local maximum problem) [12], selection strategies, set up of genetical operators (genes crossover and mutation), processing of offsprings and finally, selection of results.

After preparation, the GA engine can be run. Based on experience, the first run rarely produces expected results. Changing the GA engine parameters (described above) is necessary for making improvements. This process is iterative and if there is no progress in results after several iterations, the initial population should be reconsidered.

5 Implementation

To depersonalize data and remove unnecessary test fields (e.g. questions, answers, evaluation rules, etc.), the test data needed for processing are extracted, filtered, and stored in separate table (Fig. 6). Testcases' IDs (*tcid*), questions IDs (*quid*), question's factors (*qfactor*) and results (*qresult*) represent the only fields included in the GA processing.

tcid	quid	qfactor	qresult
534	55	0.9800000190734863	0.3499999940395355
534	64	1.3700000047683716	0.6200000047683716
534	125	1.3700000047683716	0.6399999856948853
534	51	1.1799999475479126	0.5699999928474426
534	62	0.5899999737739563	0.3100000023841858
534	143	0.7799999713897705	0.3100000023841858
534	144	1.5700000524520874	0.6499999761581421
534	86	0.7799999713897705	0.1599999964237213
534	75	0.9800000190734863	0.550000011920929
534	138	0.38999998569488525	0.12999999523162842
548	55	0.9800000190734863	0.47999998927116394
548	64	1.3700000047683716	0.7200000286102295

Fig. 6. *TestCase* records – extracted for GA processing.

The application that uses *Jenetics* Java GA framework (https://jenetics.io) is developed to prove the assumptions described above. Basic class is named *TestCase,* and it must implement *Problem* interface to be solved by GA (see Fig. 5). In this specification sequence of questions will be processed by using *BitGene* type (question will be included in test case as a solution, or not), and Double is a type that will be comparable in fitness calculations. All members (attributes and methods) of *TestCase* class are static to prevent the uncontrolled number of GA engine runs.

The answered question is represented by immutable class (*record*) named *Question.* The object of *TestCase* class is constructed with the two values: the collection of questions, and number of questions (expected as a GA processing result). The *Codec* is a core class that enables software designers to transform (encode) the real-world problem into appropriate structure for GA processing (genotype). *TestCase* codec (*_codec*) is set up to make a subset of forwarded questions. *TestCase* class also must implement *fitness* function.

The class named Engine represents the GA implementation (see Fig. 7). To be fully executable, two basic stages should be implemented. Engine construction is the first one. Initially, the *TestCase* instance was forwarded to its builder. Further, the population

```
public final class TestCase implements Problem<ISeq<Question>, BitGene, Double> {

    private static ISeq<Question> makeISeqQuestions(List<TestCaseDerived> tcds) {...16 lines }

    public record Question(double qresult, double qfactor, int quid) implements Serializable {
        public Question {
            Requires.nonNegative(value: qresult);
            Requires.nonNegative(value: qfactor);
            Requires.nonNegative(length: quid);
        }
        public static Collector<Question, ?, Question> aggregateParams() {...16 lines }
    }

    private final Codec<ISeq<Question>, BitGene> _codec;
    private final int _testCaseSize;

    public TestCase(final ISeq<Question> questions, final int testCaseSize) {
        _codec = Codecs.ofSubSet(basicSet: questions);
        _testCaseSize = testCaseSize;
    }

    @Override
    public Function<ISeq<Question>, Double> fitness() {
        return questions -> {
            final Question sum = questions.stream().collect(collector: Question.aggregateParams());
            return sum.qfactor <= _testCaseSize ? sum.qfactor : 0;
        };
    }
```

Fig. 7. Structure of the *TestCase* – basic application class that implements GA.

size, survivors' selector, offspring selector, and GA operators (set up with appropriate probabilities) should be specified (Fig. 8).

```
public static void runGA(int counter, List<TestCaseDerived> tcds){
    final TestCase testcase = TestCase.of(population: tcds);
    int numOfQuesitons = tcds.size() + tcds.get(index: 0).testQuesitonsDerived.size();
    final Engine<BitGene, Double> engine = Engine.builder(problem: testcase)
            .populationSize(size: numOfQuesitons)
            .survivorsSelector(new TournamentSelector<>(sampleSize: 10))
            .offspringSelector(new RouletteWheelSelector<>())
            .alterers(
                    new Mutator<>(probability: 0.015),
                    new SinglePointCrossover<>(probability: 0.15))
            .build();
    final ISeq<Question> testcaseFinal = engine.stream()
            .limit(prdct: bySteadyFitness(generations: 7))
            .limit(maxSize: 100)
            .collect(collector:EvolutionResult.toBestResult(decoder: testcase.codec().decoder()));
    Object[] objs = testcaseFinal.toArray();
    for (Object obj : objs) {
        Question tq = (Question) obj;
        ....
    }
}
```

Fig. 8. The core of GA implementation

The GA processing is performed in the second stage, getting the evolution stream, and set up it with the limits. In the presented code, GA will stop if there are no changes of fitness values in the last seven generations, or if the number of generations exceeds one hundred. The results are collected from *TestCase* object (more precisely, decoded from its Codec instance). For instance, if the 15 questions are specified to be selected

among the full number of 616 questions in 28 test records, that means the GA outcome will get an array of 15 questions that best fit (see Sect. 4) the problem definition. Other words, the collection of resulted questions represents the solution proposed by GA. In concrete case, instead of the 22 questions test, GA selected 15 questions that are more representative regarding the collected students results and definition of fitness function.

6 Results

The experiment was performed by using an initial population formed by 29 test cases with 10 questions (of one category) each. GA engine was set up for maximum of 100 generations and it was run 471 times (Table 1). Each time best result contained 5–7 questions of 10. The test case of 6 questions was found as optimal solution with share of 58% and maximum fitness value of ~7.

Table 1. Experiment results.

Number of questions	Frequency	Share	Avg number of generations	Altered genes (avg)
5	104	22%	~9	3936
6	273	58%	~7	3151
7	94	20%	~8	3651

Regardless of predefined maximum of 100 generations, GA engine obtained the results in small number of generations (7–9) which is reasonable owing to small number of test cases and questions (10 genes per chromosome). It implies that the GA engine always stopped due to steady fitness value. Both genetical operators are used in experiment: mutations (with probability of $P_{mutations} = 0.015$) and single point crossovers (with probability of $P_{crossovers} = 0.15$). During the experiment, varying of these values produced only minor changes. The average time needed for one GA run is 57.202188 ms. The most of this time was spent for altering (genetical operations) – 46%, the fitness calculation consumed 30% of time while the selection time participated with 24%.

7 Conclusions

Research presented in the paper proposes a procedure for improving student assessment by implementation of genetic algorithm (GA). Lot of test records give the opportunity to use students' results for evaluating the questions by GA. The questions are designed in the *Cloze* format to improve their behavior (the questions used in test as an activity that includes the interaction with the student), and to provide the evaluation of the students' answers in a more sophisticated way. The questions have initial value based on their complexity. The e-learning systems use this value and *Cloze* script embedded in the question for evaluation of student's answer(s). These results, accumulated in

the numerous tests, contain a lot of information about questions. As the problem was recognized as *knapsack* problem type, the GA was chosen as appropriate for this purpose. The optimization of assessment process includes better selection of questions, smaller number of questions per test and implicitly less time needed for performing the test.

The research presents the transformation of test case data into the dedicated problem structure that can be evolutionary processed by GA. Original fitness function designed for measuring how questions fit the problem is also presented. The implementation of the solution is realized by using *Jenetics* GA framework. The important parts of implementation are presented in the paper: adaptation of data structures to the framework; creation of the evolutionary engine and set up all its operations (evaluation, selection, genetic operators, ending conditions); running of evolutionary engine and collecting of results. Loading of test records and further processing of results are not presented as they are out of scope.

To prove the concept, the experiment based on real data (the questions of one category and student answers) was performed. It included 471 runs of the GA engine in processing the same data set. Regardless of (pseudo) randomly selected survivors and offsprings, the mutate and crossover operations with predefined probability, these runs produced similar outcomes: from 5 to 7 selected questions (of 10 questions in category) with the same resulted fitness value. The most of results were obtained in the 7th generation. The average GA run times had a small standard deviation – less than 0.0034 that implies the stability and reliability of GA engine.

In further development, the solution should be adapted for integration into the contemporary e-learning environments. This way it can be utilized for a wider educational society. The design of prototype-based pluggable module for *Moodle* learning management systems (https://moodle.org) is likely to be the next phase. Another future task is to collect and analyze feedback information about the generated tests from teachers and students.

Besides tests, the other e-learning resources will be included for optimization. The transformation of these resources into concepts appropriate for processing by evolutionary algorithm will be the biggest challenge. The presented GA framework provides great opportunities for concurrent processing of complex data structures and advanced characteristics for evolutionary engine customization.

References

1. Tang, K.Y., Chang, C.Y., Hwang, G.J.: Trends in artificial intelligence-supported e-learning: a systematic review and co-citation network analysis (1998–2019). Interact. Learn. Environ. **31**(4), 2134–2152 (2023). https://doi.org/10.1080/10494820.2021.1875001
2. Gonzalez Calatayud, V., Prendes, P., Roig-Vila, R.: Artificial intelligence for student assessment: a systematic review. Appl. Sci. **11**, 5467 (2021). https://doi.org/10.3390/app11125467
3. Grabusts, P., Zorins, A.: Evolutionary algorithms learning methods in student education. In: Proceedings of the International Scientific Conference, 28–29 May 2021, vol. 5, Covid-19 Impact on Education, Information Technologies in Education, Innovation in Language Education (2021). https://doi.org/10.17770/sie2021vol5.6153
4. Pillay, N.: The impact of genetic programming in education. Genet. Program Evolvable Mach. **21**, 87–97 (2020). https://doi.org/10.1007/s10710-019-09362-4

5. Tanweer, A., Shamimul, Q., Amit, D., Benaida, M.: Genetic algorithm: reviews, implementations, and applications. Int. J. Eng. Pedagogy (iJEP) (2020). https://doi.org/10.48550/arXiv.2007.12673
6. Dol, S., Jawandhiya, P.: Systematic review and analysis of EDM for predicting the academic performance of students. J. Inst. Eng. (India) Ser. B (2024). https://link.springer.com/10.1007/s40031-024-00998-0
7. Qushem, U.B., Oyelere, S.S., Akçapınar, G., et al.: Unleashing the power of predictive analytics to identify at-risk students in computer science. Tech. Knowl. Learn. (2023). https://doi.org/10.1007/s10758-023-09674-6
8. Miranda Lakshmi, T., Martin, A., Prasanna Venkatesan, V.: An analysis of students performance using genetic algorithm. J. Comput. Sci. Appl. **1**(4), 75–79 (2013). http://article.computersciencesapplications.com/pdf/jcsa-1-4-3.pdf
9. Kleijn, S., Pander Maat, H., Sanders, T.: Cloze testing for comprehension assessment: the HyTeC-cloze. Lang. Test. **36**(4), 553–572 (2019). https://doi.org/10.1177/0265532219840382
10. Cacchiani, V., Iori, M., Locatelli, A., Martello, S.: Knapsack problems — an overview of recent advances. Part I: single knapsack problems. Comput. Oper. Res. **143**, 105692 (2022). ISSN 0305-0548. https://doi.org/10.1016/j.cor.2021.105692
11. Avdeenko, T.V., Serdyukov, K.E., Tsydenov, Z.B.: Formulation and research of new fitness function in the genetic algorithm for maximum code coverage. Procedia Comput. Sci. **186**, 713–720 (2021). https://doi.org/10.1016/j.procs.2021.04.194
12. Pan, F.: Research on path planning of AUV based on improved genetic algorithms. In: 2021 International Conference on Computer, Control and Robotics (ICCCR), Shanghai, China, pp. 26–29 (2021). https://doi.org/10.1109/ICCCR49711.2021.9349281