International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:10, No:7, 2016

Reusing Assessments Tests by Generating Arborescent Test Groups Using a Genetic Algorithm

Ovidiu Domşa, Nicolae Bold

***Abstract*—**Using Information and Communication Technologies (ICT) notions in education and three basic processes of education (teaching, learning and assessment) can bring benefits to the pupils and the professional development of teachers. In this matter, we refer to these notions as concepts taken from the informatics area and apply them to the domain of education. These notions refer to genetic algorithms and arborescent structures, used in the specific process of assessment or evaluation. This paper uses these kinds of notions to generate subtrees from a main tree of tests related between them by their degree of difficulty. These subtrees must contain the highest number of connections between the nodes and the lowest number of missing edges (which are subtrees of the main tree) and, in the particular case of the non-existence of a subtree with no missing edges, the subtrees which have the lowest (minimal) number of missing edges between the nodes, where a node is a test and an edge is a direct connection between two tests which differs by one degree of difficulty. The subtrees are represented as sequences. The tests are the same (a number coding a test represents that test in every sequence) and they are reused for each sequence of tests.

***Keywords*—** Chromosome, genetic algorithm, subtree, test.

I. Introduction

THE usage of informatics notion in different domains is not a novel approach and the results of this usage are known and beneficial to the domain. The inclusions of informatics notions in education can be classified in inclusions on methods and methodologies and inclusions on education itself (the actual process of education and, particularly, in the process of assessment). New methods and techniques of assessment with higher relevance and better effects on a longer term on the process of learning have been studied and discovered, such as the ones presented in [1]. However, due to the nature of the human learning, the usage of these technologies is somehow controversial, numerous papers taking into account the effects of this inclusion in the educational domain and in the assessment process. Some of them study the students and teachers perceptions over online evaluation [2], [3]. Other directions of research study the effects of ICT on learning and evaluation [4] or comparisons between traditional methods of assessment and the ones based on new technologies [5]. Other papers present a new approach of ICT inclusion in education: styles in [6] and [7] and models of an e-learning platform in [8] and [9]. New strategies and approaches on the assessment process are presented in [10] and [11].

These effects refer to the mechanisms of these processes and less to the processes themselves. Thus, in the case of this paper, the accent falls on the generation of tests and not on the tests themselves or the modalities of testing. Even if the paper does not focus on the assessment process itself, the focus on the mechanism (the generation) influences the process of assessment and the evaluation itself. In these cases, such as the one presented in this paper, the benefits are obvious, because the manual selection of tests would consume time and energy. This inclusion is applied to a mechanism and not to the way of evaluation.

This paper brings into discussion the generation of tests from a battery of arranged in a tree structure based on their degree of difficulty. The tests are reused, in the way that a number coding a test represents it for all the sequences of tests. In this matter, the same tests are used to build numerous sequences of tests. The algorithm used for solving this issue output firstly the distinct sequences of tests directly related with edges. If such a sequence does not exist, the algorithm finds the sequences which have the lowest number of missing edges between the tests, this number consisting in the fitness function. The sequences of tests are output in the form of sequences of numbers, each test being codified by a number from 1 to the total number of tests and being reused for every sequence. A similar problem was described in the paper [12], but with a different fitness function and another type of algorithm.

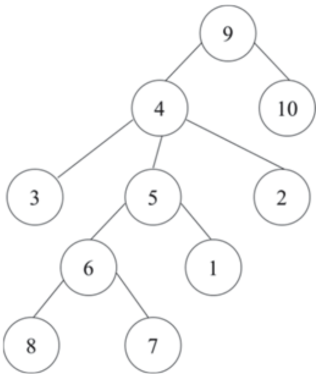


Fig. 1 Example of a tree used in the algorithm showing the arborescent structure of the relations between the tests

Ovidiu Domşa is with “1 Decembrie 1918” University of Alba Iulia, Romania (e-mail: domsaddd@yahoo.com).

Nicolae Bold is with the University of Agronomic Sciences and Veterinary Medicine Bucharest, Faculty of Management, Economic Engineering in Agriculture and Rural Development, Slatina Branch, Romania (e-mail: bold_nicolae@yahoo.com).

The arborescent structure was chosen to show the classification of tests on degrees of difficulty. This kind of structure is very useful in these cases of arborescent relations between the components of the structure and can lead to fast and reliable solutions for a problem waiting to be solved. A genetic type of algorithm was chosen for solving the purposed problem because of its runtime, its lower usage of resources and its more convenient modality of outputting the solutions.

Theories and findings from informatics domain have an extraordinary versatility and a multitude of appliances in various domains. If we refer strictly to the types of structures used in this paper, we will give examples for tree structures and genetic algorithms.

For example, trees are used in a variety of fields, such as management, education, sorting and searching [13], the process of decision [14] and other domains which use arborescent structures. Due to the nature of the arborescent structures, the trees are used in cases of hierarchical structures, where the elements are in a relation of subordination or succession or classified by different criteria (degree of difficulty, number order etc.).

As for genetic algorithms, their usage extends more and more in a multitude of areas. This fact can be deducted from their reliability and the similarities between the structures used within the algorithm and the ones within the issues wanted to be solved, as well as for their high probability of outputting in a reasonable time more accurate solutions for large sets of input data. Thus, the genetic algorithms are used in network-related issues (traffic [15], IT and Internet), design and artistic domains ([16] and [17]), web applications [18] and [19], chemistry [20], agriculture [21], education and scheduling [22] etc.

II. Defining the Problem

The paper studies a quite challenging problem, given the complexity of the used algorithm. Given a number of tests grouped in a battery of tests used for evaluation (the total set of tests is called in the paper the battery of tests) with different degrees of difficulty, thus forming an arborescent structure, the algorithm presented in this paper searches a subtree formed from a given number of nodes which keeps the property of arborescence. This property means that the nodes (tests) are related to each other in a connection of difficulty (a difficulty order, which means that an edge connects one test with a more difficult test). A condition of this search is that a node can be reached only through its parent node.

This issue can be solved by coding each test with an integer from 1 to the total number of tests and applying the algorithm described in Section III. In order to fully understand the problem, we will take a short example. Given a total number of 10 tests and the parent node array T, alongside the number of the nodes wanted in the subtree, the user wants to find a subtree which can keep the property of arborescence. The algorithm generates all these types of subtrees and the ones which are close with that type of subtree (with one node missing, then two and so on). The subtree is graphically shown in Fig. 1.

The solutions are output in a form of sequences of numbers, which shows the nodes which form the subtrees. In the example, a generated sequence can be 4 5 6 9 8 or 1 2 4 9 10. In the first case, the sequence forms a subtree and all the five nodes form a subtree (0 nodes missing). For the second sequence, a single node is needed to form a subtree (the node 5), which means that 1 node is needed to form a subtree. These missing nodes are actually the values of fitness function for each sequence. These values are ordered and the first sequences are the ones with 0 nodes missing.

The generation can be made using several methods. But, given the fact that the problem is NP-complex, it can be solved using algorithms that are exponential. Although, the backtracking type of algorithm can be used only for small values (the number of nodes and the sequence dimensions less than 20), probabilistic algorithms such as genetic algorithms are preferred (as presented also in paper [23]). Section III presents an algorithm of this type.

III. The Algorithm of the Model of Generation with Restrictions of Tests Using Genetic Algorithms

As we presented earlier in the paper, the algorithm uses arborescent structures for defining the elements used and genetic notions for solving the problem proposed in the introduction. As any algorithm structure, the one presented in the paper need input data and output data, as well as sequences, structures and mechanisms used within the algorithm for solving the problem. These components will be presented in the next lines.

A. Structures Used within the Algorithm

The algorithm uses the number of the tests, the sequence dimension, the number of generations and the parent nodes array for outputting the sequences with the given conditions. Thus, the next variables and arrays are used:

- N (the number of tests within the battery);
- No (the sequence dimension);
- no_generations (the number of generations);
- k (number of distinct solutions wanted to be output);
- NrPop (the number of the chromosomes).

As for the arrays used within the algorithm, the next list presents them:

- T[N] (the parent nodes array);
- pop[NrPop][No] (the bi-dimensional array which contains the sequences);
- v[N] (binary array used for determining fitness values; contains the visited nodes from a path);
- w[N] (binary arrays used for determining fitness values; contains the nodes which are part of a sequence).

B. Genetic Algorithm Structures

Every genetic algorithm, regardless its form, has the same major structures: chromosomes, genes, operations and fitness function. A chromosome is actually a sequence of test which respects the given conditions. A gene is a test within a sequence of tests. A general form of a chromosome is presented in Fig. 2.

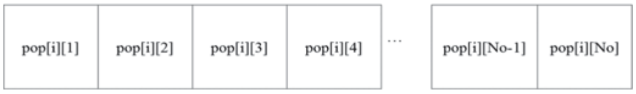


Fig. 2 The general form of a chromosome

The operations used in this algorithm are generation, mutation and crossover. The tests are randomly generated to form the initial population of chromosomes. The operation of generation is shown in Fig. 3. In this figure, the number P is in the set {1, 2, …, N} and must be different from the genes previously generated (P ≠ pop[i][j-1]; where i=1, NrPop, j=1, No).

The mutation operation consists in the random generation of a test (number) and the replacement in a previously randomly-generated sequence, if the test is different from the others within the sequence. The mutation operation is graphically presented in Fig. 4. A number M2 is randomly generated, different from the other genes in the chromosome, then the gene from the position M1 is replaced with M2.

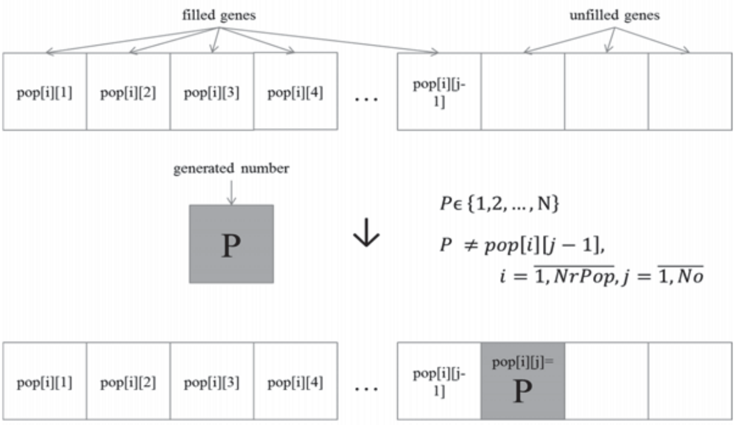


Fig. 3 Operation of generation for a chromosome (j ≤ No, P ≤ N)

The crossover operation consists in the formation of an enlarged chromosome from two chromosomes chosen randomly after the chromosomes are verified to be distinct and ordered. The resulted chromosome is then split in two new chromosomes which are added in the population. The crossover operation is graphically presented in Fig. 5. Two chromosomes with the property that one gene from a chromosome is not found in the second chromosome are generated. Then, an array with the elements from the two chromosomes is built, the values from this array are ordered and two new chromosomes are built: one from the first No elements and the second with the last No components.

The fitness function calculates the number of the missing edges from a generated subtree. In this case, the function is a minimal one, because the lowest the number of missing edges will be, the optimal solution will be output. The general form of the fitness function is:

$$f\ (pop[i][j]) = \sum_{j=1}^{No} v[pop[i][j]];\ i=\overline{1,NrPop} \qquad (1)$$

The array *v* is used for establishing how many and which nodes miss for the sequence to form a subtree. It has N elements, as the array *w*, and its elements can have two values: binaries 0 and 1. The array *w* contains how many and which nodes form a sequence and the array *v* contains the closest nodes from the sequence nodes that form a subtree (subtree which match with the sequence when the sequence forms a subtree), whose elements are 0 when the sequence forms a subtree. The mechanism of the function for a certain sequence $pop[i][j]$ is presented in Table I.

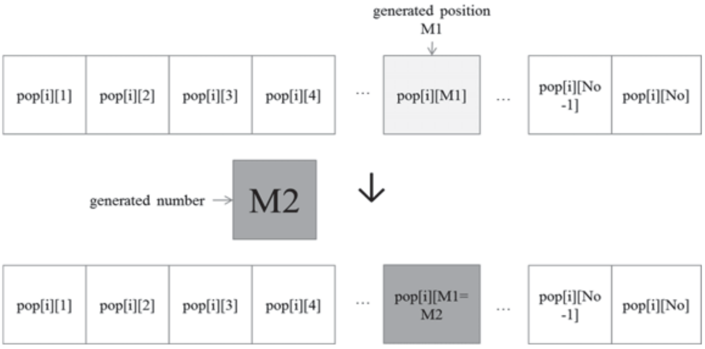


Fig. 4 Operation of mutation for a chromosome (M1 ≤ No, M2 ≤ N)

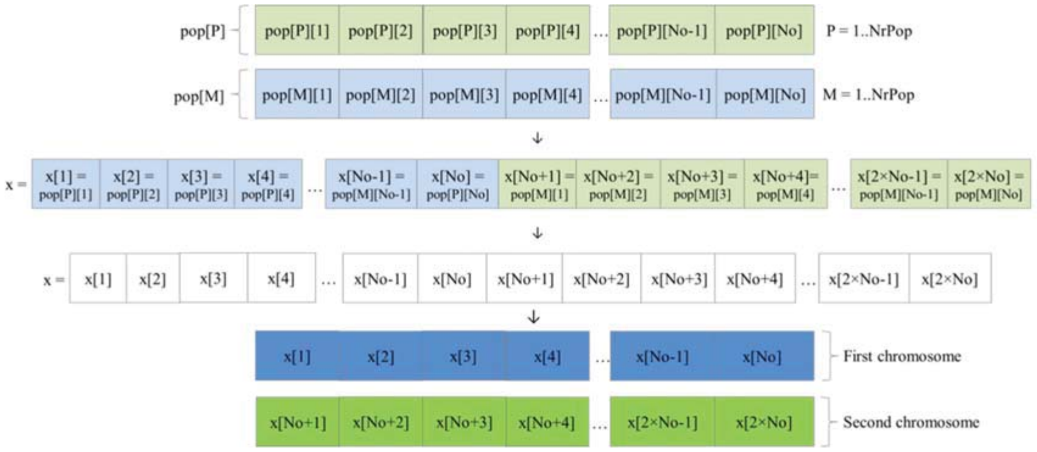


Fig. 5 Operation of crossover for a chromosome (M, P ≤ NrPop)

C. Input Data

The input data consist in the number of tests (N), the initial tree (given by the array T[N]) and the number of generations (no_generations). The general form of a tree is presented in Fig. 6.

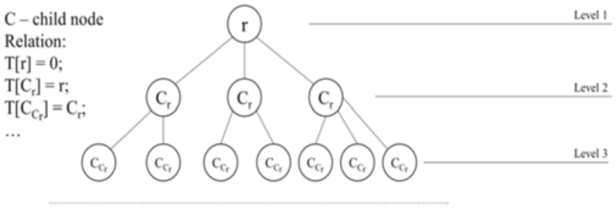


Fig. 6 General form of a tree with 3 levels

D. Output Data

As output data, the algorithm uses the first k rows of the bi-dimensional array pop[NrPop][No], because the sequences (rows) are ordered ascending by their fitness value. The output values are the distinct lines found in the population (after the genes and chromosomes are ordered ascendingly by the fitness value).

E. Conditions

The algorithm output the solutions which have the lowest fitness value. This is the main condition of the algorithm. Another condition is that the genes within a chromosome must be different.

Summing up, the conditions are:

- f(pop[i][j]) = minimal, i ≤ NrPop, j ≤ No
- pop[i][j1] ≠ pop[i][j2], i ≤ NrPop, j1,j2 ≤ No

F. Steps of the Genetic Algorithm

The algorithm has several steps.

Step 1. The input data is read. The input data consist in the number of tests (N), the initial tree (given by the array T[N]) and the number of generations (no_generations).

Step 2. The next population is generated: pop[i][j], i=1,NrPop, j=1,N+1, pop[i][N+1], storing the fitness value. In the implementation NrPop=1400 was used, with very good results.

Step 3. The crossover method is applied to the initial population. Thus, for two chromosomes M and P, we obtain the sequence x with 2×No genes which is ordered ascendingly. With the first No elements, we form the first chromosome and with the latter No elements we form the second chromosome. Both chromosomes are added to the population. The whole process is shown in Fig. 7.

pop[P] = (pop[P][1], pop[P][2], ..., pop[P][No])
pop[M] = (pop[M][1], pop[M][2], …, pop[M][No])
↓
x = (x[1], [2], ..., x[No], x[No+1], x[No+2], …, x[2No]), where
x[1] = pop[P][1], x[2] = pop[P][2], ..., x[No] = [P][No] and
x[No+1] = pop[M][1], x[No+2] = pop[M][2], …, x[2No] = pop[M][No]
↓
Ascending order of x and two new chromosomes are built
↓
(x[1], x[2], ..., x[No]) and
(x[No+1], x[No+2], …, x[2No])

Fig. 7 Step 3 structure where fitness values are calculated for each new sequence (chromosome)

Step 4. The mutation method is applied to the initial population. Thus, for a chromosome P, a position M1 and a gene M2:

pop[P] = (pop[P][1], pop[P][2], …, pop[P][M1], …, pop[P][No])
↓
pop[P] = (pop[P][1], pop[P][2], …, M2, …, pop[P][No])

Fitness values are calculated for each new sequence.

Step 5. Using any method of sorting, the values are ordered ascendingly by the fitness value of every chromosome.

Step 6. Steps 3, 4 and 5 are repeated for no_generations number of times.

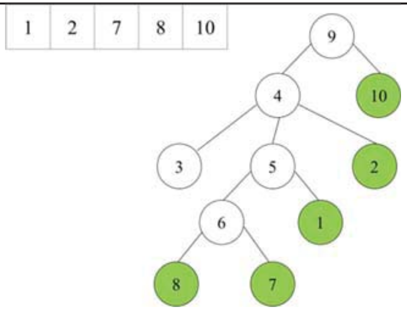
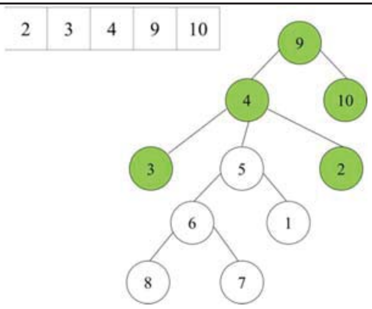
Step 7. The first k distinct rows of the bi-dimensional array pop (the sequences) are output, those being the correct values.

The fitness value is calculated using two arrays: v and w. Generally, for a gene pop[i][j], w can be defined in this way:

$$w[pop[i][j]] = \begin{cases} 1, & \text{at the position } pop[i][j] \text{ in the array } w \\ 0, & otherwise \end{cases} \quad (2)$$

The array v depends on the array w and the subtree that can be formed by nodes from the chromosome pop[i][j]. Thus, the array v contains the nodes which are part of the chromosome, but do not form a subtree with the other genes. The number of these nodes are actually the fitness value. Intuitively, this array helps at finding the closest subtree that can be formed from the genes of the chromosome pop[i][j], which can themselves form a subtree in the best case. Table I presents the scheme of the calculation of a fitness value for a subtree.

TABLE I
CALCULATION OF FITNESS VALUES FOR AN EXAMPLE OF A TREE TWO EXAMPLES OF CHROMOSOMES

Example 1	Example 2
	
w = (1, 1, 0, 0, 0, 0, 1, 1, 0, 1)	w = (0, 1, 1, 1, 0, 0, 0, 0, 1, 1)
v = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)	v = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
(w[1] = = 0 and v[1] = = 0) is false	(w[2] = = 0 and v[2] = = 0) is false
(w[5] = = 0 and v[5] = = 0) is true and the initialization v[5] = 1 is made	(w[4] = = 0 and v[4] = = 0) is false
(w[4] = = 0 and v[4] = = 0) is true and the initialization v[4] = 1 is made	(w[9] = = 0 and v[9] = = 0) is false
(w[9] = = 0 and v[9] = = 0) is true and the initialization v[9] = 1 is made	
w = (1, 1, 0, 0, 0, 0, 1, 1, 0, 1)	w = (0, 1, 1, 1, 0, 0, 0, 0, 1, 1)
v = (0, 0, 0, 1, 1, 0, 0, 0, 1, 0)	v = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
This step is repeated for each gene. After this method, v and w has the next values:	
w = (1, 1, 0, 0, 0, 0, 1, 1, 0, 1)	w = (0, 1, 1, 1, 0, 0, 0, 0, 1, 1)
v = (0, 0, 0, 1, 1, 1, 0, 0, 0, 1)	v = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
(w[10] = = 0 and v[10] = = 0) is false	(w[10] = = 0 and v[10] = = 0) is false
(w[9] = = 0 and v[9] = = 0) is false	(w[9] = = 0 and v[9] = = 0) is false
f(1,2,7,8,10) = ∑v[i] = 4, which means 4 nodes miss for a subtree (4, 5, 6 and 9, the indices where the elements of array *v* is 1).	f(2,3,4,9,10) = ∑v[i] = 0, which means the chromosome formes a subtree.

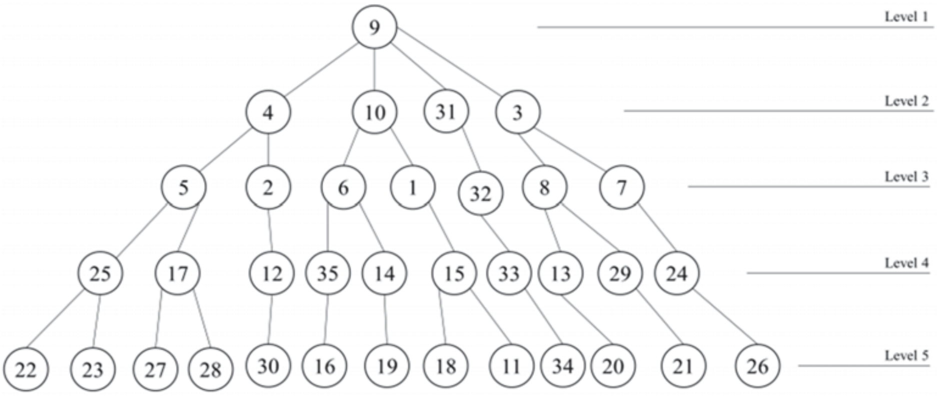


Fig. 8 The tree for our example

IV. RESULTS AND DISCUSSIONS

As an example, we will take a battery of tests related to informatics. The tests are codified with numbers from 1 to N. The input data for our example is: N=35, No=8, no_generations=400, k=15, T=(10, 4, 9, 9, 4, 10, 3, 3, 0, 9, 15, 2, 8, 6, 1, 35, 5, 15, 14, 13, 29, 25, 25, 7, 5, 24, 17, 17, 8, 12, 9, 31, 32, 33, 6). The tree is shown in Fig. 8.

The values output for the tree are presented in Table II. The number of distinct solution is 16 and the runtime for this example was 3.790 seconds. For the graph from Table I, the solution output is shown in Table III. The number of distinct solution is 16 and the runtime for this example was 3.327 seconds.

V. CONCLUSIONS

This method of generating subtrees with minimal number of missing edges is useful in case of arborescent structures defined and classified by a certain criterion. It is useful in situations when these types of structures are used. A future work would represent the development of a complex application with a friendly interface for the users. Using this application, tests can be loaded, the relations between them are set and sequences of tests can be generated. This algorithm is used in the assessment of students in various universities with very good results, alongside other types of applications.

TABLE II
RESULTS FOR THE GIVEN EXAMPLE

Sequence	Number of edges needed for forming a subtree
2 3 4 6 7 8 9 10 12 13	0
1 2 3 4 6 8 9 10 12 13	0
1 2 3 4 5 6 7 8 9 10	0
1 2 3 4 6 7 8 9 10 12	0
1 2 3 4 5 7 8 9 10 12	0
1 2 3 4 6 7 9 10 12 15	0
1 2 3 4 5 6 7 9 10 12	0
2 3 4 5 7 8 9 10 13 17	0
1 2 3 4 5 6 8 9 10 17	0
1 2 3 4 6 7 8 9 10 24	0
1 2 3 4 8 9 10 12 13 31	0
1 2 3 4 5 7 8 9 10 13	0
1 2 3 4 5 7 8 9 10 15	0
2 3 4 5 6 7 8 9 10 14	0
1 2 3 4 5 8 9 10 12 13	0
1 3 4 5 6 7 8 9 10 24	0
1 2 3 4 6 7 8 9 10 29	0

TABLE III
RESULTS FOR THE GIVEN EXAMPLE AT TABLE I AND FIG. 1

Sequence	Number of edges needed for forming a subtree
2 4 5 9 10	0
4 5 6 9 10	0
3 4 5 9 10	0
4 5 6 8 9	0
1 4 5 6 9	0
2 3 4 9 10	0
2 3 4 5 9	0
1 2 4 5 9	0
4 5 6 7 9	0
2 4 5 6 9	0
1 3 4 5 9	0
3 4 5 6 9	0
1 4 5 9 10	0
1 2 3 4 9	1
1 2 3 4 5	1
1 4 5 6 8	1
3 4 5 8 9	1

REFERENCES

[1] D. Boud, N. Falchikov, *Rethinking Assessment in Higher Education: Learning for the Longer Term*, Routledge Publishing, 2007.

[2] E. MacLellan, "Assessment for Learning: The differing perceptions of tutors and students", *Assessment & Evaluation in Higher Education* Volume 26, Issue 4, pp. 307-318, 2001.

[3] K. Struyven, F. Dochy, S. Janssens, "Students' perceptions about new modes of assessment in higher education: a review, optimising new modes of assessment: in search of qualities and standards", *Innovation and Change in Professional Education*, Volume 1, pp. 171-223, 2005.

[4] Ş. Valeriu, C. Ştefănescu, O. Roşu-Stoican, "The influence of using ICT on the quality of learning" *in International Conference on Virtual Learning – ICVL*, Timișoara, România, 2015, pp. 169-172.

[5] C. Roy, P. Wallace, "Paper–based versus computer–based assessment: key factors associated with the test mode effect", *British Journal of Educational Technology*, Volume 33, Issue 5, pp. 593–602, 2002.

[6] E. Popescu, "Adaptation provisioning with respect to learning styles in a web-based educational system: an experimental study", *Journal of Computer Assisted Learning*, Vol. 26(4), Wiley, pp. 243-257, 2010.

[7] C. Holotescu, "A conceptual model for Open Learning Environments" *in International Conference on Virtual Learning – ICVL*, Timișoara, România, 2015, pp. 54- 61.

[8] C. Baron, A. Şerb, N. M. Iacob, C. L. Defta, "IT infrastructure model used for implementing an e-learning platform based on distributed databases", *Quality-Access to Success Journal*, Vol. 15, Issue 140, pp. 195-201, 2014.

[9] C. L. Defta, A. Șerb, N. M. Iacob, C. Baron, "Threats analysis for E-learning platforms", *Knowledge Horizons. Economics*, Vol. 6 / Nr. 1, pp. 132–135, 2014.

[10] G. Martin, "Cognitive style and attitudes towards using online learning and assessment methods", *Electronic Journal of e-Learning*, Volume 1, Issue 1, pp. 21-28, 2003.

[11] J. Gaytan, C. McEwen-Beryl, "Effective online instructional and assessment strategies", *American Journal of Distance Education*, Volume 21, Issue 3, pp. 117-132, 2007.

[12] D. Nijloveanu, N. Bold, A.C. Bold, "A hierarchical model of test generation within a battery of tests" *in International Conference on Virtual Learning-ICVL*, Timișoara, România, 2015, pp. 147-153.

[13] D. E. Knuth., *The art of computer programming, volume 3: (2nd ed.) sorting and searching*, Addison Wesley Longman Publishing Co., Inc. Redwood City, CA, USA.

[14] Th. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization", *Machine Learning*, Volume 40, Issue 2, pp. 139-157, August 2000.

[15] S. Rahmani, S. M. Mousavi, M. J. Kamali, "Modeling of road-traffic noise with the use of genetic algorithm", *Applied Soft Computing*, Volume 11, Issue 1, pp. 1008–1013, 2011.

[16] L. G. Caldas, L. K. Norford, "A design optimization tool based on a genetic algorithm, Automation in Construction", *ACADIA '99*, Volume 11, Issue 2, pp. 173–184, 2002.

[17] H.-S. Kim, S.-B. Cho, "Application of interactive genetic algorithm to fashion design", *Engineering Applications of Artificial Intelligence*, Volume 13, Issue 6, Pages 635–644, December 2000.

[18] D. A. Popescu, D. Radulescu, "Approximately similarity measurement of web sites" *in ICONIP, Neural Information Processing, Proceedings LNCS*, Springer, 9-12 November 2015.

[19] D. A. Popescu, I. A. Popescu, "Model of determination of coverings with web pages for a website" *in International Conference on Virtual Learning-ICVL*, Timișoara, România, 2015, pp. 279-283.

[20] S. Darby, Th. V. Mortimer-Jones, R. L. Johnston and Ch. Roberts, "Theoretical study of Cu–Au nanoalloy clusters using a genetic algorithm", *J. Chem. Phys. 116, 1536, The Journal of Chemical Physics*, Volume 116, Issue 4, 2002.

[21] D. A. Popescu, D. Radulescu, "Monitoring of irrigation systems using genetic algorithms" *in ICMSAO*, IEEE Xplorer, 2015, pp. 1-4.

[22] D. A. Popescu, D. Nicolae, "Generating a class schedule with reduced number of constraints" *in the 9th International Scientific Conference eLearning and software for Education*, Bucharest, April 25-26, 2013, ISI Proceedings, pp. 297-300.

[23] D. A. Popescu, "Probabilistic program that generates Langford sequences", *Scientific Bulletin – University of Pitești, Mathematics and Computer Sciences Series,* (*Buletin Stiințific - Universitatea din Pitești, Seria Matematică și Informatică*), Nr. 12, pp. 129-133, 2006.