Development of the Improved Exercise Generation Metaheuristic Algorithm EGAL+ for End Users

https://doi.org/10.3991/ijet.v17i11.28099

Blanka Láng(✉), Balázs Dömsödi
Corvinus University of Budapest, Budapest, Hungary
blanka.lang@uni-corvinus.hu

Abstract—Exercise generation is a subject worthy of investigation. In our previous papers, a new multi-objective harmony search metaheuristic algorithm called EGAL was presented, designed to address a widely recognised problem: generating diverse exercises to measure students' knowledge on various topics. An improved metaheuristic algorithm (EGAL+) has since been created, and it is presented in this study. The aim of this research was to further develop EGAL and to investigate the differences between the original and the new algorithm. This newly acquired algorithm preserved the advances of EGAL – the generated exercises cover as many areas of the course as possible, the difficulty of the exercises are equal, and they are diverse. Moreover, the improved algorithm is also usable for non-expert users, since the introduced input fields are restricted to the ones which are freely editable. It is sufficient for the user to be proficient in their own field and to operate the program with subject-specific questions. These statements were confirmed by running EGAL+ on a large number of samples.

Keywords—exercise generation problem, multi-objective optimisation, harmony search metaheuristic algorithm, web-based applications, end-user development

1 Introduction

Creating and correcting tests are slow and difficult tasks in education. This is one of the practical reasons why exercise generation has been at the forefront of research for a long time. Generating exercises represents a challenging problem that has been identified by many researchers. Manually creating exercises is time-consuming, so several powerful exercise-generating systems have been proposed over the last few years, and many online learning platforms, which are robust and complex systems, generate exercises automatically. Later in this chapter, some examples of such systems are provided. These exercise-generating systems are useful for teachers because they save time and effort, and teacher workload is reduced as these tools are suitable not only for generating exercises, but also for correcting, evaluating and grading the answers. These tools are also useful for students because most of these systems are personalised and give effective feedback.

Some of these tools apply different levels of difficulty to handle students with different knowledge levels. Many were developed specifically for a particular field, such as language learning (grammar vocabulary trainers), engineering studies, mathematics or geometry exercises, SQL tasks etc. However, some of these tools are developed for general purposes. Some examples for these different types of exercise generators will now be enumerated.

It should be noted that the use of metaheuristic algorithms to solve educational problems is a common practice. Two recent examples are a genetic algorithm solution for a multi-objective optimisation problem, to create exam schedules [1], or a harmony search metaheuristic solution for automatic composition of instructional units [2]; however, there are many other similar examples [3], [4], [5], [6], [7], [8], [9], [10].

The generation of exercises has been extensively studied for a long time by many researchers e.g. [11], [12], [13], [14], [15], [16], [17], [18], [19]. Many automated exercise generation systems can be included in the Data Science domain. Data science emerged as a new field over the last few years, and new data science algorithms were developed to generate exercises [20]. Whereas the generation of exercises is a constantly evolving field of science, research published in the last decade has been examined in this chapter.

However, not only have topic-specific approaches been developed, but such tools have also been developed based on the way of reaching students. A template-based MOOC (Massive Open Online Course) approach is proposed by [12] to automate the teaching life cycle addressing creation of problems. The authors present suitable solutions and grading.

The key characteristics of MOOCs were identified by [21] which affect the affinity of students to online educational systems over a longer period. The most important measurements are the user’s affinity towards teaching content, difficulty, workload, and the duration of a lesson.

As emphasised in [11], most of the former solutions could not handle general problems, but only specified questions. The authors proposed a general solution for representing user inputs in the automated exercise generation process.

Some other applications prioritise personalisation. The ODALA+ (Ontology Driven Auto-evaluation Learning Approach) is presented by [22] for developing suitable exercises, which is a learning system on user personalisation. The authors have shown that developing adapted learning materials is an activity of fundamental importance. Knowledge, skills, and behaviour information was collected from the evaluation module and was integrated into ODALA+.

Next, some specific problem-solving applications will be presented. A solution is proposed by [23], which was developed to produce SQL questions automatically. This system uses difficulty levels and gives feedback for the users to help them improve their knowledge. The teachers do not need to create SQL questions, only database schemes.

A solution is offered by [24] for teaching differential equations in the field of Mathematics. It is personalised, because the student is identified using his or her answer history record of past answers, and the difficulty of the problems are optimised according to his or her knowledge.

Numerous task generation systems have been developed to support language learning. An approach is developed by [25] to generate FITB (Fill In The Bank English) learning exercises.

Some other applications prioritise ease of use, in that way similar of our EGAL+ algorithm. Passarola, a simple, powerful, exercise generation system was developed in such a way that anyone without a computer science background can use it to generate exercises. The researchers offer a specific language for creating more complex types of exercise than usual, such as multiple choice, data type or file comparisons [26]. The target is not a specific subject, as the system can solve general problems. The publication offers a set of examples of exercises created for courses that range from Mathematics to Music and Geography.

Moodle, one of the most popular LMSs (a learning management system is a form of software that users apply in the web-based learning process) is also capable of generating exercises for students and teachers. During such generation, different conditions may be specified. Various plug-ins have also been developed for Moodle. However, no solution can be found neither in Moodle nor among the plug-ins nor in any other LMS to the specific problem which is described in the next paragraph.

An overview of EGAL+, together with a description of the ways in which it differs from and is similar to the above, now follows. The purpose of this algorithm is to generate multiple different subsets of predetermined tasks (i.e. exercises) to test students' knowledge on various topics, in such a way that firstly, the quality of these subsets should be good enough according to a predefined quality matrix (i.e. they should cover the most important parts of the topics as much as possible), and secondly, the exercises are of equal difficulty. It is up to the teacher to decide which parts of the topics are the most important. These tasks are selected by the teacher based on which discipline – statistics, mathematics, informatics, language, and so on – he or she wants to assess students' knowledge of. For example, the topic can be the cascading style sheet subject in web development, and the parts can be some concrete CSS formatting, or another example can be the English grammar knowledge of verb tenses, and the parts can be the past tense of some concrete verbs. The previous examples [27], [28],[29] are university level exercises, but EGAL and EGAL+ can be used with students of different age and different knowledge depending on the teachers' choice.

Exercises should be diverse (according to a matrix and a diversity scale, which are both inserted into the multi-objective fitness function). Although the difficulties of these tasks can be different from each other, the difficulty of the subsets of the tasks should be the same. This is a problem without any satisfactory solution, as is shown in the above examples. Our previous studies have addressed this deficit. Unfortunately, the optimisation algorithm presented in our former publications has been usable only by researchers skilled in the field. Only they have been able to properly reconstruct and use it, based on the detailed information in our previous publications. The EGAL algorithm has now been transformed to an "easy-to-use" program, in a way that end users with no computer science and optimisation background could also use it, with certain restrictions and improvements. The improved metaheuristic algorithm (EGAL+) is presented in this paper. The goal of this research is to improve EGAL and to show the differences between EGAL and the newly acquired EGAL+.

Satisfying the needs mentioned previously – the quality of the subsets and the difficulty of the exercises – represents a much smaller-scale problem than that addressed by the other solutions in this chapter. One relevant difference is that EGAL+ is not a complex system, for example, this solution is not personalised and does not give feedback for the users.

On the other hand, this solution resembles other solutions previously described in this paper in the sense that is general, i.e. the problem can come from any field of science, e.g. statistics, mathematics, informatics, language, and so on, since EGAL+ will solve the problem regardless.

To summarise, in reviewing the literature it is found that - although many complex, adaptive, personalised, easy-to-use and effective exercise generating systems have been developed - such a tool capable of solving exactly the specified problem - to generate multiple different subsets of predetermined tasks, in such a way that the quality of these subsets should be good enough according to a predefined quality matrix, and the exercises are of equal difficulty – does not yet exist. No similar solution has previously been found, since the predefined quality matrix gives the solution a unique character, and no problem statement has previously been found where multiple different subsets of predetermined tasks are generated. Moreover, achieving all these things with an interface designed for non-expert users is also unprecedented, indicating that the added value of the EGAL+ algorithm is unquestionable.

2 End user problem

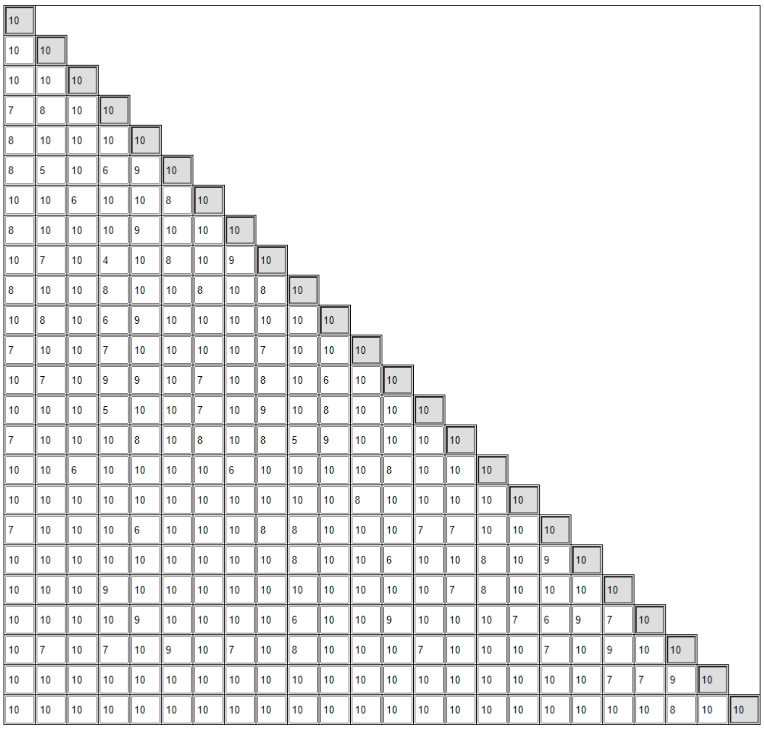
EGAL+ is an improvement of the EGAL algorithm, which was published in our former papers [27], [28], [29]. It will be shown that the original EGAL algorithm can be transformed for use by end users without a computer science and optimisation background, with some restrictions and improvements. In this paper the new algorithm EGAL+ will be presented, which differs in many respects from the original EGAL algorithm. The most important modifications are (i) choosing appropriate initial parameters; (ii) limiting the number of zeros in the quality matrix (iii); inserting a new function, which handles the difficulty problem and (iv) improving the fitness function.

EGAL+ is a web-based app that can be used via any current browser. Before going into the details of the system, it should be described how the EGAL+ algorithm is displayed to a user. A demonstration is available at https://egalplus.azurewebsites.net/. Here the user can run general cases – where the tasks are Task1, Task2, etc. –, and a specific CSS example can be run here as well – when the tasks are CSS formats. This example is one of those that were investigated in our former publications [27].

It should be noted that hosting the new algorithm imposes certain restrictions as regards running time. The probability of these restrictions occurring is small, but if the reader would like to check the algorithm without restrictions, it is recommended to use the GitHub code. The algorithm's code can be found at: https://github.com/balazs-domsodi-h53osf/EGALplus. In the *readme.txt* file, the reader can find a detailed user guide.

A brief explanation of how to run EGAL+ with the help of a short example now follows. When the teacher wants to generate some exercises using EGAL+, he/she first sets the following parameters: (i) PopSize: the number of the students in the class, for whom he or she wants to generate PopSize pieces of exercises, which are different and have acceptable quality according to a predefined quality matrix, (ii) NoT: the number of the optional tasks, (iii) the name of the tasks according to the specific problem, which in this example are: Task1, Task2, etc., (iv) TSize: the number of questions in one exercise, (v) difficulty: the difficulty of the tasks.

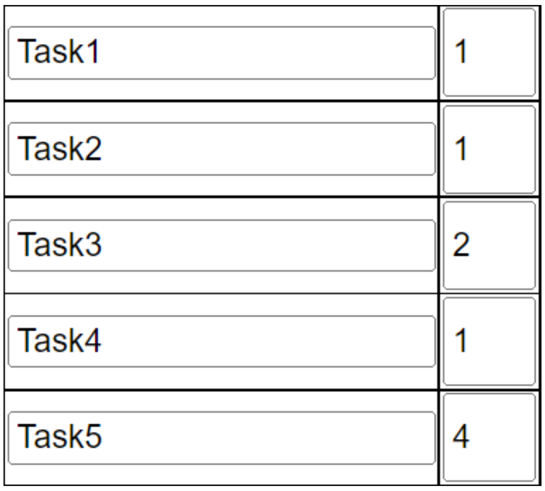
As mentioned previously, the purpose of the end user - for example, a teacher - is to generate multiple different subsets of predetermined tasks (i.e. exercises) to test students' knowledge. The first requirement of the teacher is that the quality of the subsets should be good enough according to a predefined quality matrix (i.e. they should cover the most important fields as much as possible). Consequently, the teacher sets the quality values in the quality E matrix (see Figure 1), which is a symmetric matrix. The matrix contains elements between 0 and 10, and has NoT rows and columns. The values of the elements indicate how much the user wishes to use the corresponding tasks in the same exercise. A value of zero means the two tasks cannot be run simultaneously, and a value of ten means they should appear together. In-between values indicate that the teacher prefers to include these tasks at the same time. A value closer to zero indicates that the tasks appear together less often.



10																							
10	10																						
10	10	10																					
7	8	10	10																				
8	10	10	10	10																			
8	5	10	6	9	10																		
10	10	6	10	10	8	10																	
8	10	10	10	9	10	10	10																
10	7	10	4	10	8	10	9	10															
8	10	10	8	10	10	8	10	8	10														
10	8	10	6	9	10	10	10	10	10	10													
7	10	10	7	10	10	10	10	7	10	10	10												
10	7	10	9	9	10	7	10	8	10	6	10	10											
10	10	10	5	10	10	7	10	9	10	8	10	10	10										
7	10	10	10	8	10	8	10	8	5	9	10	10	10	10									
10	10	6	10	10	10	10	6	10	10	10	10	8	10	10	10								
10	10	10	10	10	10	10	10	10	10	10	8	10	10	10	10	10							
7	10	10	10	6	10	10	10	8	8	10	10	10	7	7	10	10	10						
10	10	10	10	10	10	10	10	10	8	10	10	6	10	10	8	10	9	10					
10	10	10	9	10	10	10	10	10	10	10	10	10	10	7	8	10	10	10	10				
10	10	10	10	9	10	10	10	10	6	10	10	9	10	10	10	7	6	9	7	10			
10	7	10	7	10	9	10	7	10	8	10	10	10	7	10	10	10	7	10	9	10	10		
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	7	7	9	10	
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	8	10	10

Fig. 1. Quality matrix

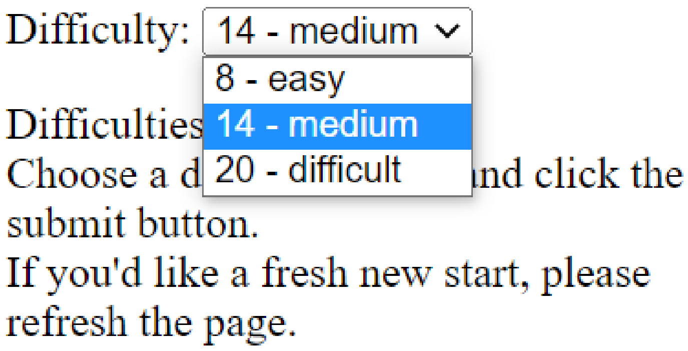
The second – and also important – expectation of the teacher that exercises should be diverse and although the difficulty of the tasks ("difficulty") can be different, the difficulty of the subsets, i.e. the sum of the difficulty of the selected tasks ("Difficulty") has to be of the same value, because it is assumed that one would like to use fair exercises. As the teacher sets the difficulty values of the tasks (see Figure 2), he or she can select these values from the set: {1,2,3,4,5}.



Task1	1
Task2	1
Task3	2
Task4	1
Task5	4

Fig. 2. Task difficulty values

The program – or more precisely, a function of the program – will offer three possibilities for the teacher – low, medium, and high – as the *Difficulty value* of the exercise. The user can choose one of these three values as can be seen in Figure 3.



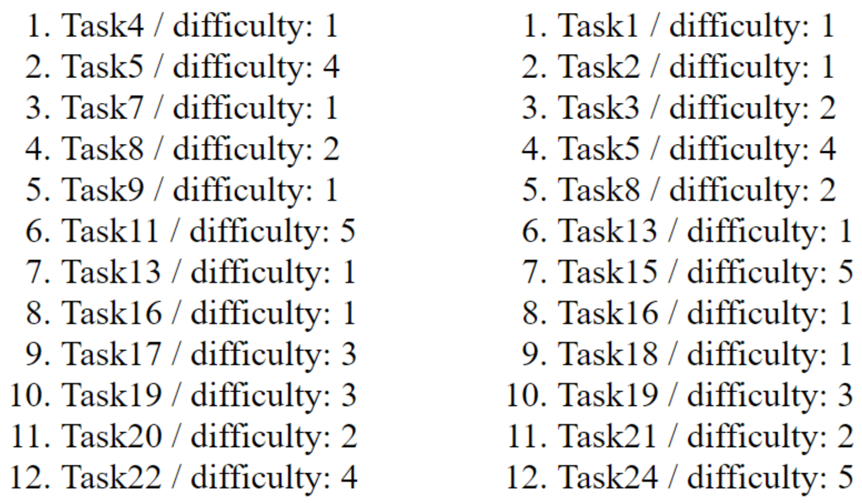
Difficulty: 14 - medium

8 - easy
14 - medium
20 - difficult

Difficulties
Choose a d … nd click the submit button.
If you'd like a fresh new start, please refresh the page.

Fig. 3. Difficulty value of an exercise

After these steps, by clicking the run button, the user receives the result: PopSize pieces exercises which are different and have acceptable quality according to a predefined quality matrix. A part of an output example is shown in Figure 4. In this case the chosen Difficulty value was 28.



```
 1. Task4 / difficulty: 1             1. Task1 / difficulty: 1
 2. Task5 / difficulty: 4             2. Task2 / difficulty: 1
 3. Task7 / difficulty: 1             3. Task3 / difficulty: 2
 4. Task8 / difficulty: 2             4. Task5 / difficulty: 4
 5. Task9 / difficulty: 1             5. Task8 / difficulty: 2
 6. Task11 / difficulty: 5            6. Task13 / difficulty: 1
 7. Task13 / difficulty: 1            7. Task15 / difficulty: 5
 8. Task16 / difficulty: 1            8. Task16 / difficulty: 1
 9. Task17 / difficulty: 3            9. Task18 / difficulty: 1
10. Task19 / difficulty: 3           10. Task19 / difficulty: 3
11. Task20 / difficulty: 2           11. Task21 / difficulty: 2
12. Task22 / difficulty: 4           12. Task24 / difficulty: 5
```

Fig. 4. A part of an EGAL+ output

In the EGAL program, the user can choose the above parameters, and the cumulated Difficulty value as well, independently of each other. In that process the user must be proficient at optimisation to select the right difficulty values, because occasionally the parameter settings could be contradictory. Some extreme examples are examined below. If every difficulty value is even but the exercise Difficulty value is not even, the parameter values are contradictory. Another contradictory case would be if a too big or too small a cumulated Difficulty value is selected: in this case, the freedom of the algorithm would be reduced. Another possible error would be if the user sets a NoT value which is less than the TSize value; in this case the values are contradictory, or the freedom of the heuristic algorithm has lost. Several other problematic examples could be shown.

It can be concluded that the user who selects the parameters must be familiar with metaheuristics, or the user and the optimisation expert must work together closely to choose the right input values. The EGAL program required the user to be a person with a solid knowledge of metaheuristics and optimisation. In the case of the EGAL+ program, the user can not choose contradictory parameter values. While developing EGAL+, the goal was to provide a modified, more user-friendly algorithm that anyone without an optimisation background can use to create powerful exercises.

3 An improved harmony search algorithm

In this chapter the improved exercise generation process, EGAL+, is introduced, with a special emphasis on improvements. Detailed description of the original EGAL can be found in our previous publications. Here the unchanged code parts are only briefly summarised, and details are given of those parts that are new or different in EGAL+.

It was not possible to use an exact algorithm to solve the exercise-generating problem mentioned above because the number of possible cases was unmanageable, so a harmony search (HS) metaheuristic algorithm [30] was applied. Since HS is a population-based heuristic algorithm, it converges to a global maximum or minimum. In the harmony memory – namely *population* – there are binary vectors for this algorithm. The fitness function value represents the quality of a vector. The quality of the individual value is maximised by maximising the fitness function value. In searching for the global optimum, the fitness function value should be maximised through further improvisations [30].

At the beginning of the HS, the corresponding parameters are set. These are (i) the harmony memory size (HM), (ii) the maximum generation number (iii), the HMCR (harmony consideration rate), and (iv) the PAR (pitch adjustment rate). After that the population of random vectors is initialised. The following is the improvisation step. According to Lee and Geem [30] a new harmony is improvised using HMCR and PAR probabilities. The quality of the population improves by each iteration until the algorithm is terminated.

The detailed description of EGAL+ is shown in the following. The first step sets the parameters. In this case, the harmony memory is the size of the population (PopSize), and the low and upper bounds are 10 and 100. HMCR and PAR were set as recommended by Lee and Geem to HMCR=0.5 and PAR=0.2. The Epsilon value was set low enough for the program to run long enough so that metaheuristics could give results which meet the specified requirements detailed in Chapter 2 with Epsilon=0.0000000001. This statement will be confirmed with the run results being shown later. The user can select only the PopSize value, as the other values are fixed.

At this point, the improvisation process had to be changed a little. If the user freely selects the difficulty values, the extreme situation occurs very rarely where the modification of the vector selected from the memory will not be successful at all. To solve this problem, if the modification does not occur after a specified period, a totally new vector is generated instead of the modified one. The probability of this case occurring is negligibly small.

It should be noted that depending on the other parameter settings, it is possible that they have a different value than the one specified, which would make the algorithm more efficient, run faster, or possibly give slightly better fitness values. For now, this increase in efficiency is not studied, because the primary goal of the research is to develop an easy-to-use program for the end user. The parameter calibration problem is worthy of further investigation in the future.

For the following parameters, upper and lower limits are specified, and the users are free to choose the values between the limits, which are presented in Table 1.

Table 1. Parameter settings

Parameters	Minimum value	Maximum value	Note
HM=PopSize	10	100	The number of students for whom exercises should be generated
Number of Tasks (NoT)	20	50	The optional tasks count
TSize	NoT/4	NoT/2	The number of the questions in one exercise
E quality matrix values	0	10	The maximum number of zeros: (((NoT-1+TSize*2)/2)*(NoT-TSize*2))/10 maximum number of zeros in a row: (NoT-TSize)/4
difficulty values	1 (easy)	5 (difficult)	The user is completely free to choose from the values of {1,2,3,4,5}

In the next step, the user sets the quality values in the quality matrix E. The procedure was described in detail in the previous chapter. One of the most important differences between EGAL and EGAL+ is the how the E matrix values are handled. In the case of EGAL, the user had to understand the optimisation and had to select the matrix values in such a way to run the algorithm without freezing. In EGAL+ a user can select relatively freely the values of the matrix, but this problem of freedom must be managed. It is necessary to limit the number of zeros in the matrix and the number of zeros in a row/column, to prevent the user from entering too many zeros. The result of too many zero values can be a problem. For example, the program loses its freedom and cannot run. You can find the suitable limits in Table 1. These limits are calculated in the following way: first those cases are specified for which the number of the zeros will prevent the algorithm from running. Then these limits are relaxed until the required freedom is reached and the algorithm will run. This statement is to be confirmed later.

Since EGAL+ is HS, the following is the initialisation. The initial population consists of random vectors. Every vector contains TSize pieces of 1-bits, and NoT-Tsize zeros. If the *ith* bit of a vector is zero, the *ith* task is not chosen, otherwise it is chosen.

In the next step, the fitness function values are calculated. In the case of EGAL the goal was to maximise the quality of exercises, and the distance between individuals in the population simultaneously. These features were inserted into the fitness function. In the case of EGAL+, the purpose is the same, but the fitness function had to be modified, because the user can select most of the parameters freely without optimisation knowledge. The two operands are the same, the modifications are the deleted normalisation of the operands, and inserting new ratio multipliers in the function.

The first operand of the fitness function was calculated as described, where p is an individual and E is the quality matrix:

$$F_{first}(p) = \sum_{i=0}^{NoT-1} \sum_{j=0}^{NoT-1} p[i] * p[j] * E[i,j] \quad (1)$$

The second operand in the fitness function is the diversity measure. The distance of two vectors in the same population equals the number of different digits. Then the diversity measure (D) of the p vector is determined. It is equal to the sum of the diversity for all vectors in the population.

The selection function combines these two operands with an addition operation. In the case of the diversity and fitness values the ranges may differ. In the case of EGAL+, the fitness function was modified due to the free parameter settings. Initially, the normalisation was deleted, then afterwards the values in the second operand were divided by PoPSize/20. The purpose in doing this was that in the case of every parameter value, the quality operand was more, or at least as, important as the difference operand. It was aimed to keep the ratio of the two operands between 0.1 and 1. In the next chapter it is shown that this value is within the required limits in all parameter settings. After these steps, the fitness function will be equal to the sum of the first operand and to the second operand*20/PopSize.

The last important difference between the original and the new algorithms is how the user selects the Difficulty value for the exercises. First the user selects the difficulty values for the tasks between 1 (easy) and 5 (difficult). In the previous chapter the freedom problem that can occur when a non-expert user selects a cumulative Difficulty value is demonstrated. The EGAL+ program will offer three possibilities for the teacher - low, medium, and high - as the Difficulty value of the exercise according to the selected difficulty values. These values were calculated in such a way as to give enough freedom to the program. It is shown in detail below how the three values were calculated.

Since examining all possible combinations is to be avoided for performance reasons, a heuristic solution is used instead of an exact algorithm. A function in the program randomly lists the possible difficulty combinations and puts the task sequences which already have been examined into a "taboo list", which stores them in a vector, so they will not be double-checked unnecessarily. A form of taboo list is used [31], with the difference that it is not FIFO; nothing is deleted from it. In addition, the possible total difficulties and their occurrences are collected in a two-dimensional vector. If the occurrence number of a total difficulty value exceeds a predetermined value (2 * PopSize), the user is allowed to select it. This function is stopped when it is considered that it has given a sufficient and well-distributed total number of difficulties. The number of pieces is three, because easy, medium, and difficult exercises are to be generated.

The distribution problem was solved according to the following heuristic: the minimum and maximum cumulative Difficulty values are considered and half of their distance (difference_goal) is taken as the starting value. The program is run for a predetermined time interval and the outputs are examined: whether the three values are returned whose distance is "*difference goal*" or not. If they have not been returned yet, the difference goal value is decreased by one again and again, always examining whether the three values are returned or not. This step is repeated until three values are returned: x1, x2, x3 as "easy", "medium", "difficult" values for the distance-goal distance from each other. The viability of this part of the program will be confirmed with random run results.

After this, more improvisation phases follow until the algorithm is terminated, which occurs if the average fitness value of the last 10 populations did not improve more than Epsilon.

The most important difference between the original and the new algorithms were (i) how the parameters were handled: some of them were fixed and some of them could be chosen by the user between some limits, (ii) how the number of zeros in the E matrix was limited, (iii) how the D difference was handled: a function was calculated and the user could choose only from the generated results and (iv) how the ratio was inserted into the fitness function. With these and some other minor modifications, the algorithm could be transformed for end users without computer science and optimisation background. By the end of the algorithm, the population will consist of exercises which are different and have acceptable quality according to a predefined quality matrix - these are guaranteed by the fitness function - and the Difficulty of the exercises will be equal. This statement will be confirmed in the next chapter, in which the program is run on a large number of samples.

4 Computational results

Reviewing the relevant literature indicates that there is no formerly existing algorithm which solves exactly the specific problem described in the Introduction, so the result of this algorithm cannot be compared with the results of other formerly existing algorithms. To illustrate the essence and viability of this algorithm, detailed computational results are given in this chapter. Random values were selected for each possible parameter between the upper and lower limits given in Table 1, and computational results were given for each case. All procedures were coded in the PHP 7.0 language. All computational results were acquired on a Laptop with Intel Core i7-9750H 2.6 GHz CPU and 16.0 GB RAM.

250 such random parameter sets were created, and the algorithm was performed for 20 runs for each instance, so the number of the total run is 5000. Aggregated results are shown in Table 2. (More detailed results can be found here: https://github.com/balazs-domsodi-h53osf/EGALplus/raw/main/result.xlsx.) Fitness function values, fitness function value improvements, difference values, the ratio of the two fitness function operands and solution times can be found in the tables. For these values, you can find minimum and maximum, average, and standard deviation values rounded to three decimal places.

Table 2. Computational results

	Minimum	**Maximum**	**Average**	**Standard deviation**
Fitness value improvement (%)	1.12%	14.95%	3.54%	0.55%
Difference (%)	2.182	25.235	13.445	0.162
Operands ratio	0.122	0.993	0.323	0.005
Solution time (sec)	4.000	1243.000	45.026	3.822

The results of the tables show that the algorithm ran for each random parameter set, and the improvement of the fitness function values can be seen. In addition, the ratio of the two fitness function operands is between 0.1 and 1 as mentioned before, the minimum value was 0.122 and the maximum value was 0.993. Although the solution times are in most cases within acceptable limits (0-360 sec), a few higher values can be found

– higher than 360 seconds (1.64% of all cases). In the next chapter, a planned improvement will be mentioned to reduce these high results. The results show that the difficulty problem is solved, and the exercises are different in one population. When the algorithm is run using the same parameter sets twenty times, the detailed standard deviation values confirm the robustness of the algorithm.

A detailed discussion of how the set goals were met now follows. As stated at the beginning of this article, the goal of this research was to improve EGAL and to show the differences between EGAL and the newly acquired EGAL+. Firstly, this new algorithm has managed to preserve the advances of EGAL – the generated exercises cover as many areas of the course as possible according to the predefined quality matrix and the multi-objective fitness function, the exercises are diverse according to the diversity measure, and their difficulty are equal. Furthermore, the improved algorithm has become usable for end users without a computer science and optimisation background since the newly introduced input fields are restricted to the ones which can be changed freely without breaking the program. It is sufficient for the user to be an expert in their field and to upload the program with questions adequate to the subject. One of the most significant changes in the algorithm is that the background calculation of the difficulty options was introduced so that the user can only select from the available goals calculated by the program. The statement – the program gives correct results within predetermined acceptable time using values within the allowed limits – was confirmed by running the algorithm on a large number of samples.

As previously mentioned, the results obtained here could not be compared to earlier research findings, so the detailed run results were investigated on their own to corroborate our statements. These values showed that the set goals have been achieved, and that EGAL+ is viable, efficient, robust and solves the specific problem described in the Introduction.

5 Conclusions and future improvements

When the algorithm was created, the goal was to generate more subsets of predefined tasks (i.e. exercises) to test students' knowledge in such a way that the quality of these subsets should be good enough according to a predefined quality matrix (i.e. should cover the main topics of a course as much as possible) and exercises should be diverse (according to a matrix and a diversity measure, which are inserted into the multi-objective fitness function). Moreover, although the difficulty of these tasks might be different from each other, the difficulty values of the subsets needed to be equal.

In this paper it has been shown that, according to the initial hypothesis, this algorithm could be made available for use by end users without a computer science and optimisation background subject to certain restrictions and improvements. The improved metaheuristic algorithm (EGAL+) was created and presented in this study. The hypothesis for this improved algorithm was confirmed by running it on a large number of samples.

Many improvements for this algorithm are planned in the future. A self-learning EGAL+ algorithm is planned to be developed in the future, where the difficulty values are updated using the results gained from users. It is also planned to modify the representations of the individuals in a certain way, with an expected decrease in run times. Furthermore, it is to be examined how this algorithm could be integrated into a learning content management system, for example into Moodle. The parameter calibration problem, as mentioned earlier, is also worthy of future investigation. Finally, it is also worth examining how the "group problem" and the "precedence relation problem" mentioned in [28] and [29] could be integrated into the algorithm.

6 References

[1] Son Ngo Tung, Jaafar, J. B., Aziz, I. A., Hoang Giang Nguyen, & Anh Ngoc Bui. (2021). Genetic Algorithm for Solving Multi-Objective Optimization in Examination Timetabling Problem. International Journal of Emerging Technologies in Learning, 16(11): 4–24. https://doi.org/10.3991/ijet.v16i11.21017

[2] Hnida, M., Idrissi, M. K., & Bennani, S. (2018). Automatic Composition of Instructional Units in Virtual Learning Environments. International Journal of Emerging Technologies in Learning (IJET), 13(06): 86–100. https://doi.org/10.3991/ijet.v13i06.8107

[3] Tu, C., Liu, Y., & Zheng, L. (2021). Hybrid Element Heuristic Algorithm Optimizing Neural Network-Based Educational Courses. Wireless Communications & Mobile Computing, 1–12. https://doi.org/10.1155/2021/9581793

[4] Lewis, R. (2012). A time-dependent metaheuristic algorithm for post enrolment-based course timetabling. Annals of Operations Research, 194(1): 273–289. https://doi.org/10.1007/s10479-010-0696-z

[5] Vaziri, S., Zaretalab, A., & Sharifi, M. (2020). Development of multi-objective simulated annealing based decision support system for course timetabling with consideration preferences of teachers and students. Muṭāliāt-i Mudīriyyat-i Ṣaṅatī, 17(55): 35–64. https://doi.org/10.22054/JIMS.2019.29331.1982

[6] Al-Betar, M. A., Khader, A. T. , & Zaman, M. (2012). University course timetabling using a hybrid harmony search metaheuristic algorithm. IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews, 42(5): 664–681. https://doi.org/10.1109/TSMCC.2011.2174356

[7] Xiangliu Chen, Xiao-Guang Yue, Rita Yi Man Li, Ainur Zhumadillayeva, & Ruru Liu. (2021). Design and Application of an Improved Genetic Algorithm to a Class Scheduling System. International Journal of Emerging Technologies in Learning (IJET), 16(01): 44–59. https://doi.org/10.3991/ijet.v16i01.18225

[8] Sukstrienwong, A. (2017). A Genetic-algorithm Approach for Balancing Learning Styles and Academic Attributes in Heterogeneous Grouping of Students. International Journal of Emerging Technologies in Learning, 12(3): 4–25. https://doi.org/10.3991/ijet.v12i03.5803

[9] Liping Wu. (2015). The application of Coarse-Grained Parallel Genetic Algorithm with Hadoop in University Intelligent Course-Timetabling System. International Journal of Emerging Technologies in Learning, 10(8): 11–15. https://doi.org/10.3991/ijet.v10i8.5206

[10] Alam, T., Qamar, S., Dixit, A., & Benaida, M. (2020). Genetic Algorithm: Reviews, Implementations, and Applications. International Journal of Engineering Pedagogy (IJEP), 10(6), 57–77. https://doi.org/10.3991/ijep.v10i6.14567

[11] Nentwich, V., Fischer, N., Sonnenbichler, A. C., & Geyer-Schulz, A. (2016). Computer aided exercise generation: A framework for human interaction in the automated exercise generation process. ICETE 2016 - Proceedings of the 13th International Joint Conference on e-Business and Telecommunications, 2: 57–63. https://doi.org/10.5220/0005947700570063

[12] Sadigh, D., Seshia, S. A., and Gupta, M. (2012). Automating exercise generation: A step

towards meeting the MOOC challenge for embedded systems. In Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education, pp 1–8, Tampere. ACM. https://doi.org/10.1145/2530544.2530546

[13] Arianna Zanetti, Elena Volodina, & Johannes Graën. (2021). Automatic Generation of Exercises for Second Language Learning from Parallel Corpus Data. International Journal of TESOL Studies, 3(2): 55–70. https://doi.org/10.46451/ijts.2021.06.05

[14] Eryiğit, G., Bektaş, F., Ali, U., & Dereli, B. (2021). Gamification of complex morphology learning: the case of Turkish. Computer Assisted Language Learning, 1–29. https://doi.org/10.1080/09588221.2021.1996396

[15] Malafeev, A. (2014). Language Exercise Generation: Emulating Cambridge Open Cloze. International Journal of Conceptual Structures and Smart Applications (IJCSSA), 2(2): 20–35. https://doi.org/10.4018/IJCSSA.2014070102

[16] Almeida, J. J., Araujo, I., Brito, I., Carvalho, N., Machado, G. J., Pereira, R. M. S., & Smirnov, G. (2013). Math exercise generation and smart assessment. 2013 8th Iberian Conference on Information Systems & Technologies (CISTI). pp 1–6.

[17] Gómez-Abajo, P., Guerra, E., De Lara, J., & Merayo, M. G. (2018). A tool for domain-independent model mutation. Science of Computer Programming, 163: 85–92. https://doi.org/10.1016/j.scico.2018.01.008

[18] Pilán, I., Volodina, E., & Borin, L. (2016). Candidate sentence selection for language learning exercises: from a comprehensive framework to an empirical evaluation. Traitement Automatique Des Langues, 57(3): 67–91.

[19] García, I., Benavides, C., Alaiz, H., & Alonso, A. (2013). A Study of the Use of Ontologies for Building Computer-Aided Control Engineering Self-Learning Educational Software. Journal of Science Education & Technology, 22(4): 589–601. https://doi.org/10.1007/s10956-012-9416-6

[20] Chrysafiadi, K., & Virvou, M. (2013). Student modeling approaches: A literature review for the last decade. Expert Systems with Applications, 40(11): 4715–4729. https://doi.org/10.1016/j.eswa.2013.02.007

[21] Adamopoulos, P. (2013). What makes a great MOOC? An interdisciplinary analysis of student retention in online courses. In Thirty Fourth International Conference on Information Systems, pp 1–21.

[22] Bouarab-Dahmani, F., Si-Mohammed, M., Comparot, C., Charrel, P.-J. (2011). Adaptive Exercises Generation Using an Automated Evaluation and a Domain Ontology: The ODALA+ Approach. International Journal of Emerging Technologies in Learning (IJET), 6(2): 4–10. https://doi.org/10.3991/ijet.v6i2.1562

[23] Basse, A., Diatta, B., Ouya, S. (2021). Ontology-Based System for Automatic SQL Exercises Generation Advances in Intelligent Systems and Computing, 1192 AISC, pp. 738-749.

[24] Ono, K., & Konaka, E. (2017). Automatic exercise generation and their equating on a coursework of differential equations. 2017 56th Annual Conference of the Society of Instrument and Control Engineers of Japan, SICE 2017, 2017–November. pp151–156. https://doi.org/10.23919/SICE.2017.8105450

[25] Mehta, S., & Smetannikov, I. (2020). Finding the Blank with Sequence Labeling for English Learning. 2020 International Conference on Control, Robotics and Intelligent System. https://doi.org/10.1145/3437802.3437834

[26] Almeida, J. J., Araujo, I., Brito, I., Carvalho, N., Machado, G. J., Pereira, R. M. S., & Smirnov, G. (2013). PASSAROLA: High-order exercise generation system. 2013 8th Iberian Conference on Information Systems & Technologies (CISTI). pp 1–5.

[27] Láng, B.; Kardkovács, T. Zs. (2016). Solving exercise generation problems by diversity oriented meta-heuristics, In: Shuang, Cang; Yan, Wang (eds.) SKIMA: 2016 10th International Conference on Software, Knowledge, Information Management & Applications: University of Information Technology, December 15- 17 2016, Chengdu,

China, pp. 49-54, https://doi.org/10.1109/SKIMA.2016.7916196
[28] LÁNG, B. (2019). Solving Exercise Generation Problems Using the Improved EGAL Metaheuristic Algorithm. SEFBIS Journal, 13: 23–31.
[29] Láng, B. (2020). Solving Exercise Generation Problems Using the Improved EGAL Metaheuristic Algorithm with Precedence Constraints (Vol. 1135). Springer International Publishing. 569-579 https://doi.org/10.1007/978-3-030-40271-6
[30] Lee, K. S., & Geem, Z. W. (2005). A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice. Computer Methods in Applied Mechanics and Engineering, 194(36): 3902–3933. https://doi.org/10.1016/j.cma.2004.09.007
[31] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. Computers and Operations Research, 13(5): 533–549. https://doi.org/10.1016/0305-0548(86)90048-1

7 Authors

Blanka Láng is an associate professor at Corvinus University of Budapest. Her research area are harmony and genetic metaheuristic algorithm development, linear regression model selection, exercise generation problems.

Balázs Dömsödi is an MSc Student in Business Informatics Engineering at Corvinus University of Budapest. His research interests are harmony metaheuristic algorithm development and exercise generation problems (email: balazs.domsodi@stud.uni-corvinus.hu).

Article submitted 2021-11-05. Resubmitted 2022-02-14. Final acceptance 2022-02-14. Final version published as submitted by the authors.