JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 34, 1405-1423 (2018)
DOI: 10.6688/JISE.201811_34(6).0004

Application of Particle Swarm Optimization to Create Multiple-Choice Tests

TOAN BUI[1], TRAM NGUYEN[2,3], BAY VO[4,5,+], THANH NGUYEN[1],
WITOLD PEDRYCZ[6,7,8] AND VACLAV SNASEL[3]
[1]*Faculty of Information Technology, Ho Chi Minh City University of Technology*
Ho Chi Minh City, 70000 Vietnam
[2]*Faculty of Information Technology, Nong Lam University*
Ho Chi Minh City, 70000 Vietnam
[3]*Department of Computer Science, Faculty of Electrical Engineering and Computer Science*
VŠB – Technical University of Ostrava
Ostrava-Poruba, 708 33 Czech Republic
[4]*Division of Data Science, Ton Duc Thang University*
Ho Chi Minh City, 70000 Vietnam
[5]*Faculty of Information Technology, Ton Duc Thang University*
Ho Chi Minh City, 70000 Vietnam
[6]*Department of Electrical and Computer Engineering*
University of Alberta
Edmonton, T6R 2V4 AB Canada
[7]*Department of Electrical and Computer Engineering*
Faculty of Engineering
King Abdulaziz University
Jeddah, 21589 Saudi Arabia
[8]*Systems Research Institute*
Polish Academy of Sciences
Warsaw, 01-447 Poland

Generating tests from question banks by using manually extracted items or involving random method consumes a great deal of time and effort. At the same time, the quality of the resulting tests is often not high. The generated tests may not entirely meet the requirements formulated in advance. Therefore, this study develops innovative ways to enhance this process by optimizing the execution time and generating results that closely meet the extraction requirements. The paper proposes the use of Particle Swarm Optimization (PSO) to generate multiple-choice tests based on assumed objective levels of difficulty. The experimental results reveal that PSO speed-ups the extraction process, and improves the quality of tests in comparison with the results produced by previously used methods such as Random or Genetic Algorithm (GA) optimized methods. In addition, PSO shows to be more efficient than GA and random selection in most criteria, such as execution time, search space, stability, and standard deviation.

Keywords: test question bank, multiple-choice tests, genetic algorithms, particle swarm optimization, creating tests

1. INTRODUCTION

To date, the process of generating tests from question banks is usually done manually or with the help of extraction programs. With the former approach, users subjectively select the questions according to their preferences; however, this requires expert know-

Received April 26, 2017; revised September 15, 2017 & January 2, 2018; accepted January 21, 2018.
Communicated by Meng Chang Chen.
[+] Corresponding author

ledge and sufficient teaching skills regarding the subject. For the latter, the majority of extraction programs are based on random selection methods, and thus the resulting tests are not based on a set of pre-defined requirements. Although both methods come with their advantages, they still exhibit some drawbacks. First, using a question bank consisting of a large number of items or generating a test that requires many questions is computationally expensive, which leads to poor performance. Second, the questions selected randomly may not be evenly spread across the question bank, leading to a lack of diversity. Thirdly, if the question bank consists of multiple-choice items based on objective difficulty levels, then manually choosing items to meet any specification of the given criteria will take a long time, while the tests produced by a random generation process may also not always meet the related criteria. A question bank includes various items, each of which exhibits a difficulty level, which can be assessed by either objective or subjective methods. A subjective difficulty level is determined by the user, but this has a low level of reliability depending on subjective feelings and preferences. In contrast, an objective difficulty level is quantified based on feedback collected from test-takers. Because the items are thus evaluated objectively, an item can be computed by

$$DK_i = \frac{the\ number\ of\ correct\ answers_i}{Total\ number\ of\ answers_i}. \quad (1)$$

with, this approach is highly reliable. In this study, we propose an approach that combines an adaptive fitness function and objective difficulty level to generate tests from a question bank that can satisfy a given set of requirements.

This paper proposes the use of the Particle Swarm Optimization (PSO) to generate multiple-choice tests based on assumed objective level of difficulty. The objective level of difficulty assumes values in (0,1], so if we use the normal probability functions, that hardly can extract the tests that have levels of difficulty meeting the specific difficulty level requirement. For example, when using random method to extract the tests in large questions bank, it is often difficult to identify every element of a question bank so it was difficult to meet the specific requirement. If it has been found to meet the requirement level, its determination consumes much time and effort. At the same time, the quality of the resulting tests is often not high. The tests may not entirely satisfy the requirements formulated in advance. To rectify the problem, we consider Particle Swarm Optimization for addressing multiple objectives encountered in continuous space problems.

Both the GA and PSO have been widely applied to solve optimization problems such as generating tests from a question bank. In 2007, simulated annealing (SA) and an adaptive simulated annealing genetic algorithm (ASAGA) were also used to achieve this objective [1]. The results of experiments and a comparative analysis showed that the GA used in combination with the proposed mutation operator is successful as nearly 100% and it produces results in noteworthy computational times. In 2010, Yildirim proposed a method using a heuristic optimization approach based on a GA [2]. The results of experiments showed that the use of GA with the proposed mutation operator was able to achieve an approximately 100% success rate in meeting the related criteria. Next, some other PSO- and GA-based algorithms have been developed and applied to benchmark and real-world optimization problems such as Particle Swarm Optimization (PSO): Hybridization perspectives and experimental illustrations [12], DE-PSO: A New Hybrid

Meta-Heuristic For Solving Global Optimization Problems [15], Improved Accelerated PSO Algorithm for Mechanical Engineering Optimization Problems [13], Directionally Driven Self-Regulating Particle Swarm Optimization algorithm (DDSR-PSO) [16], A new PSO-optimized geometry of spatial and spatiotemporal scan statistics for disease outbreak detection has been proposed in [14]. In 2016, Zou *et al.* proposed a new two-level hierarchical multi-swarm cooperative TLBO (teaching – learning-based optimization) variant [17] called HMCTLBO to solve the global optimization problem. The learners of each swarm evolve independently only in their corresponding swarm in parallel to maintaining the diversity and improving the exploration capabilities of the population.

Wang *et al.* surveyed the use of PSO algorithms from 1995 to 2016 [18]. The authors introduce its origin and background and carried out theoretical analysis. They discussed ongoing research and application in algorithm structure, parameter selection, topology structure, discrete PSO algorithm and parallel PSO algorithm, multi-objective optimization PSO.

The purpose of this study is to provide with an efficient vehicle to generate tests with multiple-choice questions coming from a question bank. In this study, PSO is used to optimize predefined criteria for selecting questions. PSO is a useful optimization algorithm because of its versatility. We address the issues of early convergence, extraction time and development of tests that meet the related requirements. We propose a suitable target function (fitness) and apply it to PSO approach to extract the items coming from a question bank. The experimental results show that the PSO approach is suitable for the selection of near-optimal questions from large-scale questions banks.

The paper is organized as follows. Section 1 introduces this study, while Section 2 formulates the problem of generating multiple-choice tests. Some related works are discussed in Section 3, while the proposed method is described in Section 4. Section 5 analyzes the experimental results of this study, followed by related discussions in Section 6. Finally, Section 7 provides some conclusions and suggests some potential directions for future research.

2. THE PROBLEM OF GENERATING MULTIPLE-CHOICE TESTS

The problem of generating multiple-choice tests, as the name implies, is to generate multiple-choice tests that satisfy the given levels of difficulty based on provided question banks and difficulty of questions. Each item in a question bank comes with an objective difficulty level, which has the half-opening interval (0,1]. For example, the objective difficulty level of an item can be computed by (1). The issue of extracting test questions that will meet the specific difficulty level requirement (DL_R) given by the user is thus a key one in this context. Let $Q = \{Q_1, Q_2, Q_3, Q_4, Q_5, ..., Q_n\}$ be a test question bank of n questions, each of which has the attributes C_i, including code question (CQ), code part (CP), and objective difficulty level (DL). Our objective is to generate a test including m questions ($m \leq n$) $Q_i = \{Q_{i1}, Q_{i2}, Q_{i3}, Q_{i4}, Q_{i5}, \ldots, Q_{im}\}$ ($Q_{ij} \in Q$) which satisfy DL_R, *i.e.*, $DL_R = \frac{\sum_{j=1}^{m} Q_{ij}.DL}{m}$. This is considered as the objective function used in the problem.

The constraints present in the problem are described as follows:

$C1$: One ensures that the questions do not overlap each other: C_{ij} each of the questions included in the test is thus unique; *i.e.* $\{\forall C_{ij} \in Q_i, \nexists C_{ij'} \in Q_i : C_{ij}.CP = C_{ij'}.CP\}$.
$C2$: This constraint ensures that each question's difficulty level, $Q_{ij}.DL$ does not converge on the required difficulty value (DL_R): $\{\forall C_{ij} \in Q_i, \nexists C_{ij'} \in Q_i : C_{ij}.DL = C_{ij'}.DL = DL_R\}$.
$C3$: This constraint ensures that all the questions in one group appear together. This means that if we pick up a single question out of a specific group then we must also accept the rest of them. Suppose there is a group of questions $Q_k = \{Q_{k1}, Q_{k2}, Q_{k3}, Q_{k4}\}$, if $Q_{kj} \in Q_i$, then it is obligatory that $Q_k \in Q_i$.
$C4$: This constraint ensures that the right number of questions is in each section (CP). Assume the question bank includes p sections, such as $P = \{CP_1, CP_2, CP_3, CP_4, \ldots, CP_p\}$, then it is necessary to extract q sections $P_i = \{CP_{i1}, CP_{i2}, CP_{i3}, CP_{i4}, \ldots, CP_{iq}\}$ ($P_i \subseteq P$), where each section CP_{ij} has the required number of questions denoted by NQ_R which satisfies the condition: $\sum_{j=1}^{q} CP_{ij}.NQ_R = m$.

3. RELATED WORK

Outcome-based education (OBE) is an educational theory that bases each part of an educational system around goals (outcomes) [19]. The previous works were based on the Bloom's taxonomy for building test question banks to evaluate individual student as presented in [3, 4]. Besides, those works investigated multi-constraints genetic algorithm approaches in designing Auto-Generator of Examination Questions (AGEQ). The analysis showed impressive results of the difference between the desired and actual AGEQ output. Here, the difficulty level of each question deposited in the question paper is determined based on the keywords found in the question; the six levels of difficulty are labeled as follows: 1-*remember*, 2-*understand*, 3-*apply*, 4-*analyze*, 5-*evaluate*, 6-*create*. A list of cognitive processes is organized from the simplest one, the recall of knowledge, to the most complex, making judgments about the value and assessing the worth of an idea. To realize the ideas, a Question Bank is needed which will be used as a reference for classifying the level of difficulty of the individual question. We see that Bloom's taxonomy based methods for building test question banks only extract the test on discursive space search problems.

GA and PSO have been applied to optimize many real-world problems. In this section, we briefly review some of the representative studies in which GA and PSO were used to produce tests. In 2010, Yildirim presented an efficient way of generating multiple-choice tests from a question bank [2]. GA was used to optimize the question-selection process with a set of predefined criteria. In this earlier study, a mutation operation was proposed to prevent the duplication of crossover individuals. The results of both experiments and analysis showed that the use of GA with the proposed mutation operator led to an almost 100% success rate in meeting the related criteria. More recently, GA was developed and applied to an approach known as the Auto-Generator of Examination Questions (AGEQ) [3], and the analysis showed impressive results when assessing differences between the desired and actual AGEQ output. Teo *et al.* aimed to optimize the selection of final examination questions based on the cognitive levels present in the Bloom's taxonomy and designed a prototype GA-based auto-generator for examination questions [4]. This approach can be also developed for use with other types of question.

Paul *et al.* modeled the Question Selection Problem as a multi-constraint optimization problem, and proposed an evolutionary approach for its implementation [5]. The results of a case study were presented, and those showed that this is a promising research direction for dealing with such problems.

Kennedy and Eberhart proposed the concept of particle swarm optimization (PSO). The PSO algorithm simulates the behavior of birds flocking, fish schooling, and social interaction within a population, in general. Each single solution is a "bird" in the search space and is considered as a "particle". All particles come with their own fitness values, which are evaluated by the fitness function to be optimized. The particles also exhibit random velocities that indicate in which direction they are moving. This velocity is adjusted according to the particle's own experience and that of its neighbors. Each particle is updated by the two best values in each iteration and keeps track of its own coordinates in the search space which is associated with the best solution (fitness) that has been achieved (the fitness value is thus also stored by each particle). This value is called pbest. Another "best" value that is tracked by the PSO is the best value obtained so far by any particle in the swarm, and since this best value is global, it is called G_{best} [7]. In 2009, Cheng, Lin, and Huang proposed a dynamic question generation system for web-based testing using PSO [8], experimental results show that PSO satisfies most criteria such as execution time and fitness value. However, the quality of results is not fully acceptable in terms of their stability, and standard deviation.

In summary, the existing research on the usage of PSO to produce tests has focused on two main approaches: (1) improving the algorithm from the original model [7, 12-14, 16], and (2) studying the application of the resulting algorithms to real-world problems [8-11, 15].

Up to now, there have not been any research using PSO to extract tests for continuous search space. Therefore, our work is the first one to propose PSO for multiple objectives in continuous space problems.

4. PROPOSED METHOD

This section is divided into three subsections, where we formulate a suitable objective function, develop GA to extract tests and propose the PSO-based approach.

4.1 Objective Function

As stated in Section 2, the objective function of the problem can be formulated as follows:

$$f(Q_{ij}.DL)\frac{\sum_{j=1}^{m} Q_{ij}.DL}{m} - DL_R \rightarrow \min$$

so that $f(Q_{ij}.DL)$ satisfies the conditions $\{C1, C2, C3, C4\}$. Here m is the total number of questions in the test, $Q_{ij}.DL$ is the difficulty level of each question, and DL_R is the requirement of difficulty level.

The objective function $f(Q_{ij}.DL)$ is used as the fitness function in the two algorithms, and the results of the objective function are considered as the fitness of the resulting test.

In this case, the better the fitness, the smaller the $f(Q_{ij}.DL)$ becomes. To improve the quality of the test, we also take into account the constraints $C1$, $C2$, $C3$, and $C4$, as mentioned in Section 2.

4.2 GA Approach to Test Extraction

When using GA [2] to extract a test from the question bank, the operators of the proposed algorithm are as follows:

Step 1: Initialize the population.
LOOP
Step 2: IF the population satisfies stop conditions THEN exit LOOP.
Step 3: Select the individuals using Roulette Wheel Selection and sort them based on the fitness function.
Step 4: Crossover the two parents to form new offspring, but also retain both parents.
Step 5: Mutate individuals to create new individuals.
END LOOP

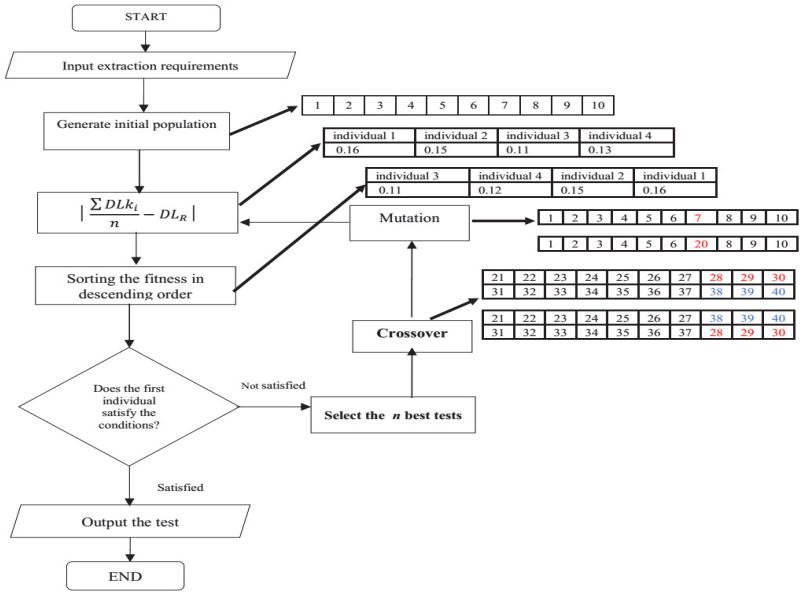

Fig. 1. The flowchart of GA used to extract a test.

In encoding instances of the problem solution, we use a series of question in the exam. The Q_i exam includes m questions: $Q_i = \{Q_{i1}.S = 1, Q_{i2}.S = 2, Q_{i3}.S = 3, Q_{i4}.S = 4, \ldots, Q_{im}.S = m\}$ (with S is the order of questions).

Selection operator is applied after the initial population is created, and the operation is repeated for each generation. The individuals are sorted by their fitness in a decreasing order. If the number of individuals in the population of the current generation is greater

than the initial number of individuals, then we remove those with the lowest fitness. If the first individual (the best) satisfies the requirements, the algorithm will stop, and that individual is treated as the result of the algorithm.

Crossover operator will crossover two individual parents to generate two offspring by using a paragraph crossover approach that retains both parents. Two individuals are selected to be parents based on their fitness ranking, and the best parents are used to produce hybrids that then produce the next pairs.

The mutation operator mutates one segment of questions which are chosen randomly from the question bank.

Stop condition: the algorithm will stop when the fitness values are smaller than ε, with ε being defined by the user.

4.3 Proposed Method: PSO Approach for Extracting Tests

When using PSO to extract a test, the operators used in the proposed algorithm are as follows:

Step 1: Initialize the population
LOOP
Step 2: Select G_{best} and P_{best};
Step 3. IF G_{best} satisfies the conditions stop THEN exit LOOP.
Step 4: Update the locations of individuals.
4.1. The P_{best} individuals approach G_{best}, with V_{Pbest}
4.2. G_{best} approach goal, with V_{Gbest}
END LOOP

Where $f(Q_{ij}.DL)\frac{\sum_{j=1}^{m} Q_{ij}.DL}{m} - DL_R$ (in section 4.1) is a fitness function, P_{best} is the locally optimal solution, with $P_{best} = \min(f1, f2, \ldots, fi)$ (where fi is the number of particles), G_{best} is the globally optimal solution, with $G_{best} = \min(P_{best1}, P_{best2}, \ldots, P_{besti})$, V_{Pbest} is the velocity of P_{best}, with $V_{Pbest} = \alpha *$ the total number of questions in the test and V_{Gbest} is the velocity of G_{best}, with $V_{Gbest} = \beta *$ the total number of questions in the test.

The solution to the problem is encoded in the PSO [15].

- The Selection operator selects the global optimal G_{best} and local optimal P_{best} based on fitness. The best individual in a population is called G_{best}, and the others are called P_{best}.
- The P_{best} individuals move to G_{best} by receiving information about the location of G_{best}, information about the location of G_{best} in the problem of extracting is the question. P_{best} moves to G_{best} with a velocity vector, which is the number of questions that transfer from G_{best} to P_{best}. After that, P_{best} receives random questions from G_{best}, and the new location of P_{best} is close to that of G_{best}. The fitness value is then checked, and if the fitness of the new location is better than that of the old location then the system updates P_{best} based on the new location, otherwise, the old location is unchanged.
- Moving G_{best} to the goal is not the same process as moving P_{best} to G_{best}. In this process, G_{best} replaces its question with other items from the question bank. The velocity vector is the number of questions as in the process when P_{best} moves to G_{best}. After receiving

questions and moving to a new position, if the fitness of G_{best} at the new location is better than that of the old location, then the old location of G_{best} is updated by the new location, otherwise, the old location is unchanged.

- Stopping condition: the algorithm will stop when the fitness values are smaller than ε, with this being defined by the user.

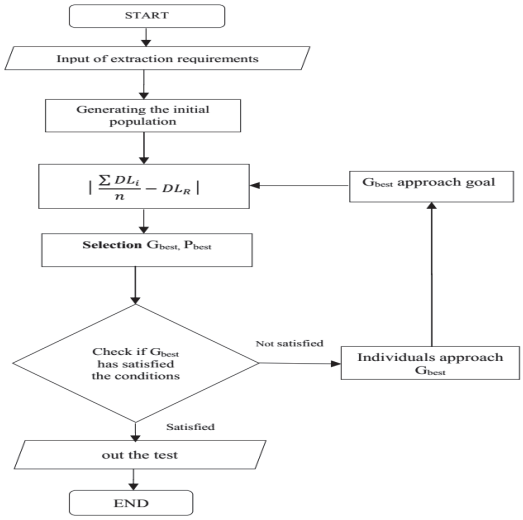

Fig. 2. Flowchart of the PSO algorithm used for extracting the test.

Example: Given is a test question bank

CQ	01	02	03	04	05	06	07	08	09	10
DL	0.3	0.2	0.8	0.7	0.4	0.6	0.5	0.8	0.2	0.3

The test extraction requirement is for four questions with a difficulty level of 0.3. The fitness smaller than $\varepsilon = 0.03$, $\alpha = 0.25$, $\beta = 0.25$ is taken as the stopping criterion.

– *Generation* 1, initialize a population that includes two individuals:

Individual 1	CQ	05	08	01	04	Fitness
	DL	0.4	0.8	0.3	0.7	0.25
Individual 2	CQ	02	06	01	03	Fitness
	DL	0.2	0.6	0.3	0.8	0.175

– Selection of G_{best} and P_{best}:

Individual 2 G_{best}	CQ	02	06	01	03	Fitness
	DL	0.2	0.6	0.3	0.8	0.175
Individual 1 P_{best}	CQ	05	08	01	04	Fitness
	DL	0.4	0.8	0.3	0.7	0.25

Individual G_{best} has the fitness of 0.175, which does not satisfy the stopping condition.

+ G_{best} approaches the goal:

Individual	CQ	02	04	01	03	Fitness
4	DL	0.2	0.7	0.3	0.8	0.2

+ P_{best} approaches G_{best} by receiving a random question 01 from G_{best}:

Individual	CQ	05	06	01	04	Fitness
3	DL	0.4	0.6	0.3	0.7	0.2

– *Generation* 2, selection of G_{best} and P_{best}:

Individual 2	CQ	02	06	01	03	Fitness
G_{best}	DL	0.2	0.6	0.3	0.8	0.175
Individual 3	CQ	05	06	01	04	Fitness
P_{best}	DL	0.4	0.6	0.3	0.7	0.2

+ G_{best} approaches the goal:

Individual	CQ	02	06	01	10	Fitness
5	DL	0.2	0.6	0.3	0.3	0.05

+ P_{best} approaches G_{best} by receiving a random question 02 from G_{best}:

Individual	CQ	02	06	01	04	Fitness
6	DL	0.2	0.6	0.3	0.7	0.15

– *Generation* 3, selection of G_{best} and P_{best}:

Individual 5	CQ	02	06	01	10	Fitness
G_{best}	DL	0.2	0.6	0.3	0.3	0.05
Individual 6	CQ	02	06	01	04	Fitness
P_{best}	DL	0.2	0.6	0.3	0.7	0.15

+ G_{best} approaches the goal:

Individual	CQ	02	05	01	10	Fitness
7	DL	0.2	0.4	0.3	0.3	0

+ P_{best} approaches G_{best} by receiving a random question 02 from G_{best}:

Individual	CQ	02	06	01	10	Fitness
8	DL	0.2	0.6	0.3	0.3	0.05

– *Generation* 3, selection G_{best} and P_{best}:

Individual 7	CQ	02	05	01	10	Fitness
G_{best}	DL	0.2	0.4	0.3	0.3	0
Individual 8	CQ	02	06	01	10	Fitness
P_{best}	DL	0.2	0.6	0.3	0.3	0.05

The fitness value of G_{best} is 0, and thus satisfies the stopping condition. As such, the test extraction requirement is met by Individual 7.

5. EXPERIMENTAL STUDIES

In this section, we first provide the setup arguments and parameters for the experiments in subsection 5.1 and the description of the database in subsection 5.2. After which, we quantify the method's execution time, search space, stability, and standard deviation in the subsections 5.3, 5.3, 5.5 and 5.6, respectively.

Algorithms are implemented in *C#* (Microsoft Visual Studio 2013), using Windows 8.1, and running on a computer with a 2.5 GHz CPU and 4 GB RAM.

5.1 Parameters Used in the Experiments

Table 1. Parameters used in GA and PSO.

GA		PSO Based on the best parameters used in GA, the parameters in PSO are selected	
Parameters	GA	Parameters	PSO
Population size	10 – 60 individuals	Population size	10 – 60 individuals.
Individual size	10, 100 questions.	Individual size	10, 100 questions.
Rate of crossover	$\alpha = 0.1$	Velocity factor P_{best}:	$\alpha = 0.1$.
Rate of mutation:	$\beta = 0.05$	Velocity factor G_{best}:	$\beta = 0.05$
Stop condition	$\varepsilon = 0.0001$ (ε is tolerance) or the number of generations: 2000	Stop condition	$\varepsilon = 0.0001$ (ε is tolerance) or the number of generations: 2000

5.2 Experimental Database

The experimental database is the question bank (shown in Table 2), from which a random number of questions are taken, and the difficulty level of each question is based on the section of the course that is being tested.

The experimental database has multiple parts, each part consists of many questions, with each question having a difficulty level, as indicated in the Table 2. Based on database in Table 2, the test extraction requirements are presented in Table 3, and used to evaluate the operating boundary, and then compare the execution times of the algorithms.

Table 2. Experimental database 1.

Level of difficulty	#Questions Part 1	#Questions Part 2	#Questions Part 3	#Questions Part 4	#Questions Part 5	#Questions Part 6	Total questions
0.2	30	20	28	23	33	21	155
0.3	22	20	26	19	28	19	134
0.4	24	20	19	22	26	19	130
0.5	23	22	20	21	25	24	135
0.6	23	22	22	17	28	24	136
0.7	23	29	24	24	36	33	169
0.8	27	19	16	29	26	24	141
Total	172	152	155	155	202	164	1000

Table 3. Requirements for each part with regard to the number of questions.

Chapter/part	#Questions	The exam with 10 questions	The exam with 100 questions
part 1	172	2	20
part 2	152	1	10
part 3	155	2	20
part 4	155	0	0
part 5	202	3	30
part 6	164	2	20

Table 4. Experimental database 2.

Level of difficulty	#Questions Part 1	#Questions Part 2	#Questions Part 3	#Questions Part 4	#Questions Part 5	#Questions Part 6	Total questions
0.2	8	9	10	13	8	11	59
0.3	1	0	0	0	0	0	1
0.7	10	4	10	3	5	8	40
0.8	13	7	11	12	9	8	60
Total	32	20	31	28	22	27	160

Table 5. Experimental database 3.

Level of difficulty	#Questions Part 1	#Questions Part 2	#Questions Part 3	#Questions Part 4	#Questions Part 5	#Questions Part 6	Total questions
0.2	4	3	3	5	3	7	25
0.3	2	5	5	1	7	2	22
0.4	5	5	5	2	4	8	29
0.5	1	5	5	3	4	4	22
0.6	6	1	0	2	4	3	16
0.7	1	7	3	2	1	2	16
0.8	2	2	4	4	5	3	20
Total	21	28	25	19	28	29	150

The evaluation of the search speed of two algorithms finding solutions in the various difficulty level search spaces (using database the Table 4).

We evaluate the two algorithms with regard to their stability and various levels of difficulty requirements (using database in Table 5).

5.3 Evaluation of the Operating Boundary

Experiments were carried out to evaluate whether individuals can have the ability to find the goal in the search space or not. Finding results were identified by the number of the generation of each algorithm in Figs. 3-5.

The database shown in Table 2 is used in these experiments. To extract a test having 10 questions, the number of questions in each part is shown in Table 3. The required difficulty level is 0.4. The number of individuals in the algorithms is set to be equal to 4.

In Fig. 4, GA exhibits some early convergence with a fitness value of 0.05 at 3rd generation and thus must be forced to retain the best individuals and replace the rest in the 50th generation.

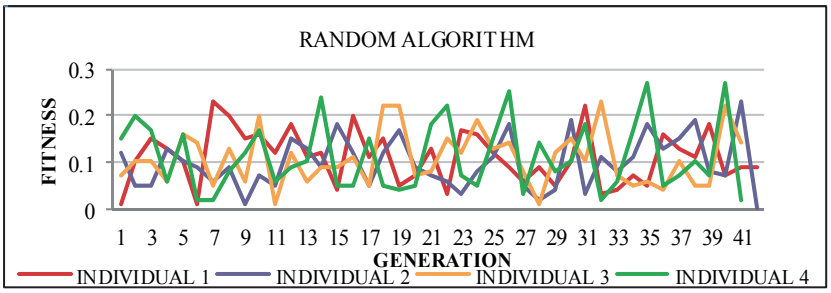

Fig. 3. Random operating boundary.

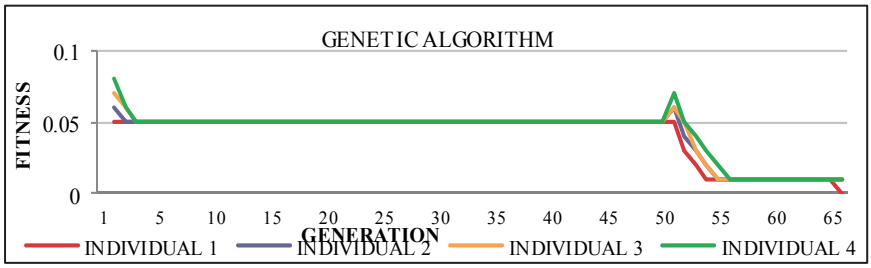

Fig. 4. GA operating boundary.

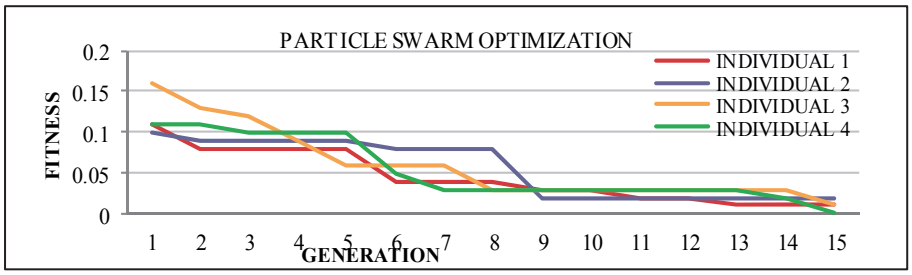

Fig. 5. PSO operating boundary.

The results show that individuals obtained using the random method find it difficult to solve the problem (Fig. 3). In contrast, GA and PSO are likely to find solutions to the problem. GA encounters the early convergence phenomenon and falls into a locally optimal solution with a fitness of 0.05 at the start of the 3rd generation (Fig. 4), and thus must be forced to retain the best individuals and replace the rest in the 50th generation. In contrast, PSO rarely encounters this issue (Fig. 5).

5.4 Runtime

Experiments were carried out to compare the execution times among the algorithms: Random, GA, and PSO.

The required difficulty level and number of individuals were both fixed, but the number of questions was changed. The experiments thus evaluate the performance of the

algorithms with a small and large number of questions. We use the database in Table 2 and extract tests with 10 questions and 100 questions, with the number of questions in each part shown in Table 3. The difficulty requirement is 0.4, the number of individuals used in the algorithms is 4, the time limit is 100 seconds, and the experiment is run 50 times. The results of the experiment are obtained as average values.

The results comparing the execution times of the algorithms with a small number of questions (*i.e.*, a test with 10 questions) are presented in Table 6.

Table 6. Comparing the execution times of the algorithms with a 10-question test.

Algorithms	#Questions	#Instances	Average time (seconds)	Quality of Result
Random	10	4	1.61	100%
GA			0.17	100%
PSO			0.16	100%

Table 7. Comparing the execution times of the algorithms with a 100-questions test.

Algorithms	#Questions	#Instances	Average time (seconds)	Quality of Result
Random	100	4	97.65	4%
GA			5.361	100%
PSO			2.123	100%

The results comparing the execution times of the algorithms with a large number of questions (100 questions) are presented in Table 7.

From the experimental results, we can see that with a test that includes 10 questions and has a difficulty level of 0.4, the random method can handle the extraction but the time needed is longer than that required by the other two algorithms. Moreover, PSO is faster than GA.

However, with the 100-question test and a difficulty level of 0.4, the random method cannot handle the extraction, while PSO is once again faster than GA. When we increase the size of the population, the runtime of PSO is longer, but GA can fall the state of local search.

In the case of a fixed number of questions, it is possible to change the required difficulty and the number of individuals used so that the performance of algorithms can improve, (using the database in Table 2). The aim it to extract a test with 100 questions and the number of parts are shown in Table 3, with the required difficulty being changed. With regard to the number of individuals, GA uses 40 while PSO uses 10. These are the number at which each algorithm performs best. To carry out the experiments, with the number of individuals by 10, 20, 40, 60 and the result of experiments displayed in Table 8 shows that our speed depends on the number of individuals using the algorithm.

Table 8. The speed of GA and PSO.

#Questions	#Individuals	GA (s)	PSO (s)
100	4	4.62	2.82
100	10	3.07	1.47
100	20	2.79	1.93
100	40	1.7	2.43
100	60	2.49	3.1

The results show that both algorithms successfully extracted the test, with the execution times shown in Table 9.

Table 9. The average executions times of the algorithms when using different levels of required difficulty.

Level of difficulty requirement	Average time (s)	
	GA	PSO
0.3	192.58	7.91
0.4	1.7	1.42
0.5	1.54	0.52
0.6	1.66	1.32
0.7	254.9	7.24

The results presented in Table 9 show that both algorithms can solve the problem of extracting an appropriate exam. However, when the difficulty level near the boundary is set as 0.3 and 0.7, GA is not as good as PSO with regard to finding the direction in the search space. In addition, PSO is faster than GA for all the different levels of difficulty.

5.5 Evaluation of the Standard Deviations

In this section, we present the evaluation of both algorithms with regard to their stability with various different difficulty requirements.

The experiments used the database shown in Table 5. The difficulty requirements are changed and the aim is to extract a test with 100 questions while considering a random number of questions for each part. The experiments were run for 2,000 generations, with a total of 50 experiments.

Table 10. Comparison of the standard deviations.

Level of difficulty	Standard deviations	
	GA	PSO
0.3	0.100513	0.075440
0.4	0.004460	0
0.5	0	0
0.6	0.057978	0.036421
0.7	0.159298	0.136646
Average	0.064449	0.049701

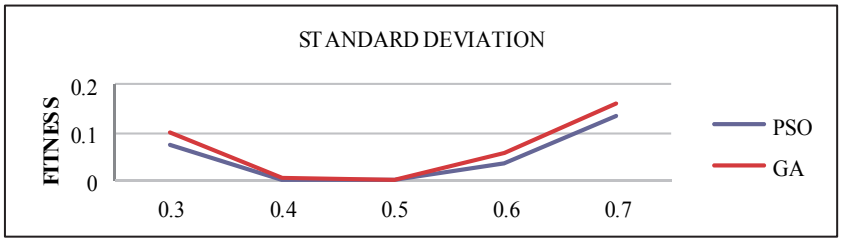

Fig. 6. Evaluation of standard deviations.

The results in Table 10 and Fig. 6 show that the standard deviations of PSO are always lower than those of GA. This means that the tests extracted by PSO have difficulty levels, which are very close to the initial requirements.

5.6 Evaluation of the Search Speed

Experiments were also carried out to evaluate the abilities of the two algorithms to find the solutions in the search spaces for different levels of difficulty. The experiments were performed with a question bank (as shown in Table 4). The best test selected has the fitness of 0.001. Both algorithms used 40 individuals and performed the extraction in the 2,000th generation.

The results in Table 11 show that the PSO is better than the GA with regard to finding the best test in the question bank. More specifically, three times out of 10 PSO found the best exam.

Table 11. Comparison of the search capabilities of the GA and PSO algorithms.

#Questions	#Individuals	PSO	GA
		The best fitness	The best fitness
100	40	0.003	0.05
		0.004	0.037
		0.002	0.038
		0.001 (the 1,749th generation)	0.029
		0.002	0.029
		0.002	0.038
		0.001 (the 1,680th generation)	0.039
		0.004	0.039
		0.003	0.054
		0.001 (the 1,733th generation)	0.038

6. DISCUSSION

The extracting test results used algorithms as random, GA that achieved in [2-4], however, the first, it only extracted the test with the objective levels of difficulty has discursive value domain; the second, in a more complex situation that requires a larger number of questions in each test, or involves unknown difficulty levels, a number of problems are raised that highlight the advantages and disadvantages of each algorithm. This study compared GA and PSO with the same constraints, and the results showed that GA obtains a locally optimal solution and converges quite early (Fig. 3). In contrast, PSO operates well because it is capable of moving towards the best solution to the problem (Fig. 4). By comparing the results recorded for the operating boundary (Figs. 3 and 4), speed (Tables 6-8), standard deviation (Table 10) (Fig. 5) and search capabilities (Table 11), we can see that the results of PSO are better than those of GA.

We can see that our proposed method achieves a good result for extracting tests by searching in continuous space with various constraints that must be complied. Hence, it can be applied to build education applications such as online-quiz system with affordable infrastructure.

7. CONCLUSIONS AND FUTURE STUDIES

The problem of extracting a test from the question bank with a given difficulty level is very common in today's educational environment. The PSO is applied as an optimization vehicle. The results showed that the method was able to find the best test to fit the requirements from a given question bank with a high level of accuracy and an improved execution time, as well as a high level of stability. The results obtained in this work show that GA-based [2] has some drawbacks, such as early convergence and the solution quality also deteriorates with the increase of population size in the search space. PSO avoids these issues and has a shorter execution time, and the standard deviations of its results are always lower than those produced by the GA.

However, there are some limitations if we only use GA or PSO to solve optimization problems. For example, GA may fall into a state of local search, while in a few special cases the PSO is only good for global optimization but not good for local optimization. To overcome these two restrictions, we proposed two possible solutions. The first, to change the calculation of fitness function: taking the average instead of taking the lowest or highest in PSO, so it improves the diversity of populations, increases accuracy and overcome in only good for global optimization PSO; In the second, we use an emigrant multi-swarm approach for particle swarm optimization algorithm: a case study on extracting multiple choice tests and we also aim to apply this approach and improve the accuracy of the method presented in the current study.

In this paper, we proposed a new and better solution for single objective optimization problem for test extraction. However, as the difficulty and size of question bank increase, the amount of time needed to produce a test may grow very large. Thus, future studies may focus on solving optimization problem with multiple objectives in combination with parallelism and migration for large question banks.

REFERENCES

1. M. Yildirim, "Heuristic optimization methods for generating test from a question bank," *Advances in Artificial Intelligence*, Vol. 4827, 2007, pp. 1218-1229.
2. M. Yildirim, "A genetic algorithm for generating test from a question bank," *Computer Applications in Engineering Education*, Vol. 18, 2010, pp. 298-305.
3. N. H. I. Teo, N. A. Bakar, and S. Karim, "Designing GA-based auto-generator of examination questions," in *Proceedings of IEEE 6th UKSim/AMSS European Symposium on Computer Modeling and Simulation*, 2012, pp. 60-64.
4. N. H. I. Teo, N. A. Bakar, and M. R. A. Rashid, "Representing examination question knowledge into genetic algorithm," in *Proceedings of IEEE Global Engineering Education Conference*, 2014, pp. 900-904.
5. D. V. Paul, S. B. Naik, and J. D. Pawar, "An evolutionary approach for question selection from a question bank: A case study," *International Journal of ICT Research and Development in Africa*, Vol. 4, 2014, pp. 61-75.
6. D. V. Paul and J. D. Pawar, "Elitist-multi-objective differential evolution for multiple question paper generation," *International Journal of Web Applications*, Vol. 6, 2014, pp. 43-56.

7. J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks*, 1995, pp. 1942-1948.
8. S. C. Cheng, Y. T. Lin, and Y. M. Huang, "Dynamic question generation system for web-based testing using particle swarm optimization," *Expert Systems with Applications*, Vol. 36, 2009, pp. 616-624.
9. S. Nootyaskool, "The hybrid implementation genetic algorithm with particle swarm optimization to solve the unconstrained optimization problems," in *Proceedings of the 4th International Conference on Knowledge and Smart Technology*, 2012, pp. 57-61.
10. M. A. Sahnehsaraei, M. J. Mahmoodabadi, M. Taherkhorsandi, K. K. Castillo-Villar, and S. M. Mortazavi Yazdi, "A hybrid global optimization algorithm: particle swarm optimization in association with a genetic algorithm," *Studies in Fuzziness and Soft Computing*, Vol. 319, 2015, pp. 45-86.
11. K. Premalatha and A. M. Natarajan, "Hybrid PSO and GA for global maximization," *International Journal of Open Problems Computer Science and Mathematics*, Vol. 2, 2009, pp. 597-608.
12. R. Thangaraj, M. Pant, A. Abraham, and P. Bouvry, "Particle swarm optimization: Hybridization perspectives and experimental illustrations," *Applied Mathematics and Computation*, Vol. 127, 2011, pp. 5208-5226.
13. N. B. Guedria, "Improved accelerated PSO algorithm for mechanical engineering optimization problems," *Applied Soft Computing*, Vol. 40, 2016, pp. 455-467.
14. H. Izakian and W. Pedrycz, "A new PSO-optimized geometry of spatial and spatio-temporal scan statistics for disease out break detection," *Swarm and Evolutionary Computation*, Vol. 4, 2012, pp. 1-11.
15. M. Pant, R. Thangaraj, and A. Abraham, "DE-PSO: a new hybrid meta-heuristic for solving global optimization problems," *New Mathematics and Natural Computation*, Vol. 7, 2011, pp. 363-381.
16. M. R. Tanweer, R. Auditya, S. Suresh, N. Sundararajan, and N. Srikanth, "Directionally driven self-regulating particle swarm optimization algorithm," *Swarm and Evolutionary Computation*, Vol. 28, 2016, pp. 98-116.
17. F. Zou, D. Chen, R. Lu, and P. Wang, "Hierarchical multi-swarm cooperative teaching-learning-based optimization for global optimization," *Soft Computing*, 2016, pp. 1-22.
18. D. Wang, D. Tan, and L. Liu, "Particle swarm optimization algorithm: an overview," *Soft Computing*, 2017, pp. 1-22.
19. A. M. Faizah, "The use of reflective journals in outcome based education during the teaching practicum," *Malaysian Journal of ELT Research*, Vol. 4, 2008, pp. 32-42.

Toan Bui is currently a M.Sc. student in Computer Science from University of Information Technology – Viet Nam National University HCMC. His research interests include evolutionary algorithm, particle swarm optimization, genetic algorithm, grid computing, parallel computing. E-mail: bm.toan@hutech.edu.vn

Tram Nguyen received her M.Sc. degree in Computer Science from University of Information Technology – Viet Nam National University HCMC in 2013. She is currently a Ph.D. student at VŠB-Technical University of Ostrava, Czech Republic. Her research interests include evolutionary algorithm, particle swarm optimization, genetic algorithm, grid computing, parallel computing. E-mail: phuongtram.itnl@gmail.com

Bay Vo is an Associate Professor from 2015. He received his Ph.D. degree in Computer Science from the University of Science, Vietnam National University of Ho Chi Minh, in 2011. His research interests include association rule mining, classification, incremental mining, distributed databases, and privacy preserving in data mining. He serves as an associate editor of the ICIC Express-Letters, Part B: Applications, a member of the review board of the International Journal of Applied Intelligence, and an editor of the International Journal of Engineering and Technology Innovation. He also served as co-chair of several special sessions such as ICCCI 2012; ACIIDS 2013, 2014, 2015, 2016; KSE 2013, 2014; SMC 2015; as reviewer of many international journals such as IEEE-TKDE, KAIS, ESWA, IEEE-SMC: Systems, Information Sciences, Knowledge Based Systems, Soft Computing, JISE, PLOS ONE, IEEE Access, *etc.* E-mail: vodinhbay@tdt.edu.vn

Thanh Nguyen is currently a software engineer. His research interests include evolutionary algorithm, particle swarm optimization, genetic algorithm, parallel computing. E-mail: nv.thanh0515 @gmail.com

Witold Pedrycz is a Professor and Canada Research Chair (CRC) in Computational Intelligence in the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada. He is also with the Systems Research Institute of the Polish Academy of Sciences, Warsaw, Poland. He holds an appointment of special professorship in the School of Computer Science, University of Nottingham, UK. In 2009 Dr. Pedrycz was elected a foreign member of the Polish Academy of Sciences. In

2012 he was elected a Fellow of the Royal Society of Canada. Witold Pedrycz has been a member of numerous program committees of IEEE conferences in the area of fuzzy sets and neurocomputing. In 2007 he received a prestigious Norbert Wiener award from the IEEE Systems, Man, and Cybernetics Council. He is a recipient of the IEEE Canada Computer Engineering Medal 2008. In 2009 he has received a Cajastur Prize for Soft Computing from the European Centre for Soft Computing for "pioneering and multifaceted contributions to Granular Computing". In 2013 he was awarded a Killam Prize. In the same year he received a Fuzzy Pioneer Award 2013 from the IEEE Computational Intelligence Society. His main research directions involve computational intelligence, fuzzy modeling and Granular Computing, knowledge discovery and data mining, fuzzy control, pattern recognition, knowledge-based neural net-works, relational computing, and software engineering. He has published numerous papers in this area. He is also an author of 15 research mono-graphs covering various aspects of Computational Intelligence, data mining, and Software Engineering. E-mail: wpedrycz@ualberta.ca

Vaclav Snasel's research and development experience includes over 25 years in the Industry and Academia. He works in a multidisciplinary environment involving artificial intelligence, multidimensional data indexing, conceptual lattice, information retrieval, semantic web, knowledge management, data compression, machine intelligence, neural network, web intelligence, data mining and applied to various real-world problems. He has given more than 10 plenary lectures and conference tutorials in these areas. He has authored/co-authored several refereed journal/conference papers and book chapters. He has published more than 400 papers (147 are recorded at Web of Science). He has supervised many Ph.D. students from Czech Republic, Jordan, Yemen, Slovakia, Ukraine and Vietnam.

From 2001 he is a visiting scientist in the Institute of Computer Science, Academy of Sciences of the Czech Republic. From 2003 he is vice-dean for Research and Science at Faculty of Electrical Engineering and Computer Science, VSB-Technical University of Ostrava, Czech Republic. He is full professor since 2006. Before turning into a full-time academic, he was working with industrial company where he was involved in different industrial research and development projects for nearly 8 years. He received Ph.D. degree in Algebra and Geometry from Masaryk University, Brno, Czech Republic and a Master of Science degree from Palacky University, Olomouc, Czech Republic.

Besides, the Editor-in-Chief of two journals, he also serves the editorial board of some reputed International journals. He is actively involved in the International Conference on Computational Aspects of Social Networks (CASoN); Computer Information Systems and Industrial Management (CISIM); Evolutionary Techniques in Data Processing (ETID) series of International conferences. He is a Member of IEEE, ACM, AMS and SIAM. E-mail: vaclav.snasel@vsb.cz