ON THE PROBLEM OF GENERATING A LARGE NUMBER OF COMPARABLE TEST VARIANTS

Mikuláš Gangur[1], Miroslav Plevný[2]✉

[1]Faculty of Economics, University of West Bohemia in Pilsen, Plzeň, Czech Republic

✉[2]Faculty of Economics, University of West Bohemia in Pilsen, Plzeň, Czech Republic, plevny@kem.zcu.cz

Highlights

- *The algorithm of generation a large number of content identical test variants in acceptable time is presented*
- *The tests may consist of more questions stemming from various thematic fields*
- *The content of individual test variants is identical but the similarity (congruence) is minimal*
- *The time of generation was shortened from hours to seconds in complex cases*

Abstract

The paper presents a possible way of solving the problem of creating more test variants for a large number of students divided into groups. The proposed solution may consist in introducing a parameterized automatic test generator. The principle of an automatic parameterized test generator is shown. The process of the question tree construction according to the increasing numbers of question in the banks of the particular subjects leads to a combinatorial explosion. This often results in excessive time of generation of the different variants of tests. To solve this problem, a heuristic method based on a pre-processing stage that precedes the construction of the searching tree is proposed. Further, the results of the experiments comparing the time of the test generation and the congruence of the test variants generated by the algorithm either using or non-using this heuristics are presented. According to these results the use of the generator with the proposed heuristics provides a considerably shorter time of generation, and the congruence of the generated test variants is even better in most cases.

Article type

Full research paper

Article history

Received: October 16, 2018
Received in revised form: December 4, 2018
Accepted: December 19, 2018
Available on-line: January 1, 2019

Keywords

automatic test generation, combinatorial explosion, evaluation methodologies, XSL transformation

Gangur M., Plevný M. (2018) "On the Problem of Generating a Large Number of Comparable Test Variants", *Journal on Efficiency and Responsibility in Education and Science, Vol. 11, No. 4*, pp. 78-84, online ISSN 1803-1617, printed ISSN 2336-2375, doi: 10.7160/eriesj.2018.110402.

Introduction

One of the most common forms of ascertaining (or measuring) the level of the gained knowledge is testing. Apart from the issue of the correct and well-balanced composition of the test, which we are not going to deal with herein, there is a specific issue of the need for designing a number of comparable variants for the same purpose of testing. A requirement calling for the creation of more variants of a test of the same type is quite common in this respect. This may be the case of testing the knowledge of a large number of students divided into groups. The same need arises when both the tests and the corrective tests are prepared simultaneously, or when the tests are repeated with a time delay. Another case is testing the knowledge in an e-learning environment where every student handles the test in a different time and it is therefore necessary to create a variant of the test for each student individually (Rosman and Buřita, 2014).

The problem of compilation of such a test set consists mainly in meeting the following requirements:

- All variants of the test should contain the same number of questions, as well as the same intensity expressed in points for each thematic field of the test;
- The similarity of the individual variants of the test should be minimal (a minimal congruence requirement).

Probably the biggest problem, when processing the necessary amount of the test variants manually (i.e. collection of tasks from each guarantee of an individual thematic field, the assembly of question groups for each individual variant of the test, etc.), consists in receiving the materials in different formats, in unbalanced difficulty of manually compiled variants of the test, and, apart from that, a time-consuming detection and correction of errors arising from editing the final form of the test. The use of MS Word templates for creating questions by the guarantees of thematic fields is seen as underperforming. Automatic generation of the test variants seems to be a possible solution to this problem.

Generating the necessary amount of variants of the given test may be facilitated by applying the computing technology. However, in connection with the above, it is necessary to solve a few problems. From the point of view of the generated tests mainly the following must be considered:

- guarantee of a comparable structure; and
- comparable level of difficulty.

The key issue of comparability of tests in terms of difficulty has been addressed in other publications. Authors Klůfa and Kaspříková (2012) use probability approach to solve the problem by binomial distribution and to answer the questions concerning the number of correct answers or the probability that the number of correct answers exceeds a given number. Similarly, Klůfa (2016) analyses the point number differences in the mathematics test among several variants of entrance examination test according to difficulty of variants. He studies how the results of entrance examinations depend on test variants.

For mentioned purpose the computing technology has been using for a long time. In the era of the Internet and the on-line technologies another significant advantage of using the computing technology for the purpose of testing appeared, namely the remote knowledge verification using on-line connection (Hürst, Jung and Welte, 2007; Niazi and Mahmoud, 2000).

The objective of this article is to present a proposal of the system for the automatic test generation according to the set requirements related to the structure (score, number of questions, issues tested) and the requirements concerning the

mutual relations of the tests, i.e. the disparity of their contents. The algorithm of the test construction is described and a method for a solution of the problem of the combinatorial explosion during the question tree construction is proposed. The proposal of the above generator is based on a few input requirements for the basic characteristics and the generator functionality. These requirements are described in detail by (Gangur, 2014). The most important requirements are:

- the need for the existence of a simple control mechanism when setting parameters and generating tests;
- the possibility of the text contents structure parameterization, i.e. the possibility of determining the total score in the test and the number of questions stemming from the individual thematic fields;
- the possibility to ensure the minimal congruence of the test contents with regard to the random choice of questions from the question bank.

The above requirements were the starting points in the process of searching for such a system or, as the case may be, in applying the principles of the already designed and published systems.

In the next section of this paper the related work is presented, the functionality of the generator is briefly introduced, and the following parts describe the methods and algorithm of the generator implementation as depth/first tree searching with a backtracking mechanism. Then, our new proposal of a heuristic method for the solution of the combinatorial explosion is introduced. This explosion results from the question tree construction with respect to the number of questions. The use of the aforementioned generator remains only in theory without an application of the designed heuristics, due to the time-consuming calculation. Using this heuristics a gap between the theoretical and practical utilization of the generator is overcome, enabling the real use of the generator in practice. Finally the outcomes of experiments of the test generating process are presented while the results of the processes either using or non-using this heuristics are compared.

Materials and Methods

Related work

The issue of the automated test generating has been dealt with in a number of publications; see e.g. Brusilovsky and Pathak (2002), Sung, Lin and Chen (2007) or Zeng et al. (2013).

The similar difficulty of each variant is considered as a key issue of test variants design. Contributions dealing with this problem use various approaches to solve it. For example Klůfa and Kaspříková (2012) reflect the results of statistical analysis using probability for evaluation of appropriateness of test variants. Foltýnek (2009) applied another approach that enables to compare test variants results according to different difficulty level using scoring process and correctness coefficients.

Automated creation of adaptive tests with regard to the level of knowledge of the individual students is an independent field in which intensive research is being carried out (Mine, Shoudai and Suganuma, 2000; Kapusta, Munk and Turčáni, 2010). Fakhrusy and Widyani (2017) developed Moodle plugins to generate quiz as a part of LMS using genetic algorithm. Nuthong and Witosurapot (2017) focused on diverse difficulty of quizzes and proposed the 5-level difficulty ranking score using a hybrid similarity measurement approach to increase the number of usable generated quizzes and their sensible generation.

Seemingly simple issues of the automated test generation controlled by parameterized requirements concerning the test structure are not paid so much attention with regard to the quantity of publications on this topic. Authors Yang, Wu and Wang (2008) proposed and implemented a robust system with adaptive elements for administration and a follow up selection of the test questions from the database with regard to the previous test results is described. The system enables a random choice of the test questions with regard to the set parameters, such as the percentage of the required type of questions (e.g. multi-choice, open questions) or the fields (knowledge points) out of which questions are selected. In the key issue of the choice of questions the system uses a complicated mechanism of arithmetic calculations which ensures meeting the set requirements for the test structure. The system is extensive and with regard to the process of choosing the questions and feeding the question database it may seem difficult for the users.

None of the above systems deals with the problem of insertion and namely the typesetting of the mathematical text. The authors Tomas and Leal (2013) deal with the issues of the mathematical text by means of an external application. So, as to finalize the creation of the tests, the above authors use some functionalities of a web application for the presentation and evaluation of the mathematical expressions.

The above described systems meet the basic requirements for the test creation from the randomly chosen questions with regard to the set parameters of difficulty and coverage of various fields of issues to be studied. These are complex and extensive systems covering a number of other functionalities and requiring a time consuming creation of a question bank. In most cases, generators do not deal with the issue of the mathematical text typesetting and they are not quite flexible in the matter of the choice of the generated tests output format. The majority of the above stated tools as well as other examined instruments only generate online web tests.

The functionality of the generator

In this chapter we will discuss the functionality of the proposed automatic test generator along with methods and algorithms of the implementation of such a generator. Attention will be paid to the solution of the combinatorial explosion with focus on the mutual congruence of the generated tests.

Let us, first of all, describe the final product of the process of generation, which is the necessary amount of variants of the required test. The input data here are the ***source questions*** in the ***question bank***. The following attributes have to be entered for each question:

- ***thematic field*** – a thematic area related to the given question; for each thematic field the required number of questions and the overall number of points for this field must be entered as the input parameters of the process of generation;
- ***score*** - number of points awarded if the answer is correct, this value should express the difficulty of a question;
- ***group*** - it determines whether a question is or is not incorporated in the test in context with other questions – for more detail see following subchapter).

Resulting test consists of questions generated within the individual *thematic fields.*

The functionality of the generator is controlled by a set of input parameters. We used the following parameters as the basic input generator parameters which then determine the system functionality:

- the total number of questions stemming from the individual thematic fields;
- the total score stemming from the individual thematic fields;
- the format of the resulting tests;

- the total number of the generated tests;
- the number of the tests in a package to be used simultaneously.

By means of the first two parameters it is possible to select the quantity of questions stemming from the given field in the entire test and at the same time to select its difficulty by a suitable combination of the number of questions and the score for the given thematic field. The generator supports the distributed creation of the individual fields of questions by various creators who can save the final question bank in an online repository of questions used by the generator. The administrator then controls the final tests generation. This approach enables, in some special cases (entrance tests and such like), hiding the contents of the complete test from the individual creators and letting only one authorized person create the test.

The strength of the generator consists in the possibility of selecting a template for generating the required output format. The test itself is generated in the proposed universal XML format, and by means of the XSLT processor it is, with the help of the inserted transformation template, transformed into the required output format (Kosek, 2013). The possibility of selecting this output format is flexible and it enables the user to create his/her own template and to generate his/her own output format (LaTeX, AcroTeX, Moodle XML).

The parameters determining the total number of the generated tests and the number of the tests in a package to be used simultaneously also control, among other things, the format of other generator outputs, namely the calculation of the mutual percentage congruence of the test variants, and the suggestion of the most suitable combinations of the test variants to be used simultaneously. The administrator, with the help of these suggestions, tries to compile the tests so that there are tests with the lowest level of congruence of questions between the individual rounds. The test questions are selected randomly and some questions, with regard to the required total number of questions in comparison with the number in the bank of a given field, may be repeated in the tests.

Even the question banks stemming from the individual fields can be listed in the generator outputs. The possibility of simple creation of such a question bank by means of freely available editors is one of the requirements for the generator functionality.

The control information related to the individual questions can be seen as another parameter influencing the test compilation. It determines, apart from the evaluation of a question by scoring, also listing a question in a group of questions. The group of questions enables similar questions not to be listed in one test and, at the same time, to list more questions with the joint settings in one group.

Another functionality of the generator considers congruence among test variants. As support for the prevention of undesirable cooperation among the examinees the generator considers the percentage congruence of tests and proposes combinations of the individual test variants to be grouped together. The percentage congruence of two variants is defined as a ratio of the number of identical questions in the considered variants and the total number of questions.

The proposal for the composition of variants in the individual rounds, i.e. the test packages, results from the requirement for the minimal congruence between the individual rounds. This limits the possibility of influencing the test as a result of possible communication of the examinees in the time gap between rounds when the examinees from one round may pass on as little information related to the particular questions as possible to the examinees in the following round.

The question structure and information control

The current version of the presented generator uses the Aiken question format (Aiken, 2013) and it can be extended by the possibility of the questions with a short or numerical answer and by the possibility of inserting more correct answers in case of the multiple response questions.

Each question is introduced by a tag with an abbreviation of the thematic field to which the question belongs (see 'OV' in the listing below). The tag contains control information influencing the listing of questions in the compiled test. Within this information the question bank creator determines the evaluation of the question by score and the group to which the creator lists the question.

```
    <OV score="2" group="381"> The objective function for achieving the highest total possible number of the manufactured products in the linear mathematical model of an optimization task for the above stated settings can have the following from:
    A) <math>\max z = \sum\limits_{i=1}^{n} w_i</math>
    B) <math>\max z = c_j \sum\limits_{i=1}^{n} w_i</math>
    C) <math>\max z = \sum\limits_{i=1}^{n} c_{ij} w_{ij}</math>
    D) <math>\max z = \sum\limits_{i=1}^{n} \sum\limits_{k=1}^ {p} b_{jk} w_j</math>
    E) <math>\max z = \sum\limits_{i=1}^{n} c_j w_j</math>
    ANSWER: A
```

The numerical code identifying the group is important. The digits of this code control the listing of the question in the stage of constructing the test according to the following scheme:

- Group 0 - the question can be listed without limitation;
- Group <1 – 99> - questions with the same number are not listed together in one test;
- Group <100 – 999> - group questions; mostly it is more questions with joint settings;
    - questions with the same first digit and different second digit are not listed in the same test;
    - questions with the same first and second digit belong to the same group and either all of them are listed in the test or none of them at all;
    - the last digit determines the order of questions in the group; the first one is often a question with the joint settings.

One of the other features of the generator is the possibility to insert the mathematical text into the text of a question or, as the case may be, also the exact listing (e.g. algorithm listing and such like) as well as a figure in the JPEG format as a complement to the task settings (Gangur, 2011; Gladavská and Plevný, 2014).

The algorithm of the test assembling

In case we require the compilation of the test out of the questions based on the set criteria and with regard to the question control information (group) the algorithm of the recursive depth-first search of tree is applied. It is the so called backtracking algorithm which selects, out of the questions for the given field, one or more questions (according to the group number) and it always checks whether the criteria of the total number of questions and the required total score are met. If one of these parameters is exceeded, it recurs by one question (more questions) and selects another one.

The core of this algorithm is the ***combination*** recursive function. The input in this function, when it is called for the first time, is the list of the question bank suitable for the given field. Random permutation and question selection is applied in case of this list and therefore the order of questions and the depth-first search of tree are always different. The following listing shows the headline of the applied function and its first call.

```
    function combination($list, $current_
list, $num_points, $deep)
    new_test_list = combination($question_
bank, Array(), 0, 0)
```

In case of further recursive calls of the function this list is entered in the function without the questions that had already been used. In this sense, the current list is also an input parameter in which selected questions are stored (in case of the first call the list is empty). Other parameters are: the score of the questions currently inserted into the test and the depth of tree which represents the number of questions in the compiled test. Upon the first call both the values are null. With each question (group of questions) being added the depth of tree gets higher.

Out of the list of free questions the recursive function call creates the rest of the list of the tested questions. The recursion ends upon achieving the required score and the number of questions for the given field. In case of the retrospective finishing of the individual calls of the function a list of the test questions is formed starting from the back and moving forward and at each level this tail of the list is added to the currently selected question or the question group and like this a new tail of the list is created at the given call level.

If, upon the function call, the values of score or number of questions are exceeded, the selected question (group of questions) is not accepted and another question in the list of free questions is chosen, until the list is empty. After that the algorithm recurs back by one level of the call (backtracking), and it selects another question out of the list of the free questions at the given level.

By means of the above described procedure of backtracking the depth-first search of tree is implemented. Upon returning back to the first call level the whole list of questions according to the set criteria is created if the finishing condition is met.

The process of depth/first searching algorithm according to the number of used questions leads to the combinatorial explosion and it takes too much time. When increasing the number of the source questions for a thematic field as well as the number of the demanded questions for this field the time demand of the generation process increases significantly. In many cases, this time is expressed in the order of tens of minutes (or hours sometimes). This can be very annoying for users, and therefore it is necessary to solve this problem. A proposal of a possible way to solve this problem is described in the next part.

The solution of the problem of combinatorial explosion

One of the possibilities to solve the problem of the combinatorial explosion of questions tree is to decrease the whole number of questions in one thematic field. This approach narrows the selection of different questions and increases the possibility of higher congruence among the generated tests.

We, on the other hand, propose the solution that also decreases the number of questions put in the process of depth-first tree searching, but the algorithm randomly selects these questions from the bank of all questions of one thematic field. If the assembling of the questions list for the generated test according to the input parameters setting is not successful, the process continues with the selection of new questions from the rest of questions in the bank. When the bank of questions for the considered field is emptied all questions are returned to the bank and the process starts again with the new selection.

In the preselection we take into account the exclusive questions with respect to the group number (1 – 99), and at the same time the preselection process controls the whole number of the demanded points for the field. This number of points is satisfied by the preselection of questions individually for sets of different *n*-points questions. For example 10 points can be assembled from four 3-points questions, five 2-points questions, and ten 1-point questions. In the same way the question list is constructed for every new test variant, and it contributes to the low congruence among different test variants.

The step of the random selection is implemented as the selection of all the remaining questions in the bank in the permuted order. This preselection has to reflect the group questions, i.e. if the one question of the group is selected the other ones have to be included to the selection. The described process has quadratic time complexity according to the number of the preselected questions and the number of all the source questions.

The algorithm of the preselection is described in the next steps. The input to this preselection process is a permuted list of the source questions.

Step 1. Take the input current list of the source questions as *pl*.

Step 2. Create an array of maximal numbers of questions for every *n*-point question according to the prescription *floor (demanded points for field / n)+1*.

Step 3. While selected number of *n*-point questions is not greater or equal to the maximal numbers of questions determined in step 2 for every *n* or all questions of list *pl* are checked do

- i. Take question from source list *pl* and fill *n* as a score of question.
- ii. If (number of *n*-question is less than the maximal number of *n*-questions) and group of question < 100 is not in the resulting question list *rl* then
  - a. if the group of question >= 100 then select all questions of the group and add them to question;
  - b. add question or all selected question in group to the resulting question list *rl*.
  - c. remove the selected question(s) from the list of the source questions *pl*.

Step 4. Return the resulting list of questions *rl*.

The output from the preselection is the list of questions and it is one of the input parameters for the *combination* function (see subchapter *The question structure and information control*). If the assembling of questions for the field is successful the selected questions are removed from the permuted list of the source questions and for assembling the next variant the

preselection process selects questions from the modified list of the source questions. If the assembling is not successful the list of the source questions is fulfilled with all the questions and the preselection is applied on this list. If the assembling is not successful on this new preselected list the generator stops without result.

The proposed heuristics not only decreases the time consumption but also, with respect to selected questions pools (that are mostly mutually exclusive), supports lower congruence among particular variants of the generated tests. The comparison of the results of the generation processes using or not using the proposed heuristics is presented in the next part.

Results of experiments

The outputs of the generating process in different output formats according to the input parameters are presented by (Gangur, 2014). The current version of the generator offers XSL templates mainly for the formats Moodle XML, LaTeX and AcroTeX (interactive PDF). In case of TeX format (LaTeX and AcroTeX) it is possible to create PDF documents by means of the post-processor methods.

Next the computing times and the congruence among generated variants are presented. The generator was used for the same source of questions with the same input parameters, i.e. the same number of the demanded questions for every field and the same demanded score for every field. In the first case the generator was implemented without the above described heuristics, and in the second case with this heuristics. The data in the table 1 show the numbers of question in the banks of the particular subjects (thematic fields) together with the values of the input parameters.

Particular subject (thematic field)	Bank size (No of source questions)	No of demanded questions	Demanded score of questions
Economics	176	10	20
Business economy	140	11	20
Management	81	5	10
Marketing	106	6	10
Business finance	70	5	10
Accounting	70	5	10
Management science	59	4	10
Statistics	69	5	10
Average	96.375	6.375	12.5

Table 1: Numbers of questions in banks and input parameters values, 2017 (source: own calculation)

The output coefficients of comparison are the congruence of variants and the speed (time complexity) of the generating process. In one experiment 10 or 5 test rounds are processed. The outputs of the experiment are the average time per one round (AT) for each compared algorithm (without heuristics - NH, and with heuristics - H), and the average coefficient of congruence among particular variants per one round (AC).

The input parameters for experiments are as follows:

- The number of test rounds.
- The number of the source questions represented as an average number of the source question per one thematic field (ANSQ). This number is determined by the selection of the configured percent part of the basic source files (see example of questions number in the table 1). More source questions generally imply larger computing time and the decrease of the resulting congruence among particular variants of test.
- The number of the demanded questions for every thematic field is represented by the average number of the demanded question per one field (ANDQ). These numbers are configured by the absolute number of questions for every field. As with the previous parameter ANSQ the higher number of the source questions generally imply the larger computing time and the decrease of the resulting congruence among particular variants of test.
- The number of the demanded variants of the test is configured as the absolute number of variants. In the experiments this number was set to 6 variants (see later).

Version	Algorithm with heuristics (H)						Algorithm without heuristics (NH)					
	A	B	C	D	E	F	A	B	C	D	E	F
A	-	0.00	0.00	0.00	0.00	0.00	-	5.88	7.84	7.84	1.96	5.88
B	0.00	-	0.00	0.00	0.00	0.00	5.88	-	0.00	3.92	3.92	1.96
C	0.00	0.00	-	0.00	0.00	0.00	7.84	0.00	-	3.92	1.96	9.80
D	0.00	0.00	0.00	-	0.00	0.00	7.84	3.92	3.92	-	7.84	5.88
E	0.00	0.00	0.00	0.00	-	0.00	1.96	3.92	1.96	7.84	-	13.73
F	0.00	0.00	0.00	0.00	0.00	-	5.88	1.96	9.80	5.88	13.73	-

Table 2: Values of mutual congruence using algorithm H and NH [%], 2017 (source: own calculation)

The values of both parts of the table 2 are summarized and represented by one coefficient of the average congruence. The lower value of the congruence means a lower level of mutual similarity of variants. In our case the values of this coefficient are 0.00% for the generation processes using the proposed heuristics (H), resp. 5.49% for the generation processes non-using the proposed heuristics (NH).

The outputs are the average congruence of one test round of the generator and the time of generating the demanded number of variants. The searching tree is constructed for every thematic field of every variant and that's why the complexity increases linearly according to the increasing number of the demanded variants. With respect to the number of the source questions the average congruence among variants increases when the demanded variants increase. If we process more than one generator round in one experiment it is possible to characterize the experiment results as the whole average congruence per one round and the whole average time per one round. The example of such an experiment is illustrated by the following table 3 where 100% of all questions were used, i.e. on average 96.375 questions per one field and on average 6.375 demanded questions per one field (see table 1). In table 3 the average congruence among variants in % for every round and both algorithms are stated in the first two rows. The computing time of generating the demanded variants in seconds for every round and both algorithms are presented in the last two rows. The last column shows the whole averages per one round.

	N	1	2	3	4	5	6	7	8	9	10	Avg.
Congruence [%]	H	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	NH	5.49	6.14	7.04	7.44	7.97	6.79	7.71	6.53	6.01	8.49	7.01
Time [s]	H	0.21	0.19	0.23	0.26	0.25	0.24	0.30	0.37	0.66	1.19	0.39
	NH	0.76	0.73	1.27	0.92	1.00	7.48	2.09	2.00	3.67	15.38	3.53

Table 3: Average congruence and the time of generation, 2017 (source: own calculation)

The described experiment can be characterized by the following values of the input and output parameters:

ANSQ= 96.375

ANDQ= 6.375

AC = 7.01% for NH, 0% for H

AT = 3.53 [s] for NH, 0.39 [s] for H

In the implemented experiments the values AC (table 4) and AT (table 5) were measured for different values of ANSQ from 100% to 20% of the basic file of questions and for the increasing values of ANDQ.

For every ANSQ and ANDQ the pair NH | H of values for the algorithm without heuristics and algorithm with heuristics is

presented. In every experiment 10 rounds for ANDQ=6.375 and 5 resp. 3 rounds for ANDQ=11.25 resp. ANDQ=20 were processed with respect to the time complexity of computing. If the test was not able to assemble according to the demanded criteria (the number of points and the number of questions for every thematic field) in the given round, the random selection form of the source questions was implemented again until the test was assembled or the experiment finished after the configured time (it was set to 2 hours in all the experiments provided).

ANSQ	ANDQ = 6.375		ANDQ = 11.25		ANDQ = 20.00	
	NH	H	NH	H	NH	H
96.375 (100%)	7.15	0.00	Inf	4.83	Inf	21.54
86.50 (90%)	7.97	0.00	23.53	11.16	Inf	25.60
76.75 (80%)	8.34	0.14	19.04	13.35	Inf	35.98
67.25 (70%)	10.57	0.80	21.91	16.31	Inf	x
57.50 (60%)	11.88	1.96	24.88	20.18	Inf	x
48.00 (50%)	15.20	4.30	31.14	29.36	Inf	x
38.25 (40%)	18.20	6.54	38.74	39.33	x	x
28.50 (30%)	25.45	17.41	x	x	x	x
19.00 (20%)	37.38	38.17	x	x	x	x

Marking used in the table 4 and 5:

x - *the test was not assembled with respect to the demanded criteria according to a small number of the source questions,*

Inf - *the computing finished according to overtime (2 hours).*

Table 4: Average congruence (AC) without heuristics (NH) and with heuristics (H) [%], 2017 (source: own calculation)

ANSQ	ANDQ = 6.375		ANDQ = 11.25		ANDQ = 20.00	
	NH	H	NH	H	NH	H
96.375 (100%)	1.03	0.16	Inf	0.30	Inf	187.13
86.50 (90%)	1.43	0.16	115.34	0.26	Inf	92.37
76.75 (80%)	1.01	0.15	111.95	0.23	Inf	5.37
67.25 (70%)	0.65	0.13	7.41	0.27	Inf	x
57.50 (60%)	0.83	0.12	7.28	0.25	Inf	x
48.00 (50%)	0.41	0.13	10.70	0.20	Inf	x
38.25 (40%)	0.27	0.10	0.72	0.27	x	x
28.50 (30%)	0.20	0.12	x	x	x	x
19.00 (20%)	0.13	0.10	x	x	x	x

Table 5: Average time (AT) without heuristics (NH) and with heuristics (H) [s], 2017 (source: own calculation)

Discussion

As it is apparent from the values in the tables 4 and 5 for the increasing number of the demanded questions (ANDQ > 6.375) and the decreasing number of the source questions the assembling of the test was not successful in the configured max time of the process. The presented values depend on the structure of the source questions in the thematic field according to the enlistment of questions to the groups and the mutual excluding in one test as well as the number of the tied questions and the tied groups. That's why it is suitable to compare the obtained values for NH algorithm and H algorithm that were computed in relation to the same data, i.e. to the same structure of the source questions.

Despite the previous statement it is obvious that with increasing ANDQ the congruence increases as well as the time complexity. Especially the time complexity can be characterized as $O(K(\mathrm{ANSQ}, \mathrm{ANDQ}))$, where K is the binomial coefficient, i.e. the time complexity depends above all on the number of the source questions (ANSQ) and the number of the demanded questions (ANDQ). The proposed heuristics decreases the number of the source questions in the process of the searching tree construction and this way the time complexity of the generation process decreases, but it does not increase the congruence. By contrast, in most comparisons the final congruence of the test generation the algorithm with heuristics is lower than the algorithm without heuristics.

The advantage of the proposed and in practice applicable generator is a simple system of parameter setting that controls generation in terms of difficulty and number of questions from each of the areas considered. Another advantage is the simple format of source texts for individual questions and thus a simple process of creating questions without the need to use a more complex SW. Optionally, it is possible to use already prepared questions that may be in plain text format. Among the undeniable advantages over similar systems is the choice of different output test formats. Tests can be generated in pdf format suitable for a written test printing, in html format for placing on the web for online testing or into the LaTeX structure for further processing.

As limitations of the proposed solution, the impossibility to value the individual questions in view of the difficulty of other tasks in the final test may be considered. Each question is rated by a specific point value within a given area. On one side, the comparative difficulty level of each variant of the test is determined by the described approach of assembling the tests according to the given number of questions and the total number of score points representing difficulty from the given thematic field. On the other side, this assumes a correct assessment of the difficulty of questions on a scale of 1-3 points by the authors of questions from each individual field. The overall difficulty of variants of the entire test can be compared only with regard to this evaluation of individual questions by the good fit tests. Equally, the different level of students' knowledge is not considered by these tests (Klůfa, 2016). Compared to Klůfa and Kaspříková (2012), the results of previous tests are not taken into account by the proposed method, and the structure of the test is not modelled using estimated probability distributions.

Similarly, the possibility of more sensitive control of the difficulty of individual tasks is limited. All questions are of the MCQ type and are designed by the designers of the questions as static from the domain. Therefore, the system does not allow to variate the distractors (bad answers) dynamically with respect to adjusting the difficulty by choosing ontologically close distractors. The maximal number of questions in the area is limited also with regard to the system of grouping questions, which controls the placement of questions in the test within the required context. However, the proposed algorithm for classifying questions into groups is versatile, allowing the extension of the grouping interval used and thus increasing the maximal number of questions for the given area.

In the future, we consider extending the functionality of the test generator by the implementation the option into the generator to use not only statically specified but also parameterized questions.

Furthermore, we will focus on the issue of comparable difficulty level of individual variants both from the point of view of individual students' results and from the point of view of measurement and comparison of the difficulty of individual variants among themselves. This would extend the use of the generator significantly, and partially eliminate some of the above limitations.

Conclusions

The system described in this paper is a useful aid, namely in the process of preparing tests stemming from various fields, for example from the sphere of the entrance examination procedure applied in the authors' institution. When using a larger number of sources or demanded questions the original algorithm non-using the heuristics described above does not allow for assembling the demanded test variants in an acceptable time. On the contrary,

the proposed heuristics allows the use of the generator in the real time.

The paper describes the functionality and the process of implementation of the generator of questions stemming from the set fields by means of a combination of questions according to the set criteria, i.e. namely the number of questions in the test for the given thematic field and the overall scoring of questions in the given field.

The solution of the generation process is based on the construction of a searching tree and its depth-first searching combined with the backtracking algorithm. The main input parameters are the number of source questions and the number of the demanded questions. These parameters most influence not only one of the most important final outputs of the generation process, i.e. the mutual congruence among particular variants of the test, but also the time complexity of the generation process. The depth-searching of the tree results in the combinatorial explosion and high increasing of time complexity. In some configurations of the input parameters, especially the source questions and the demanded questions, the generation process does not finish within the required maximal time limit of 2 hours. That is the reason why to propose the pre-process of preselection of a limited smaller number of questions without placing them back to the source questions. This preselection process also respects some input criteria, and it is able to satisfy them in the quadratic time complexity. If the generating process is unsuccessful because of the empty list of the source questions or a small number of questions in the list, the generator starts the process again with a new selection from all the newly permuted source questions. The proposed improving of algorithms decreases the complexity of computing namely with respect to the increasing number of the demanded questions, and contributes to the low congruence among different test variants. The mutual congruence of variants generated by the algorithm using the proposed heuristics is at least as good (and mostly even better) as the congruence of the test variants generated by means of an algorithm non-using this heuristics.

Acknowledgment

This work was supported by the University of West Bohemia in Pilsen under Grant No. SGS-2018-042.

References

Aiken (2013) *MoodleDocs: Aiken format.* [Online], Available: http://docs.moodle.org/22/en/Aiken_format [26 Jul 2016].

Brusilovsky, P. and Pathak, S. (2002) 'Assessing Student Programming Knowledge with Web-based Dynamic Parameterized Quizzes'. A*ssociation for the Advancement of Computing in Education (AACE)*, pp. 1548-1553.

Fakhrusy, M.R. and Widyani, Y. (2017) 'Moodle plugins for quiz generation using genetic algorithm'. *2017 International Conference on Data and Software Engineering (ICoDSE)* Palembang, Indonesia, INSPEC Accession Number: 17578381. http://dx.doi.org/10.1109/ICODSE.2017.8285882

Foltýnek, T. (2009) 'A New Approach to the Achievement Test Items Evaluation: the Correctness Coefficients'. *Journal on Efficiency and Responsibility in Education and Science*, vol. 2, no. 1, pp. 28-40.

Gangur, M. (2011) 'Automatic generation of cloze questions'. C*SEDU 2011 - Proceedings of the 3rd International Conference on Computer Supported Education*, Portugal, SciTePress - Science and Technology Publications, pp. 264-269.

Gangur, M. (2014) 'Automatic Parameterized Generation of Test'. *DIVAI 2014: 10th International Scientific Conference on Distance Learning in Applied Informatics*. Prague, Wolters Kluwer, pp. 55-64.

Gladavská, L. and Plevný, M. (2014) 'Problems of automatic generation of questions for the purpose of testing the knowledge in a management science course'. *DIVAI 2014: 10th International Scientific Conference on Distance Learning in Applied Informatics.* Prague, Wolters Kluwer, pp. 325-335.

Hürst, W., Jung, S. and Welte, M. (2007) 'Effective learn-quiz generation for handheld devices'. Pro*ceedings of the 9th international conference on Human computer interaction with mobile devices and services*, ACM, pp. 364-366.

Kapusta, J., Munk, M. and Turčáni, M. (2010) 'Evaluation of adaptive techniques dependent on educational content'. *4th International Conference on Application of Information and Communication Technologies, AICT2010*, INSPEC Accession Number: 5611791. http://dx.doi.org/10.1109/ICAICT.2010.5611791

Klůfa, J. and Kaspříková, N. (2012) 'Multiple Choice Question Tests for Entrance Examinations - A Probabilistic Approach'. *Journal on Efficiency and Responsibility in Education and Science*, vol. 5, no. 4, pp. 195-202. http://dx.doi.org/10.7160/eriesj.2012.050402

Klůfa J. (2016) 'Comparison of the Test Variants in Entrance Examinations'. *Journal on Efficiency and Responsibility in Education and Science,* vol. 9, no. 4, pp. 111-116, http://dx.doi.org/10.7160/eriesj.2016.090404

Kosek, J. (2013) *XML for everyone.* [Online], Available: http://www.kosek.cz/xml/index.html [20 May 2016].

Mine, T., Shoudai, T. and Suganuma, A. (2000) 'Automatic Exercise Generator with Tagged Documents Considering Learner's Performance'. *Association for the Advancement of Computing in Education (AACE)*, Chesapeake, VA, pp. 779-780.

Niazi, R. and Mahmoud, Q.H. (2000) 'A Web-based Tool for Generating Quizzes for Mobile Devices'. *A poster at the 39th ACM Technical Symposium on Computer Science Education (SIGCSE).* Portland, OR.

Nuthong, S. and Witosurapot, S. (2017) 'Enabling fine granularity of difficulty ranking measure for automatic quiz generation'. *9th International Conference on Information Technology and Electrical Engineering (ICITEE)*, Phuket, Thailand, pp. 1-6. http://dx.doi.org/10.1109/ICITEED.2017.8250492

Rosman, P. and Buřita, L. (2014) 'Concept of the computer science course and some aspects of ICT integration into education'. *E&M Ekonomie a management*, vol. 17, no. 3, pp. 169-180. http://dx.doi.org/10.15240/tul/001/2014-3-013

Sung, L.-C., Lin, Y.-C. and Chen, M.C. (2007) 'An Automatic Quiz Generation System for English Text'. *7th IEEE International Conference on Advanced Learning Technologies, Proceedings*, Niigata, Japan, INSPEC Accession Number: 9868711. http://dx.doi.org/10.1109/ICALT.2007.56

Tomas, A.P. and Leal, J.P. (2013) 'Automatic generation and delivery of multiple-choice math quizzes'. P*rinciples and Practice of Constraint Programming. CP 2013. Lecture Notes in Computer Science,* Vol. 8124. Springer, Berlin, Heidelberg, pp. 848-863. http://dx.doi.org/10.1007/978-3-642-40627-0_62

Yang, A., Wu, J. and Wang, L. (2008) 'Research and design of test question database management system based on the three-tier structure'. *WSEAS Transactions on Systems*, vol. 7, no. 12, pp. 1473-1483.

Zeng, J., Sakai, T., Yin, C., Suzuki, T. and Hirokawa, S. (2013). 'Automatic generation of tourism quiz using blogs'. *Artificial Life and Robotics*, vol. 17, no. 3-4, pp. 412-416. http://dx.doi.org/10.1007/s10015-012-0076-7