MODEL FOR COMPETENCY-BASED TEST GENERATION TO SUPPORT TEAM SELECTION USING GENETIC ALGORITHMS

Nicolae Bold[1], Ion Alexandru Popescu[2]

Within companies, research and government institutions, there are frequent situations when work teams need to be formed in which members are specialized in certain fields. For this specific context, the paper proposes an automatic knowledge test generation model based on genetic algorithms, used to assess the skills required for different roles in a team. In this context, a model was designed, implemented, and tested using a database of evaluation topics and a random selection of test items based on genetic algorithms. This approach facilitates the efficient selection of team members by generating knowledge assessment tests tailored to the competencies required for each role. The items are organized into categories that correspond to specific skills relevant to the positions within the team. Separate sections describe the proposed model and the applied algorithms, while the final section presents the results obtained from the model's implementation.

Keywords: tests assessment, genetic algorithm, categories, generation

1. Introduction

The paper presents a model for automatic generation of knowledge assessment tests based on genetic algorithms. The model is designed to evaluate candidates' competencies for specific roles within a team, supporting the selection process rather than directly forming the team.

The proposed model aims to address the problem of selecting members of a work team for projects, research activities, or other complex tasks that require diverse expertise. Team members are expected to possess competencies in specific domains relevant to the task at hand. The model is based on a structured database of assessment items, organized into competency-specific categories, and integrates mechanisms for generating role-specific evaluation tests.

[1]Assistant, Department of Mathematics and Computer Science, Pitesti University Center, National University of Science and Technology POLITEHNICA Bucharest, e-mail: nicolae.bold@upb.ro

[2]PhD Student, Department of Mathematics and Computer Science, Pitesti University Center, National University of Science and Technology POLITEHNICA Bucharest, e-mail: alexionpoplescu@gmail.com

Details of the model are provided in Section 3. The test types were separated to allow greater flexibility in selecting the algorithm used for generating the evaluation tests. In addition, the section outlines the constraints applied to the selection of test items, ensuring that they align with the competency requirements of the team members.

The algorithm used for the model is presented in Section 4. It uses groups of items, from each group a number (specified by the one who selects the team members) of items is selected used to build the evaluation test. Then, the implementation and testing of the model are presented in Section 6 of this paper.

The main contribution of the paper is a flexible framework for generating role-adapted knowledge assessment tests using genetic algorithms. The proposed model introduces a weighted role vector that aligns test item selection with the competencies required for each team role. Although the system does not perform direct team formation, it supports the process by providing competency-based evaluation results that can guide role assignment and team composition. Instead of randomly generating tests, the proposed model builds tests according to the requirements of each role in a team, based on a vector of importance by domains. Furthermore, the test results are interpreted to recommend the allocation of candidates to specific roles, providing an automatic and objective mechanism for building teams. This approach transforms the test generator into an active tool for selecting and organizing human resources, which is the original contribution of the paper.

2. **Literature review**

Automatic test generation using optimization algorithms has been widely explored. Rahim et al. [1] and Popescu et al. [2] applied genetic algorithms for selecting exam items under predefined constraints. Other authors, such as Tan et al. [3] and Duan [4], proposed adaptive or personalized course generation. However, these studies focus on learning and assessment at the individual level. In contrast, the present work introduces a role-based configuration of tests, aligning item selection with team role requirements rather than with personal adaptation.

Generally, generating tests with certain restrictions refers to the assessment of learning for courses, as referred in Rahim et al. [1]. For example, models presented by Seman et al., [5], Bradley [6] and Popescu et al. [2] are based on the selection of test items are presented that are similar to those used by us in the model we present.

The selection of items is an issue that takes part in a larger group of problems that need greater computational resources, where a lot of data must be computed, such as research, economics (as shown by Horoias et al.), [7], Popescu and Radulescu [8], banking (presented by Stefan and Tita)[9] or education (as shown by Šimić) [10], either teaching, learning (presented by Tan

et al.) [3] or assessment (described by Yang) [11]. For the selection of items to test the competencies of the work team members, we will use genetic algorithms, because the number of items in the database can be very large, there may be various restrictions and also the number of items in the tests can be large, as preseneted in the study of Popescu et al. [12]. This does not allow the use of exponential algorithms. The results obtained in the paper of Nutescu and Mocanu [13], Constantin et al.[14] and Popescu and Nastase [15] and confirmed by our research show that classic or improved (as described by Zhang et al. [16]) genetic algorithms are a good choice, according to Duan [4] for solving this problem.

The type of items is the one used in the works of Nutescu and Mocanu [13], Hamidi et al. [17] and Kolb [18] with certain modifications. The algorithm used to extract the items from each group of items uses genetic algorithms and is presented in Section 5. Variants of this algorithm for selecting data from a database were used in works of Bobade et al. [19], Popescu and Nastase [20] and Popescu et al.[2].

As for the formation of teams, representative works show that automatic team formation can be approached through evolutionary optimization assisted by reinforcement learning and person-job matching, like mentioned by Guo et al. [21], through artificial intelligence tools that combine coalition generation, Bayesian learning and Belbin roles for heterogeneous teams, as presented by Alberola et al. [22], respectively through multi-criteria goal programming models that integrate human factors and AI acceptance in personnel allocation in organizations, described in the works of La Torre et al. [23].

These approaches confirm that the integration of test generation with role-based evaluation and intelligent allocation strategies forms a solid foundation for developing automated systems that not only assess competencies but also actively support the efficient and optimized formation of work teams.

3. Description of the model

3.1. Purpose

The purpose of the proposed model is to facilitate the rapid and efficient selection of work team members, based on the specific competencies required for each domain, through an automated system based on knowledge assessment. The model uses a database of assessment topics and applies genetic algorithms to generate role-adapted tests that measure candidates' skills in a wide range of domains. The term "role-adapted test" refers to the fact that the structure of the test is adapted to the unique configuration of skills required by a specific team or role — not to the individual characteristics of a given test taker. These tests are tailored to assess the competencies relevant to the team positions, thus ensuring an optimized, dynamic and flexible selection process. Through the random and diversified selection of questions, the genetic algorithm optimizes the tests, guaranteeing a complete and balanced assessment

of skills, contributing to the formation of effective work teams that meet the specific requirements of each project or organization.

Initially, the model indirectly contributes to team formation by generating specialized tests for each role, but by the addition of several aspects — such as defining importance vectors and evaluating scores by category — the model also allows for direct and assisted team formation, by allocating candidates based on compatibility with role profiles.

3.2. An example

The model uses a database in which assessment items are grouped by categories. For example, if the work team has to create an IT application, we could have the following item groups: design and specifications, databases, algorithms, graph theory, security, testing and validation. In this concrete scenario, a company needs to assemble a work team to develop a custom IT application for managing internal resources. The team must cover the mentioned areas of competence. The global process would comprise the next steps:

Step 1: the team leader defines the number of test items per competence category, for example:
$m = 2_{design} + 2_{databases} + 3_{algorithms} + 1_{graphtheory} + 2_{security} + 2_{testing}$, totaling 12 items.

Step 2: the genetic algorithm is run for each category in order to obtain the set of items selected for a category. In this example, the genetic component is run for six times, each time for each of the six categories.

Step 3: the test is formed from the union of all the six obtained sets of items. The obtained test would be optimized related to the score of the items. The optimisation is related to a lower difference between the lowest and highest score of an item within a category, in order to express consistency of assessment within a category.

Step 4: the test is applied to the potential members of the team, with focus on scoring results on each of the domain.

Step 5: a scoring system would consist in the determination of the score of each category. In the example shown in Table 1, the scoring results would show that the candidate is the most proficient in Databases and Testing categories, making one's competencies valuable in these categories within the process.

An item in the database will contain: item statement, answer options (choices), the correct answer, the score and the category the item belongs to.

An example of an item would be:

- **item statement**: How many nodes does a tree with 100 edges have?
- **choices**: A) 100 B) 99 C) 101 D) 102
- **the correct answer**: C) 101
- **the score**: 2 points
- **category**: graph theory

TABLE 1. Example of scoring system for a candidate performance

Category	**Items**	**Max Score**	**Example Score**	**Score (%)**
Design	2	10	7	70%
Databases	2	10	9	90%
Algorithms	3	15	12	80%
Graph Theory	1	5	3	60%
Security	2	10	7	70%
Testing	2	10	9	90%
Total	12	60	47	78.3%

The model user must choose the groups used to generate the test and how many items to use from each group.

In order to further emphasize the teamwork formation, we will automate the choosing of the number of items appropriate for a specific role. In this matter, for a given role within the team, a weights vector wr will be determined for that specific role. In our example, the role r would be "Full Stack Developer" and each component of the vector would have values comprised between 0 and 10. Continuing the example, we would have the weights vector $wr = (6, 6, 9, 3, 6, 6)$, corresponding to the given assessment categories C. The next steps of the determination of the configuration of items would consist in:

- the determination of the total weight, made by summing up the values of the wr vector: $\sum wr = 6 + 6 + 9 + 3 + 6 + 6 = 36$;
- the distribution of the number per category, made by the calculation of a simple proportion: $m_k = \left\lfloor \frac{wr_k}{\sum wr} \cdot m \right\rfloor$;
- the distribution of the left empty places (if existent).

After the appliance of the steps, we would obtain the next distribution $m_k = (2, 2, 3, 1, 2, 2)$.

3.3. Mathematical description

The model is comprised of several components related to an assessment environment. In this matter, we are referring to a database of items and a set of categories, a mapping function, a feature vector for an item and a feature vector for a category, an item selection algorithm and an optimization function, which determines the score obtained by a candidate. The description of the mathematical model contains the next elements:

- data sets:
  - I, $I = \{i_1, i_2, ..., i_N\}$: the set of items. An item is a specific form of evaluation means, such as multiple-choice, essay, short-answer etc.. The total number of items in this set is N. In this matter, this set may be identified in the implementation with a database of items. The item component will be described further. An example would

be $I = \{1, 2, ..., 500\}$, containing the identification particle of each item.

- C, $C = \{c_1, c_2, ..., c_K\}$: the set of item categories, where K is the number of categories. Each category can be established as a subset $C_k \subseteq I, k = \overline{1, K}$, that is, for a category c_k, the subset C_k contains all the items within the k^{th} category. The user may select a desired number K_d of categories to be included in the test. An example would be $C =$ { Databases, Algorithms, Graph Theory, Security, Testing }.
- T, $T = \{i_1, i_2, ..., i_m\}$: the obtained assessment test, where m is the total number of items within the test, $m = \sum_{k=1}^{K_d} m_k$, m_i being the desired number of items desired from each category within the test:

$$T = \bigcup_{k=1}^{K_d} \{i_{k1}, i_{k2}, ..., i_{km_{K_d}}\}, i_{kj} \in C_k, \sum_{k=1}^{K_d} m_k = m \tag{1}$$

The meaning of the entire test collection of items T is that the test is comprised of m items, an item being also part of the entire test ($i_i \in T, i = \overline{1, m}$) as well as extracted and part of the subset of the items from the test that belong to a specific category ($i_i = i_j \in C_k, j = \overline{1, kl}, l = \overline{1, m_{K_d}}$). An example would be $T = \{22, 15, 104, 12, 56, 45, 35, 98, 11, 2, 432, 341\}$ with $m_k = (2, 2, 3, 1, 2, 2)$.

- functions:
  - cat, $cat : I \to C$:the mapping function, which associate an item with a specific category. The function associate an item with a category:

$$cat(i_j) = c_k, i_j \in C_k \tag{2}$$

For example, an item would be part of a category by the mapping $cat(356) = Database$.

- feature vectors:
  - x, $x_j \in \mathbb{R}^d, d = 1$: the feature vector for each item. This is the description of a specific item related to its characteristics. d is the number of characteristics each item has, in this model $d = 1$, because each item is characterized by a score p_j. For example, $x_{356} = [7.4]$, meaning that the item 356 has a score of 7.4.
  - v: for a specific category $C_k = \{i_{k1}, i_{k2}, ..., i_{kM_k}\}$, the feature vector includes the average value of the scores of all items within the category:

$$v_k = \frac{1}{M_k} \times \sum_{i_{kj} \in C_k} x_{kj} \tag{3}$$

For example, $v_{Database} = [6.9]$, meaning that the average score of the Database category is 6.9.

– t: for a specific test, the feature vector includes the total score of the test $S_T = \sum_{j=1}^{m} p_{i_j}$, the average score of the test $M_T = \frac{1}{m} S_T$ and the average deviation $d_T = \frac{1}{m} \sum_{j=1}^{m} |p_{i_j} - M_T|$:

$$t = [S_T, M_T, d_T], t \in \mathbb{R}^3 \tag{4}$$

For example, $t_{42} = [14, 3.5, 1.0]$, meaning that the test has a total of 14 points, with an average score of 3.5 and an average deviation of 1.0 (variated scores).

- algorithms:
  – *select*: the algorithm that uses a selection function that extracts a certain number m_{K_d} of items from each category C_k, obtaining a subset of items T:

$$T = \bigcup_{k=1}^{K} select(C_k, m_{K_d}) \tag{5}$$

The model schematics is presented in Figure 1.

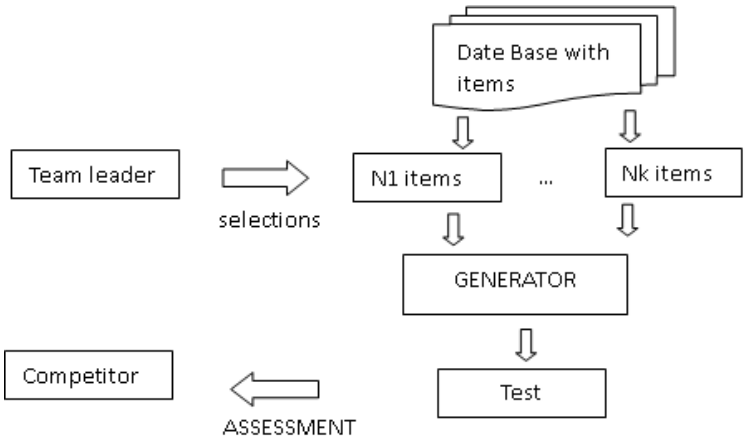


FIG. 1. Model components

As a conclusion, the generator selects items from the database categories of items that correspond to the assessment requirements. For each category, the generator uses a different genetic algorithm to select m_1, m_2, ..., m_{K_d} items to form the test with $m = m_1 + m_2 + ... + m_{K_d}$ assessment items.

3.4. **Input data and expected results**

Essentially, the input consists of a data set about domains, questions, and parameters for the genetic algorithm, and the output consists of an optimized test, with the highest possible fitness, that respects the requirements for selecting questions from each domain and distributing their scores.

More specific, the input data consists in:

- $m_k, k = \overline{1, K_d}$: the number of desired items to be selected from each category;
- P: the parameters of the genetic algorithm, presented in the Section 4, presented as a tuple $P = (NrP, NG, rm)$, where NrP is the size of the initial population (the number of chromosomes within the population), NG is the number of generations of the genetic algorithm and rm is the rate of mutation.

As for the determination of m_k according to the given example, the model contains several aspects:

- the set of roles r: $r = \{r_1, r_2, ..., r_Q\}$, where r_i is a given role (e.g., "Frontend Developer", "Backend Developer" etc.);
- the weights vector wr: $wr = \{wr_1, wr_2, ..., wr_k\}, wr_i \in \{1, 10\}$, which defines the importance of the domain or item category C for a specific role.

Then, the vector m_k is determined using:

$$m_k = \left\lfloor \frac{wr_k}{\sum wr} \cdot m \right\rfloor \tag{6}$$

The desired output consists in:

- **practical output**: a test T containing m items, each category c_k being represented by a number of m_{K_d} items. The optimization requirements relies on the choose of the test which has the most balanced items related to score differences. In this matter, the test with the least deviations within a category are selected:

$$F(T) = \frac{1}{K_d} \sum_{j=1}^{K_d} F(T_k), F(T_k) = 1 - \frac{d_{T_k}}{S_{T_k}} \tag{7}$$

, where $T_k = \{i_{k1}, i_{k2}, ..., i_{km_{K_d}}\}$ and $T = \bigcup_{k=1}^{K_d} T_k$.

- **experimental output**: analysis of the performance of the genetic algorithm:
  - *A1*: related to fitness results and convergence (fitness value analysis);
  - *A2*: related to runtime performance (runtime analysis).

4. Description of the algorithm in the model

For the model presented in Figure 1, the algorithm steps are as follows:

Step 1: The item categories along with the number necessary to generate the evaluation test are taken from the team leader: $m_1, m_2, ..., m_{K_d}$.

Step 2: Filter the database for each category $c_k, k = \{1, 2, \ldots, K_d\}$, and send the candidate test items together with m_{K_d} to the GENERATOR.

Step 3: Take the assessment test from the GENERATOR and send it to the competitor.

Step 4: Take the competitor's answers and send them to the team leader for assessment.

To implement the presented algorithm, a web implementation can be used, to obtain a web application that involves multiple users with access and security accounts in going through the stages of the model.

5. Description of the GENERATOR

5.1. Components

For the genetic algorithm we use:

- **genes**: a gene $g_j, j = \overline{1, m_{K_d}}$ codifies an item within both the category sub-test (T_k) and the whole test;
- **chromosomes**: a chromosome (Ch_j) codifies a sub-test T_k (a part of the test that contains items from the same category) and also, after the union of all chromosomes, a test T;
- **fitness function**: the fitness function (f) determines for a sub-test T_k the balance of the score within the sub-test. In other words, the tests with less difference between the lowest score item and the highest score item have a greater fitness value. In this way, more consistent tests are generated. The form of the fitness value has the form presented in Equation 7.
- **genetic operators**: the genetic operators bring modifications to the chromosomes in the population. For a chromosome $Ch = (g_1, g_2, ..., g_{m_{K_d}})$, there are three operators that can be used in the algorithm:
  - *mutation (Mut)*: the mutation operator swaps a randomly-chosen gene with a randomly-generated gene, as follows:
$$g'_i = u, u = \overline{1, N} \tag{8}$$
  - *crossover (Crs)*: the crossover operator selects two random parents and splits them at a random position, recombining the resulted parts of the two split chromosomes:
$$Ch_{child} = [P_1[1 : p], P_2[(p + 1) : m_{K_d}]] \tag{9}$$
where P_1 and P_2 are the two chromosomes selected as parents and $p \in \{1, m_{K_d} - 1\}$ is the randomly-generated position.
  - *sorting (Srt)*: the sorting operator ensures the elitism (the selection of the chromosomes with the highest fitness values). After the appliance of the Mut and Crs operators and the new chromosomes are added in the population, the chromosomes are sorted descending according to their fitness value and the first NrP chromosomes are kept.

5.2. Methodology

The steps of the GENERATOR algorithm are as follows:

Step 1: Repeat step 2 for m_{K_d} and the corresponding item category.

Step 2: Retrieve the data from the database I from the item category used for assessment.
Step 3: Calculation of the fitness for the obtained test.
Step 4: The obtained test is used to assess the competitors.

6. **Implementation and Results**

The presented algorithm was implemented using a web application. Statistical analysis and result interpretation were performed using descriptive and comparative metrics generated during algorithm execution. The code outputs include convergence plots, deviation and difficulty evolution graphs, and a summary dataset containing the results for different parameter configurations. The implementation and analysis were performed on a Windows 11 operating system using Python 3.11, with development carried out in Visual Studio Code. The implementation of the proposed model was developed in Python 3.11. Data management was handled through a local SQLite database that stores all generated items, including their difficulty levels, fuzzy membership values, assigned points, response times, and associated keywords. The algorithm evaluates each test using a multi-component fitness function that minimizes:

- the deviation from the target difficulty profile;
- the imbalance between disciplines;
- the difference between the actual and desired proportions of items per category.

Data analysis and visualization were carried out using NumPy, Pandas, Matplotlib, and Seaborn libraries. The results are saved in a CSV file for post-processing and reproducibility.

The experimental results were obtained using a combination of Python-based analytical tools and the Orange Data Mining environment. The implementation and data processing pipeline were developed in Python 3.11, using the following libraries:

- NumPy and Pandas - for numerical computation, data manipulation, and generation of descriptive statistics (means, deviations, fitness values);
- Matplotlib and Seaborn - for data visualization, including convergence plots, parameter influence, and comparative analysis;
- SQLite3 - for managing the local database of generated test items (difficulty, fuzzy values, time, and keywords);
- Orange Data Mining (v3.36.1) – for visual statistical analysis, correlation and regression exploration, and graphical comparison of parameters.

The Orange environment was used to import and analyze the CSV file exported from the Python implementation. Through the use of Orange widgets such as Data Table, Scatter Plot, Linear Regression, Correlation, and Distributions, additional insights were obtained regarding the behavior of the genetic algorithm under different configurations. This workflow enabled the comparison of the influence of algorithmic parameters: number of generations

(NG), population size (NP), and chromosome length (m) on both runtime and fitness values.

The obtained results confirm that the genetic algorithm achieves better optimization performance compared to random generation. All experiments and evaluations were executed locally in a reproducible environment. The code used for processing and evaluation is available upon request for reproducibility purposes. The form used to generate tests is presented in Figure 2.

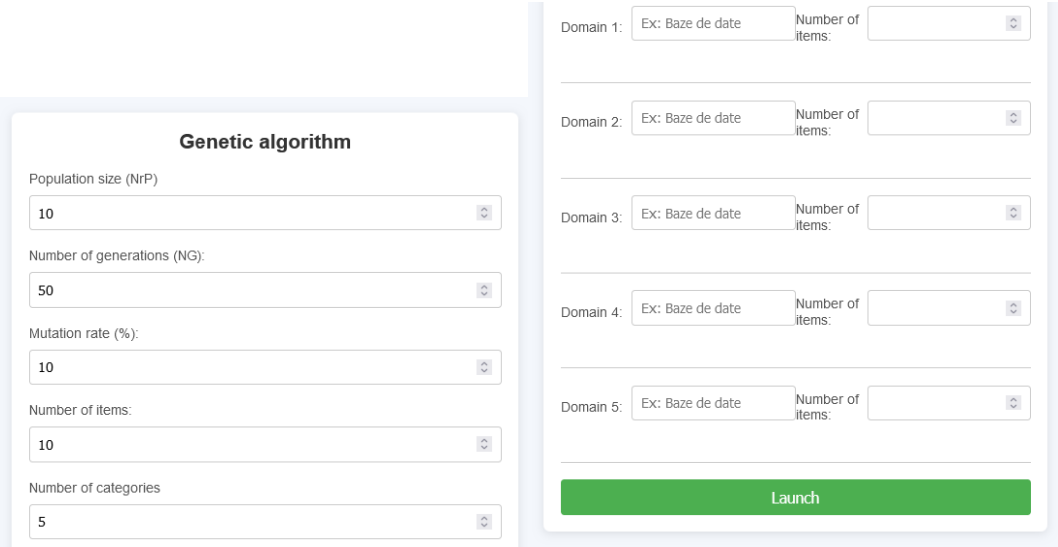


FIG. 2. Screenshot of the web application interface

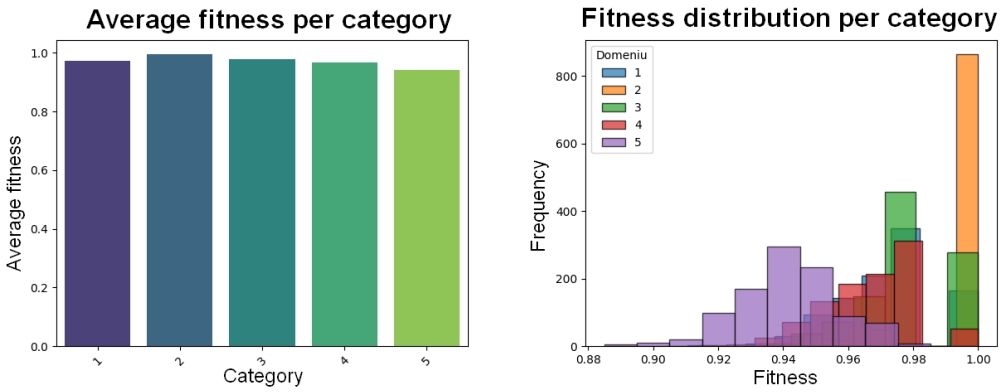
There were two directions of analysis of the results, $A1$ and $A2$, described earlier. These two directions are presented in the next subsections.

6.1. **Fitness analysis - A1**

For the first part of the analysis, two elements were studied: the fitness values for each domains, the influence of the genetic parameters on the values and the fitness convergence. In this matter, within the implementation, there were set a number of 1000 experiments. In each experiment, the genetic algorithm was run and the fitness for each category and the general fitness were determined. The values of the parameters were established as follows:

- the number of items in the database $N = 1000$;
- the initial population size $NrP = 10$;
- the number of generations $NG = 50$;
- the mutation rate $rm = 0.1(10\%)$;
- the number of categories $K_d = 5$;
- the number of desired items within the test for each category $m = (10, 6, 9, 11, 17)$.

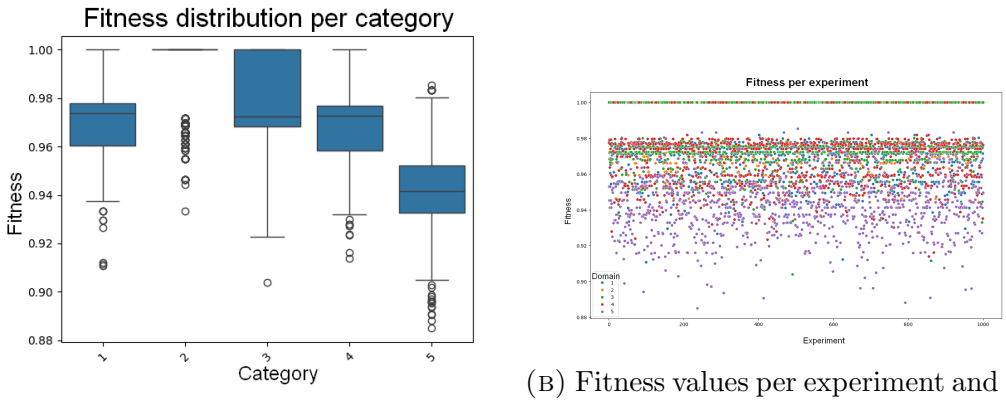
The results of the genetic algorithm are shown in Figure 3.



(A) Average fitness per category (B) Fitness distribution per category

FIG. 3. Average and distribution of fitness values per category, showing the optimization behavior of the genetic algorithm.

Figures 3 and 4 illustrate the distribution of fitness values for five distinct categories (Domain 1-5), each represented by different colors. Category 2 clearly dominates, with the highest frequency (over 800) and a strong concentration at fitness = 1.0, indicating superior performance. Categories 3, 4 and 5 show more balanced distributions, with fitness values concentrated between 0.94 and 0.98, partially overlapping. Category 5 covers a wider range of fitnesses, but with higher frequencies in the lower range (0.90-0.96). In contrast, Category 1 has a reduced presence, with low frequencies and fitness values limited close to the maximum (1.0). This distribution highlights a differentiated performance between categories, with Category 2 being the best performing and Category 1 the least represented.



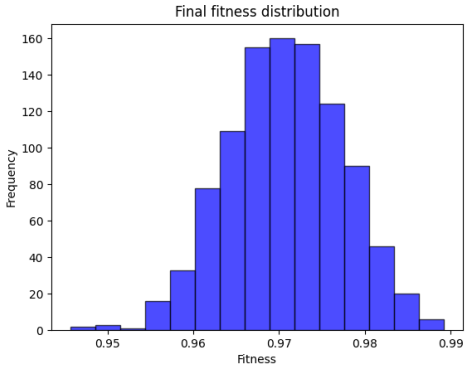
(A) Fitness distribution per category (B) Fitness values per experiment and category

FIG. 4. Fitness values distribution

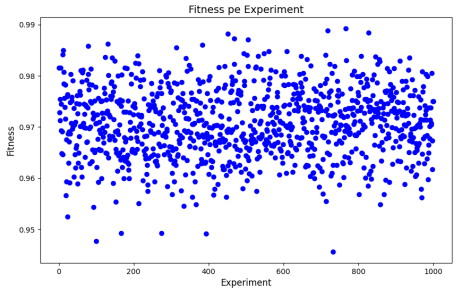
The average fitness values for the five categories range between 0.9496 and 1.0, highlighting different performances. Category 2 stands out with a

perfect fitness of 1.0, but includes the smallest number of questions (6), which may suggest a lower difficulty. Categories 1, 3 and 4 have similar average fitnesses, ranging from 0.9714 to 0.9756, but differ in the number of questions, being 10, 9 and 11, respectively. In contrast, Category 5, which has the largest number of questions (17), presents the lowest average fitness (0.9496), which could reflect a higher difficulty or a more inconsistent performance. These data highlight differences in both performance and distribution across categories.

We have also studied the behaviour of the final fitness value (F_T). The resulted values are shown in Figure 5.



(A) Fitness distribution per category



(B) Fitness values per experiment and category

FIG. 5. Final fitness values distribution

We can observe a relative high value of the average fitness between the categories. As related to convergence, a number of 500 generations was run in order to observe the progressive value of the fitness while the generations increase. The results are shown in Figure 6.

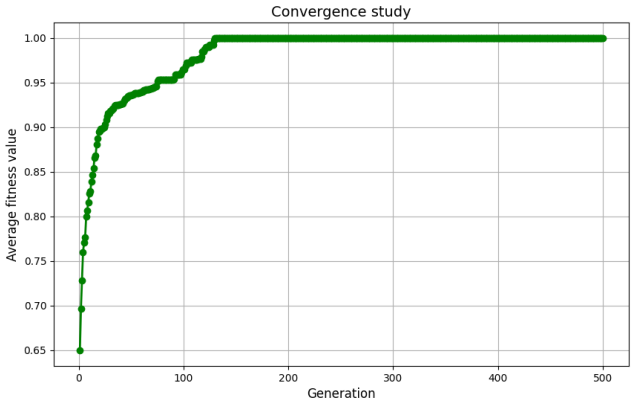
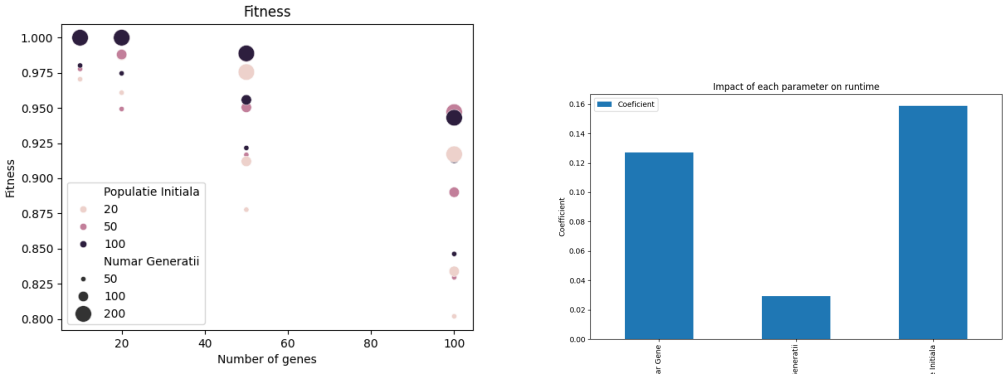


FIG. 6. Convergence study

Analyzing the evolution of the fitness value over generations, a significant increase is observed until generation 130, when it reaches the value of 1.0, a sign that the optimization algorithm has found the optimal solution. After this point, the fitness remains constant at 1, indicating that the process has fully converged, and the solutions have reached a state of stability in which no further changes are needed. This behavior suggests that the algorithm has evolved efficiently and identified a global maximum, without recording further improvements in subsequent stages.

6.2. **Runtime analysis - A2**

The algorithm was studied then for the efficiency related to generation time. The algorithm was studied in the context of a development of a regression, various data values for the three main parameters of the genetic algorithm being taken into account (NrP, NG and m_{K_d}, in order to observe their influence on the runtime of the algorithm. Some values are presented in Figure 7.



(A) The influence of the three parameters on the runtime (B) The coefficients of the obtained regression function

FIG. 7. Runtime analysis

Analyzing the results obtained from the various combinations of parameters (number of genes, number of generations and initial population), it is observed that the fitness reaches the maximum value of 1 in most cases with large parameters: for 100 genes, 100-200 generations and a large population of 50-100, for example, in the configuration 100 genes, 100 generations, 100 population, the fitness is 0.9094, and the execution time is 34.99 seconds. In contrast, combinations with fewer resources, such as 10 genes, 50-100 generations and a small population of 20-50, have much shorter execution times, such as for 10 genes, 50 generations, 20 population (fitness = 1, time 0.083 seconds), but with a lower fitness. Thus, for fast solutions and decent fitness, small values for genes, generations, and population can be used, and

for optimal solutions, it is recommended to increase them, accepting a longer execution time.

Regression analysis shows that the number of genes, the number of generations and the initial population positively influence fitness, with the highest coefficient being associated with the initial population (0.1588), followed by the number of genes (0.1271) and the number of generations (0.0294). The Mean Squared Error (MSE) is 351.76, indicating a significant error in the model predictions, and the regression intercept is -11.59, suggesting a negative value of fitness in the absence of the analyzed factors.

7. **Conclusions**

The presented model is an efficient way for the selection of candidates for building a work team based on competencies from various fields of activity. The evaluation is based on accumulated and verified knowledge and can be used in the section of candidates for different positions in an interdisciplinary team.

The algorithms used and the technologies used to implement the model are up-to-date and efficient. The results obtained by the implemented application are very good and show its usefulness.

The model allows the use of other item selection algorithms for the creation of evaluation tests. The model can also be used for other purposes, where evaluations using items must be performed, for example, the assessment of student progress in an academic course, such as those presented in [13], [15] and [20].

REFERENCES

[1] T. N. T. Abd Rahim, Z. Abd Aziz, R. H. Ab Rauf, N. Shamsudin, Automated exam question generator using genetic algorithm, in: 2017 IEEE Conference on e-Learning, e-Management and e-Services (IC3e), IEEE, 2017, pp. 12–17.

[2] D. A. Popescu, G. C. Stanciu, D. Nijloveanu, Evaluation test generator using a list of keywords, in: ITS 2021, 2021, pp. 481–489.

[3] X.-h. Tan, R.-m. Shen, Y. Wang, Personalized course generation and evolution based on genetic algorithms, Journal of Zhejiang University SCIENCE C 13 (12) (2012) 909–917.

[4] X. Duan, Automatic generation and evolution of personalized curriculum based on genetic algorithm, International Journal of Emerging Technologies in Learning (Online) 14 (12) (2019) 15.

[5] L. O. Seman, R. Hausmann, E. A. Bezerra, On the students' perceptions of the knowledge formation when submitted to a project-based learning environment using web applications, Computers Education 117 (2018) 16–30.

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

[7] R. Horoiaș, V. Tița, D. Nijloveanu, Study on the new management strategy for agricultural farms in romania, as a result of the energy crisis from 2021-2022. (2022).

[8] D. P. Anastasiu, D. Radulescu, Monitoring of irrigation systems using genetic algorithms, in: 2015 6th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO), IEEE, 2015, pp. 1–4.

[9] I. O. Ştefan, V. Tița, Analysis of the profitability ratios of romanian companies listed on bucharest stock exchange: trends and perspectives. (2021).
[10] G. Šimić, A. Jevremović, D. Strugarević, Improvement of the teaching process using the genetic algorithm, in: International Conference on Future Access Enablers of Ubiquitous and Intelligent Infrastructures, Springer, 2024, pp. 80–90.
[11] H. Yang, Design of intelligent exam management optimization system based on improved genetic algorithm, International Journal of High Speed Electronics and Systems (2024) 2540142.
[12] D. A. Popescu, C. I. Gosoiu, D. Nijloveanu, Learning testing model using test generators and mobile applications., in: L2D@ WSDM, 2021, pp. 41–48.
[13] C. I. Nutescu, M. Mocanu, Test data generation using genetic algorithms and information content, U.P.B. Scientific Bulletin, Series C 2 (2020).
[14] D. Constantin, C. Balcau, D. A. Popescu, Ica model estimation using a mixed learning rule based on genetic algorithms and neural networks, in: 3rd Evolutionary Data Mining and Machine Learning Workshop IEEE International Conference on Data Mining, IEEE Computer Society Press, Shanghai, China, 2023.
[15] D. A. Popescu, M. M. Nastase, Evaluation test generation model using two intervals of difficulty and keywords, in: MIS4TEL (Workshops), 2022, pp. 13–22.
[16] H. Zhang, B. Xiao, J. Li, M. Hou, An improved genetic algorithm and neural network-based evaluation model of classroom teaching quality in colleges and universities, Wireless Communications and Mobile Computing 2021 (1) (2021) 2602385.
[17] F. Hamidi, M. Meshkat, M. Rezaee, M. Jafari, Information technology in education, Procedia Computer Science 3 (2011) 369–373.
[18] D. A. Kolb, Experiential learning: Experience as the source of learning and development, FT Press, 2014.
[19] V. Bobade, S. Dandge, M. Pund, A study of different algorithm for automatic generation of question paper, International Science and Technology Journal 7 (5) (2018).
[20] D. A. Popescu, M. M. Nastase, Evaluation test generation model using degrees of difficulty and keywords, in: International Conference on Intelligent Tutoring Systems, 2022, pp. 197–203.
[21] Y. Guo, H. Wang, L. He, W. Pedrycz, P. N. Suganthan, L. Xing, Y. Song, A reinforcement learning-assisted genetic programming algorithm for team formation problem considering person-job matching, Neurocomputing (2025) 130917.
[22] J. M. Alberola, E. Del Val, V. Sanchez-Anguix, A. Palomares, M. D. Teruel, An artificial intelligence tool for heterogeneous team formation in the classroom, Knowledge-Based Systems 101 (2016) 1–14.
[23] D. La Torre, C. Colapinto, I. Durosini, S. Triberti, Team formation for human-artificial intelligence collaboration in the workplace: A goal programming model to foster organizational change, IEEE Transactions on Engineering management 70 (5) (2021) 1966–1976.