Hindawi
Scientific Programming
Volume 2020, Article ID 7081653, 15 pages
https://doi.org/10.1155/2020/7081653

Research Article

Multiswarm Multiobjective Particle Swarm Optimization with Simulated Annealing for Extracting Multiple Tests

Toan Bui,[1] Tram Nguyen,[2,3] Huy M. Huynh,[4] Bay Vo,[1] Jerry Chun-Wei Lin,[5] and Tzung-Pei Hong[6,7]

[1]*Faculty of Information Technology, Ho Chi Minh City University of Technology (HUTECH), Ho Chi Minh, Vietnam*
[2]*Faculty of Information Technology, Nong Lam University, Ho Chi Minh, Vietnam*
[3]*Department of Computer Science, Faculty of Electrical Engineering and Computer Science, VŠB-Technical University of Ostrava, Ostrava, Czech Republic*
[4]*Institute of Research and Development, Duy Tan University, Da Nang 550000, Vietnam*
[5]*Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen, Norway*
[6]*Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung, Taiwan*
[7]*Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan*

Correspondence should be addressed to Bay Vo; vd.bay@hutech.edu.vn

Received 5 February 2020; Revised 8 May 2020; Accepted 8 July 2020; Published 1 August 2020

Academic Editor: Kifayat Ullah Khan

Education is mandatory, and much research has been invested in this sector. An important aspect of education is how to evaluate the learners' progress. Multiple-choice tests are widely used for this purpose. The tests for learners in the same exam should come in equal difficulties for fair judgment. Thus, this requirement leads to the problem of generating tests with equal difficulties, which is also known as the specific case of generating tests with a single objective. However, in practice, multiple requirements (objectives) are enforced while making tests. For example, teachers may require the generated tests to have the same difficulty and the same test duration. In this paper, we propose the use of Multiswarm Multiobjective Particle Swarm Optimization (MMPSO) for generating k tests with multiple objectives in a single run. Additionally, we also incorporate Simulated Annealing (SA) to improve the diversity of tests and the accuracy of solutions. The experimental results with various criteria show that our approaches are effective and efficient for the problem of generating multiple tests.

1. Introduction

In the education sector, evaluation of students' study progress is important and mandatory. There are many methods such as oral tests or writing tests to evaluate their knowledge and understanding about subjects. Because of the scalability and ease of human resources, writing tests are used more widely for the final checkpoints of assessment (e.g., final term tests), where a large number of students must be considered. Writing tests can be either descriptive tests, in which students have to fully write their answers, or multiple-choice tests, in which students pick one or more choices for each question. Even though descriptive tests are easier to create at first, they consume a great deal of time and effort during the grading stage. Multiple-choice tests, on the other hand, are harder to create at first as they require a large number of questions for security reasons, as in Ting et al. [1]. However, the grading process can be extremely fast, automated by computers, and bias-free from human graders. Recently, many researchers have invested their efforts to make computers automate the process of creating multiple-choice tests using available question banks, as in the work of Cheng et al. [2]. The results were shown to be promising and, thus, make multiple-choice tests more feasible for examinations.

One of the challenges in generating multiple-choice tests is the difficulty of the candidate tests. The tests for all students should have the same difficulty for fairness. However, it can be

seen that generating all tests having the same level of difficulties is an extremely hard task, even in the case of manually choosing questions from a question bank, and the success rate of generating multichoice tests satisfying a given difficulty is low and time-consuming. Therefore, to speed up the process, some authors chose to automatically generate tests with the use of computers and approximate the difficulties of the required difficulties. This is also known as generating tests with a single objective where the level of difficulty is the objective. For example, Bui et al. [3] proposed the use of particle swarm optimization to generate tests with approximating difficulties to the required levels from users. The tests are generated from question banks that consist of various questions with different difficulties. The difficulty value of each question is judged and adapted based on users via previous real-life exams. The work evaluates three random oriented approaches, which are Genetic Algorithms (GAs) by Yildirim [4, 5] and Particle Swarm Optimization (PSO). The experiment result shows that PSO gives the best performance concerning most of the criteria by Bui et al. [3]. Previous works only focused on solving a single objective of the extracting test based on the difficulty level requirement of the user defined. In practice, exam tests can depend on multiple factors such as questions' duration and total testing time. Thus, designing a method that can generate tests with multiple objectives is challenging. Furthermore, the proposed approaches can only extract a single test at each run. To extract multiple tests, the authors have to execute their application multiple times. This method is time-consuming, and duplicate tests can occur because each run is executed separately. Besides, they do not have any information about each other to avoid duplication.

In this paper, we propose a new approach that uses Multiswarm Multiobjective Particle Swarm Optimization (MMPSO) to extract k tests in a single run with multiple objectives. Multiswarms are the same as the multitest in extracting k tests. However, they are based on their search on multiple subswarms instead of one standard swarm that executes their application multiple times to extract multiple tests. The use of diverse subswarms to increase performance when optimizing their tests is studied in Antonio and Chen [6]. Additionally, we use Simulated Annealing (SA) to initialize the first population for PSO to increase the diversities of generated tests. We also aim to improve the results on various criteria such as diversities of solutions and accuracy.

The main contributions of this paper are as follows:

(1) We propose a multiswarm multiobjective approach to deal with the problem of extracting k tests simultaneously.

(2) We propose the use of SA in combining with PSO for extracting tests. SA was selected as it is capable of escaping local optimum solutions.

(3) We propose a parallel version of our serial algorithms. Using parallelism, we can control the overlap of extracted tests to save time.

The rest of this paper is organized as follows. Section 2 describes the related research. The problem of extracting k tests from question banks is explained in Section 3. Correlated studies of normal multiswarm multiobjective PSO and multiswarm multiobjective PSO with SA for the problem of extracting k tests from question banks are presented in Sections 4 and 5. The next section analyzes and discusses the experimental results of this study. Finally, the future research trends, and the conclusions of the paper are provided in Section 6.

2. Related Work

Recently, evolutionary algorithms have been applied to many fields for optimization problems. Some of the most well-known algorithms are Genetic Algorithms (GAs) and Particle Swarm Optimization (PSO). GAs were invented based on the concept of Darwin's theory of evolution, and they seek solutions via progressions of generations. Heuristic information is used for navigating the search space for potential individuals, and this can achieve globally optimal solutions. Since then, there have been many works that used GAs in practice [7–11].

Particle swarm optimization is a swarm-based technique for optimization that is developed by Eberthart and Kennedy [12]. It imitates the behavior of a school of fishes or the flock of birds. PSO optimizes the solutions via the movements of individuals. The foundation of PSO's method of finding optima is based on the following principles proposed by Eberhart and Kennedy: (1) All individuals (particles) in swarms tend to find and move towards possible attractors; (2) each individual remembers the position of the best attractor that it found. In particular, each solution is a particle in a swarm and is denoted by two variables. One is the current location, denoted by present[], and the other is the particle's velocity, denoted by v[]. They are two vectors on the vector space R^n, in which n changes based on the problems. Additionally, each particle has a fitness value that is given by a chosen fitness function. At the beginning of the algorithm, the initial generation (population) is created either in a random manner or by some methods. The movement of each particle individual is affected by two information sources. The first is P_{best}, which is the best-known position of the particle visited in the past movements. The second is G_{best}, which is the best-known position of the whole swarm. In the original work proposed by Eberhart and Kennedy, particles traverse the search space by going after the particles with strong fitness values. Particularly, after disjointed periods, the velocity and position of each individual are updated with the following formulas:

$$\begin{aligned} v^{t+1} &= v^t + c1.\text{rand} * P^t_{\text{best}} - \text{present}^t \\ &\quad + c2.\text{rand} * G^t_{\text{best}} - \text{present}^t, \\ \text{present}^{t+1} &= \text{persent}^t + v^{t+1} \end{aligned} \tag{1}$$

where rand() is a function that returns a random number in the range (0,1) and $c1$, $c2$ are constant weights.

While PSO is mostly used for the continuous value domain, recently, some works have shown that PSO can also be prominently useful for discrete optimization. For

example, Sen and Krishnamoorthy [13, 14] transformed the original PSO into discrete PSO for solving the problem of transmitting information on networks. The work result proves that the proposed discrete PSO outperforms Simulated Annealing (SA).

To further improve the performance for real-life applications, some variants of PSO have been proposed and exploited such as multiswarm PSO. Peng et al. [15] proposed an approach for multiswarm PSO that pairs the velocity update of some swarms with different methods such as the periodically stochastic learning strategy or random mutation learning strategy. The experiments have been run on a set of specific benchmarks. The results showed that the proposed method gives a better quality of solutions and has a higher chance of giving correct solutions than normal PSO. Vafashoar and Meibodi [16] proposed an approach that uses Cellular Learning Automata (CLA) for multiswarm PSO. Each swarm is placed on a cell of the CLA, and each particle's velocity is affected by some other particles. The connected particles are adjusted overtime via periods of learning. The results indicate that the proposed method is quite effective for optimization problems on various datasets. In order to balance the search capabilities between swarms, Xia et al. [17] used multiswarm PSO in combination with various strategies such as the dynamic subswarm number, subswarm regrouping, and purposeful detecting. Nakisa et al. [18] proposed a strategy to improve the speed of convergence of multiswarm PSO for robots' movements in a complex environment with obstacles. Additionally, the authors combine the local search strategy with multiswarm PSO to prevent the robots from converging at the same locations when they try to get to their targets.

In practice, there exist a lot of optimization problems with multiple objectives instead of a single objective. Thus, a lot of work for multiobjective optimization has been proposed. For example, Li and Babak [19] proposed multiobjective PSO combining with an enhanced local search ability and parameter-less sharing. Kapse and Krishnapillai [20] also proposed an adaptive local search method for multiobjective PSO using the time variance search space index to improve the diversity of solutions and convergence. Based on crowded distance sorting, Cheng et al. [21] proposed an improved version, circular crowded sorting, and combined with multiobjective PSO. The approach scatters the individuals of initial populations across the search space in order to be better at gathering the Pareto frontier. The method was proven to improve the search capabilities, the speed of convergence, and diversity of solutions. Similarly, Adel et al. [22] used multiobjective with uniform design instead of traditional random methods to generate the initial population. Based on $R2$ measurement, Alan et al. [23] proposed an approach that used $R2$ as an indicator to navigate swarms through the search space in multiobjective PSO. By combining utopia point-guided search with multiobjective PSO, Kapse and Krishnapillai [24] proposed a strategy that selects the best individuals that are located near the utopia points. The authors also compared their method with other algorithms such as NSGA-II (Nondominated Sorting Genetic Algorithm II) by Deb et al. [25] or CMPSO (Coevolutionary multiswarm PSO) by Zhan et al. [26] on several benchmarks and demonstrated the proposed method's effectiveness. Saxena and Mishra [27] designed a multiobjective PSO algorithm named MOPSO tridist. The algorithm used triangular distance to select leader individuals which cover different regions in Pareto frontier. The authors also included an update strategy for Pbest with respect to their connected leaders. MOPSO tridist was shown to outperform other multiobjective PSOs, and the authors illustrated the algorithm's application with the digital watermarking problem for RBG images. Based on chaotic particle swarm optimization, Liansong and Dazhi [28] designed a multiobjective optimization for chaotic particle swarm optimization and based on comprehensive learning particle swarm optimization, and Xiang and Xueqing [29] proposed an extension, the MSCLPSO algorithm, and incorporated various techniques from other evolutionary algorithms. In order to increase the flexibility of multiobjective PSO, Mokarram and Banan [30] proposed the FC-MOPSO algorithm that can work on a mix-up of constrained, unconstrained, continuous, and/or discrete optimization problems. Recently, Mohamad et al. [31] reviewed and summarized the disadvantages of multiobjective PSO. Based on that, they proposed an algorithm, M-MOPSO. The authors also proposed a strategy based on dynamic search boundaries to help escape the local optima. M-MOPSO was proven to be more efficient when compared with several state-of-the-art algorithms such as Multiobjective Grey Wolf Optimizer (MOGWO), Multiobjective Evolutionary Algorithm based on Decompositions (MOEA/D), and Multiobjective Differential Evolution (MODE).

An extension of multiple objective optimization problems is the dynamic multiple objective optimization problems, in which each objective would change differently depending on the time or environment. To deal with this problem, Liu et al. [32] proposed CMPSODMO which is based on the multiswarm coevolution strategy. The author also combined it with special boundary constraint processing and a velocity update strategy to help with the diversity and convergence speed.

To make it easier for readers, Table 1 summarizes different application domains in which PSO algorithms have been applied for different purposes.

The abovementioned works can be effective and efficient for the optimization problems in Table 1; however, applying them for the problem of generating k test in a single run with multiple objectives is not feasible according to the work of Nguyen et al. [33]. Therefore, in this work, we propose an approach that uses Multiswarm Multiobjective Particle Swarm Optimization (MMPSO) combined with Simulated Annealing (SA) for generating k tests with multiple objectives. Each swarm, in this case, is a test candidate, and it runs on a separate thread. The migration happens randomly by chance. We also aim to improve the accuracy and diversity of solutions.

Table 1: Summarizes different application domains in which PSO algorithms have been applied for different purposes.

Categories	References	Algorithm	Types of optimization problems								Modern optimization techniques
			Unconstrained	Constrained	Continuous	Discrete	Single-objective	Multiobjective	Single-swarm	Multiswarm	
Academia and scientometrics	[22]	Particle swarm optimization Based on multiobjective functions with uniform design (MOPSO-UD)		X	X			X			Multiobjective PSO with uniform design generates the initial population instead of traditional random methods
	[23]	*R*2-based multi/many-objective particle swarm optimization		X	X			X			The proposed approach used *R*2 as an indicator to navigate swarms through the search space in multiobjective PSO
	[21]	An improved version, circular crowded sorting, and combined with multiobjective PSO		X	X			X			The individuals of initial populations across the search space to be better at gathering the Pareto frontier
	[20]	An adaptive local search method for multiobjective PSO		X	X			X			An adaptive local search method for multiobjective PSO using the time variance search space index to improve the diversity of solutions and convergence
	[24]	Combining utopia point-guided search with multiobjective PSO		X	X			X			A strategy that selects the best individuals that are located near the utopia points
	[19]	A novel MOPSO with enhanced local search ability and parameter-less sharing		X	X			X			The proposed approach estimates the density of the particles' neighborhood in the search space. Initially, the proposed method accurately determines the crowding factor of the solutions; in later stages, it effectively guides the entire swarm to converge close to the true Pareto front
	[28]	Chaotic particle swarm optimization		X	X			X			The work improves the diversity of the population and uses simplified mesh reduction and gene exchange to improve the performance of the algorithm
	[32]	A coevolutionary technique based on multiswarm particle swarm optimization		X	X			X	X		The authors combined their proposed algorithm with special boundary constraint processing and a velocity update strategy to help with the diversity and convergence speed
	[31]	Particle swarm optimization algorithm based on dynamic boundary search for constrained optimization		X	X			X			The authors proposed a strategy based on dynamic search boundaries to help escape the local optima
	[30]	A new PSO-based algorithm (FC-MOPSO)	X	X	X	X	X	X			FC-MOPSO algorithm can work on a mix-up of constrained, unconstrained, continuous and/or discrete, single-objective, multiobjective optimization problems algorithm that can work on a mix-up of constrained, unconstrained, continuous, and/or discrete optimization problems
	[15]	A novel particle swarm optimization algorithm with multiple learning strategies (PSO-MLS)		X	X		X			X	The authors proposed an approach for multiswarm PSO that pairs the velocity update of some swarms with different methods such as the periodically stochastic learning strategy or random mutation learning strategy.
	[16]	Cellular Learning Automata (CLA) for multiswarm PSO		X	X		X			X	Each swarm is placed on a cell of the CLA, and each particle's velocity is affected by some other particles. The connected particles are adjusted overtime via periods of learning
	[17]	Improved particle swarm optimization algorithm based on dynamical topology and purposeful detecting.		X	X		X				In order to balance the search capabilities between swarms. The extensive experimental results illustrate the effectiveness and efficiency of the three proposed strategies used in MSPSO
	[29]	Particle swarm optimization with differential evolution (DE) strategy		X	X			X		X	The purpose is to achieve high-performance multiobjective optimization
	[26]	Coevolutionary multiswarm PSO		X	X		X				To modify the velocity update equation to increase search information and diversity solutions to avoid local Pareto front. The results show superior performance in solving optimization problems

Table 1: Continued.

Categories	References	Algorithm	Types of optimization problems								Modern optimization techniques
			Unconstrained	Constrained	Continuous	Discrete	Single-objective	Multiobjective	Single-swarm	Multiswarm	
	[18]	Particle swarm optimization with local search		X		X	X			X	The authors proposed a strategy to improve the speed of convergence of multiswarm PSO for robots' movements in a complex environment with obstacles. Additionally, the authors combine the local search strategy with multiswarm PSO to prevent the robots from converging at the same locations when they try to get to their targets
Application (artificial intelligence)	[27]	Based on new leader selection strategy to improved particle swarm optimization		X	X			X			The algorithm used triangular distance to select leader individuals that cover different regions in the Pareto frontier. The authors also included an update strategy for P_{best} with respect to their connected leaders. MOPSO tridist was shown to outperform other multiobjective PSOs, and the authors illustrated the algorithm's application with the digital watermarking problem for RBG images
	[13, 14]	Discrete PSO		X		X	X				For solving the problem of transmitting information on networks. The work result proves that the proposed discrete PSO outperforms Simulated Annealing (SA)
	[2]	Novel approach of particle swarm optimization (PSO)		X		X		X	X		The dynamic question generation system is built to select tailored questions for each learner from the item bank to satisfy multiple assessment requirements. The experimental results show that the PSO approach is suitable for the selection of near-optimal questions from large-scale item banks
Application (multichoice question test extraction)	[3]	Particle swarm optimization (PSO)		X	X	X	X		X		The authors used particle swarm optimization to generate tests with approximating difficulties to the required levels from users. The experiment result shows that PSO gives the best performance concerning most of the criteria
	[33]	Multiswarm single-objective particle swarm optimization		X	X	X		X	X		The authors use particle swarm optimization to generate multitests with approximating difficulties to the required levels from users. In this parallel stage, migration happens between swarms to exchange information between running threads to improve the convergence and diversities of solutions

3. Problem Statement

3.1. Problem of Generating Multiple Tests. In our previous works [3, 33], we have proposed a PSO-based method to multichoice test generation; however, it was a single-objective approach. In this paper, we introduce a multiobjective approach of multichoice test generation by combining PSO and SA algorithms.

Let $Q = \{q_1, q_2, q_3, \ldots, q_n\}$ be a question bank with n questions. Each question $q_i \in Q$ contains four attributes {QC, SC, QD, OD}. QC is a question identifier code and is used to avoid duplication of any question in the solution. SC denotes a section code of the question and is used to indicate which section the question belonged to. QD denotes a time limit of the question, and OD denotes a real value in the range [0.1, 0.9] that represents an objective difficulty (level) of the question. QC, SC, and QD are discrete positive integer values as in the work of Bui et al. [3] and Nguyen et al. [33].

The problem of generating multiple k tests (or just multiple tests) is to generate k number of tests simultaneously in a single run, e.g., our objective is to generate a set of tests, in which each test $E_i = \{q_{i1}, q_{i2}, q_{i3}, \ldots, q_{im}\}$ ($q_{ij} \in Q$, $1 \le j \le m$, $1 \le i \le k$, $k \le n$) consists of m ($m \le n$) questions. Additionally, those tests must satisfy both the requirements of objective difficulty ODR and testing time duration TR that were given by users. For example, ODR = 0.8 and TR = 45 minutes mean that all the generated tests must have approximately the level of difficulty equal to 0.8 and the test time equal to 45 minutes.

The objective difficulty of a test E_i is defined as $\mathrm{OD}_{E_i} = \sum_{j=1}^{m} q_{ij} \cdot \mathrm{OD}/m$, and the duration of the test E_i is determined by $T_{E_i} = \sum_{j=1}^{m} q_{ij} \cdot \mathrm{QD}$.

Besides the aforementioned requirements, there are additional constraints each generated test must satisfy as follows:

$C1$: each question in a generated test must be unique (i.e., a question cannot appear more than once in a test).

$C2$: in order to make the test more diverse, there exists no case that all questions in a test have the same difficulty value as the required objective difficulty ODR. For example, if ODR = 0.6, then $\exists\, q_{ki} \in T_{E_k}$: $q_{ki} \cdot \mathrm{OD} \neq 0.6$.

$C3$: some questions in a question bank must stay in the same groups because their content is relating to each other. The generated tests must ensure that all the questions in one group appear together. This means if a question of a specific group appears in a test, the remaining questions of the group must also be presented in the same test [3, 33].

$C4$: as users may require generated tests to have several sections, a generated test must ensure that the required numbers of questions are drawn out from question banks for each section.

3.2. Modeling MMPSO for the Problem of Generating Multiple Tests. The model for MMPSO for the problem of generating multiple tests can be represented as follows:

$$\begin{cases} \min f_1(x_1, x_2, x_3, \ldots, x_m), \\ \min f_2(x_1, x_2, x_3, \ldots, x_m), \\ \vdots \\ \min f_k(x_1, x_2, x_3, \ldots, x_m), \end{cases} \tag{2}$$

where $f_1, f_2, \ldots, f_k$ are swarms that represent multiple tests; $x_1, \ldots, x_m$ are the number of questions in the test.

Assume that F is an objective function for multiobjective of the problem; it can be formulated as follows:

$$F = \sum \alpha_i F_i, \tag{3}$$

where α_i is a weight constraint ($\alpha_i \in (0, 1]$) and $\sum \alpha_i = 1$ and F_i is a single-objective function. In this paper, we use an evaluation of the two functions, which are the average levels of difficulty requirements of the tests F_1 and total test duration F_2.

$F_1 = f(q_{kj} \cdot \mathrm{OD}) = (\sum_{j=1}^{m} q_{kj} \cdot \mathrm{OD}/m) - \mathrm{ODR}$, where F_1 satisfies the conditions {$C1$, $C2$, $C3$, $C4$} and m is the total number of questions in the test, $q_{kj} \cdot \mathrm{OD}$ is the difficulty value of each question, and ODR is the required difficulty level.

$F_2 = f(q_{kj} \cdot \mathrm{QD}) = 1 - (\sum_{i=1}^{m} q_{kj} \cdot \mathrm{QD}/\mathrm{TR})$, where F_2 satisfies the conditions {$C1$, $C2$, $C3$, $C4$} and m is the total number of questions in the test, $q_{kj} \cdot \mathrm{QD}$ is the duration for each question, and TR is the required total time of tests.

The objective function F is used as the fitness function in the MMPSO, and the results of the objective function are considered the fitness of the resulting test.

In this case, the better the fitness, the smaller the F becomes. To improve the quality of the test, we also take into account the constraints $C1$, $C2$, $C3$, and $C4$.

For example, provided that we have a question bank as in Table 2, the test extraction requirements are four questions, a difficulty level of 0.6 (ODR = 0.6), a total duration of the test of 300 seconds (TR = 300), and a weight constraint ($\alpha = 0.4$). Table 3 illustrates a candidate solution with its fitness = 0.1 computed by using formula (3).

4. MMPSO in Extracting Multiple Tests

4.1. Process of MMPSO for Extracting Tests. This paper proposes a parallel multiswarm multiobjective PSO (MMPSO) for extracting multiple tests (MMPSO) based on the idea in Bui et al. [3]. It can be described as follows. Creating an initial swarm population is the first step in PSO, in which each particle in a swarm is considered a candidate test; this first population also affects the speed of convergence to optimal solutions. This step randomly picks questions in a question bank. The questions, either stand-alone or staying in groups (constraint $C3$), are drawn out for one section (constraint $C4$) until the required number of questions of the section is reached and the drawing process is repeated for next sections. When the required number of questions of the candidate test and all the constraints are met, the fitness value of the generated test will be computed according to formula (3).

The G_{best} and P_{best} position information is the contained questions. All P_{best} slowly move towards G_{best} by using the

Table 2: The question banks.

QC	01	02	03	04	05	06	07	08	09	10
OD	0.3	0.2	0.8	0.7	0.4	0.6	0.5	0.8	0.2	0.3
QD	100	110	35	40	65	60	60	35	110	100

Table 3: An example of test results.

An individual that satisfies the requirement	QC	05	08	01	04	Fitness (1) (ODR = 0.6; TR = 300; $\alpha = 0.4$)
	OD	0.4	0.8	0.3	0.7	$(0.4 \times \lvert(2.2/4) - 0.6\rvert) + [(1 - 0.4) \times \lvert1 - (240/300)\rvert] = 0.1$
	QD	65	35	100	40	

location information of G_{best}. The movement is the replacement of some questions in the candidate test according to the velocity P_{best}. If the fitness value of a newly found P_{best} of a particle is smaller than the particle's currently best-known P_{best} (i.e., the new position is better than the old), then we assign a newly found position value to P_{best}.

G_{best} moves towards the final optimal solution in random directions. The movement is achieved by replacing its content with some random questions from the question bank. In a similar way to P_{best}, if the new position is no better than the old one, the G_{best} value will not be updated.

The algorithm ends when the fitness value is lower than the fitness threshold ε or the number of movements (iteration loops) surpasses the loop threshold λ. Both of the thresholds are given by users.

4.2. Migration Parallel MMPSO for the Extracting Test (Parallel MMPSO). Based on the idea in Nguyen et al. [33]; we present the migration parallel approach of MMPSO for increasing performance. Each swarm now corresponds to a thread, and the migration happens by chance between swarms. The migration method starts with locking the current thread (swarm) to avoid interference from other threads in.

In the dual-sector model [34], Lewis describes a relationship between two regions, the subsistence sector and the capitalist sector. We can view the two types of economic sectors here as the strong (capitalist) sectors and the weak (subsistence) sectors (while ignoring other related aspects of the economy). Whether a sector is strong or weak depends on the fitness value of G_{best} positions of its swarm. However, when applying those theories, some adjustments are made so that the parallel MMPSO can yield better optimal solutions.

The direction of migration changes when individuals with strong P_{best} (strong individuals) in strong sectors move to weak sectors. The weak sectors' G_{best} may be replaced by the incoming P_{best}, and the fitness value of the weak swarms should make a large lift, as in the work of Nguyen et al. [33].

Backward migration from the weak swarms to strong swarms also happens alongside forwarding migration. For every individual that moves from a strong swarm to a weak swarm, there is always one that moves from the weak swarm back to the strong swarm. This is to ensure that the number of particles and the searching capabilities of the swarms do not significantly decrease.

The foremost condition for migration to happen is that there are changes in the fitness values of the current G_{best} compared to the previous G_{best}.

The probability for migration is denoted as γ, and the unit is a percentage (%).

The number of migrating particles is equal to $\delta \times$ the size of the swarm (i.e., the number of existing particles in the swarm), where δ denotes the percentage of migration.

The migration parallel MMPSO-based approach to extract multiple tests is described in a form of a pseudocode in Algorithm 1.

The particle updates its velocity (V) and positions (P) with the following formulas:

$$V^{t+1} = V^t + r_1 \times V^t_{p\text{best}} + r_2 \times V^t_{g\text{best}}, \tag{4}$$

where $V_{p\text{best}}$ is the velocity of P_{best}, with $V_{p\text{best}}$ determined by $V_{p\text{best}} = \alpha \times m$; $V_{g\text{best}}$ is the velocity of G_{best}, with $V_{g\text{best}}$ determined by $V_{g\text{best}} = \beta \times m$, $\alpha, \beta \in (0, 1)$, r_1, r_2 are random values, and m is the number of questions in the test solutions.

$$P^{t+1} = P^t + V^{t+1}. \tag{5}$$

The process of generating multiple tests at the same time in a single run using migration parallel MMPSO includes two stages. The first stage is generating tests using multi-objective PSO. In this stage, the algorithm proceeds to find tests that satisfy all requirements and constraints using multiple threads. Each thread corresponds to each swarm that runs separately. The second stage is improving and diversifying tests. This stage happens when there is a change in the value of G_{best} of each swarm (for each thread) in the first stage. In this second stage, migration happens between swarms to exchange information between running threads to improve the convergence and diversity of solutions based on the work of Nguyen et al. [33]. The complete flowchart that applies the parallelized migration method to the MMPSO algorithm is shown in Figure 1.

4.3. Migration Parallel MMPSO in Combination with Simulated Annealing. As mentioned above, the initial population affects the convergence speed and diversity of test solutions. The creation of a set of initial solutions (population) is generally performed randomly in PSO. It is one of the drawbacks since the search space is too wide, so the probability of getting stuck in a local optimum solution is

Function Migration_Par_MMPSO: Extracting tests using Migration Parallel MMPSO
For each available thread t **do**
Generate the initial population with random questions;
While stop conditions are not met **do**
Gather G_{best} and all P_{best};
Foreach P_{best}
Move P_{best} towards G_{best} using location information of G_{best},;
Update velocity using equation (4);
Update position using equation (5);
End for
G_{best} moves in a random direction to search for the optimal solution;
If the probability for migration γ is met **then**
Execute function **Migration_MMPSO** with t;
End if
End while
End for
Function Migration _MMPSO: Improving solutions with migration method
Lock the current thread (i.e., block all modifications from other threads to the current thread) to avoid interference from other threads to the current thread during migration procedure.
Select λ, which are the set of stronger individuals for migration except for the G_{best};
Lock other threads so that no unintended changes will happen to them during the migration:
Choose a thread that has a G_{best} weaker than the one in the current thread;
Unlock the other threads except for chosen thread;
Set the status of the chosen thread to "Exchanging";
Move the λ selected individuals to the chosen thread;
Remove those λ selected individuals;
Select the λ weakest individuals in the chosen thread;
Add those λ weakest individuals to the current thread;
Set the status of chosen thread to "Available";
Unlock the current thread and the chosen thread;

Algorithm 1: Pseudocode: migration parallel MMPSO.

also high. In order to improve the initial population, we apply SA in the initial population creation step of migration parallel MMPSO instead of the random method. SA was selected since it is capable of escaping local optimums in Kharrat and Neji [35]. In this study, the process of finding new solutions using SA is improved by moving P_{best} to G_{best} using the received information about the location of G_{best} (which is commonly used in PSO). The MMPSO with SA is described by a pseudocode in Algorithm 2.

5. Experimental Studies

5.1. Experimental Environment and Data. Bui et al. [3] evaluated different algorithms such as the random method, genetic algorithms, and PSO-based algorithm for extracting tests from a question bank of varying sizes. The results of the experiment showed that the PSO-based algorithms are better than others. Hence, the experiment in this paper only evaluated and compared the improved SA parallel MMPSO algorithm with the normal parallel MMPSO algorithm in terms of the diversity of tests and the accuracy of solutions.

All proposed algorithms are implemented in C# and run on 2 computers which are a 2.5 GHz Desktop PC (4-CPUs, 4 GB RAM, Windows 10) and a 2.9 GHz VPS (16-CPUs, 16 GB RAM, Windows Server 2012). The experimental data include 2 question banks. One is with 998 different questions (the small question bank) and the other one is with 12000 different questions (the large question bank). The link to the data is https://drive.google.com/file/d/1_EdCUNyqC9IGziFUIf4mqs0G1qHtQyGI/view. The small question bank consists of multiple sections, and each section has more than 150 questions with different difficulty levels (Figure 2). The large question bank includes 12,000 different questions in which each part has 1000 questions with different difficulty levels (Figure 3). The experimental parameters of MMPSO are presented in Table 4. The results are shown in Tables 5 and 6 and Figures 4 and 5.

Our experiments focus on implementing formula (3) and an evaluation of the two functions, which are the average levels of difficulty requirements of the tests F_1 and total test duration F_2.

5.2. Evaluation Method. In this part, we present the formula for the evaluation of all algorithms about their stability to produce required tests with various weight constraints (α). The main measure is the standard deviation, which is defined as follows:

$$A = \sqrt{\frac{\sum_{i=1}^{z} (y_{i-} \overline{y})^2}{z-1}}, \quad (6)$$

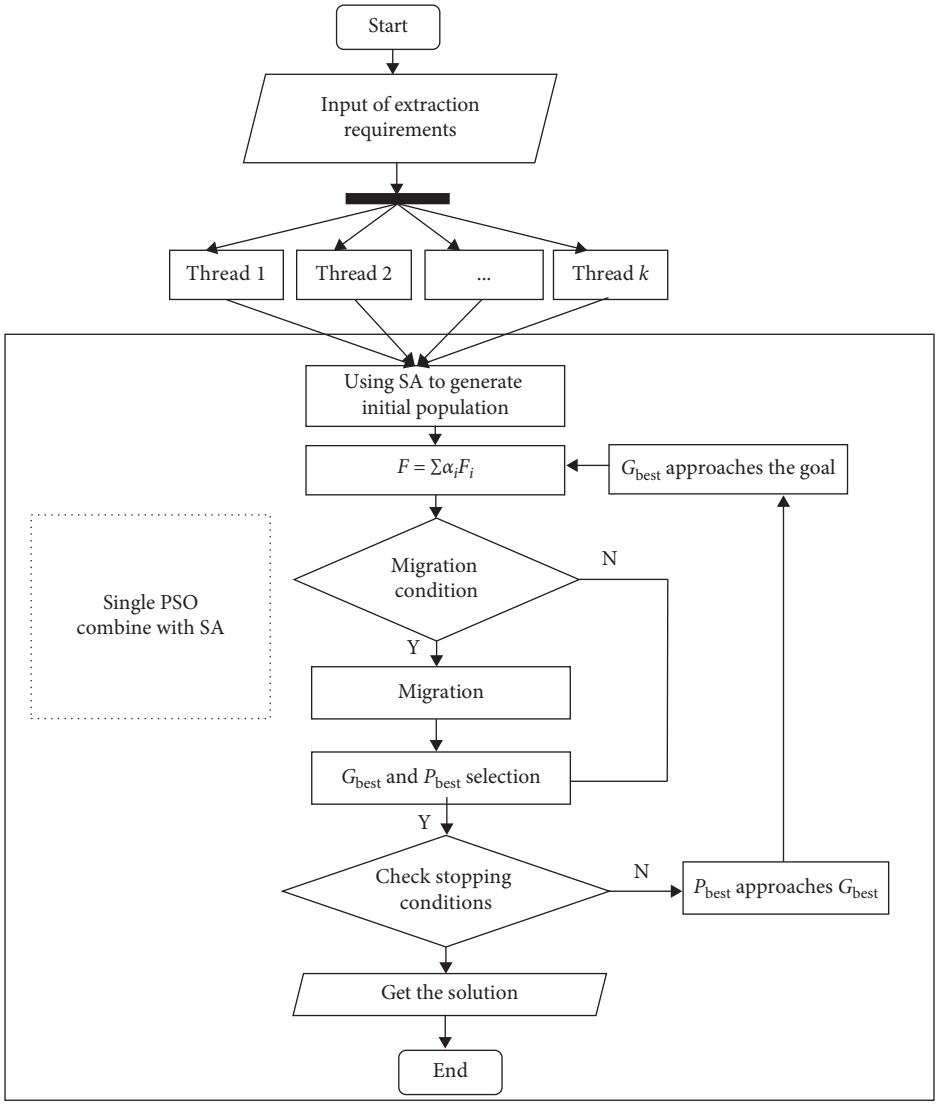

Figure 1: The flowchart of the MMPSO algorithm in migration parallel.

Function Migration_Par_MMPSO_SA: Extracting tests using Migration Parallel MMPSO and SA
For each available thread t **do**
Generate the initial population by using SA as it is capable of escaping local minimums. In this stage, the process of finding new solutions using SA is improved by moving P_{best} to G_{best} using the received information about the location of G_{best} and remove any incorrect solutions;
While stop conditions are not met **do**
Gather G_{best} and all P_{best};
For each P_{best}
Move P_{best} towards G_{best} using location information of G_{best};
Apply velocity update using (4),
Apply position update using (5),
End for
G_{best} moves in a random direction to search for the optimal solution;
If the probability for migration γ is met **then**
Execute function **Migration_MMPSO** with t;
End if
End while
End for

Algorithm 2: Pseudocode: migration parallel MMPSO with SA.

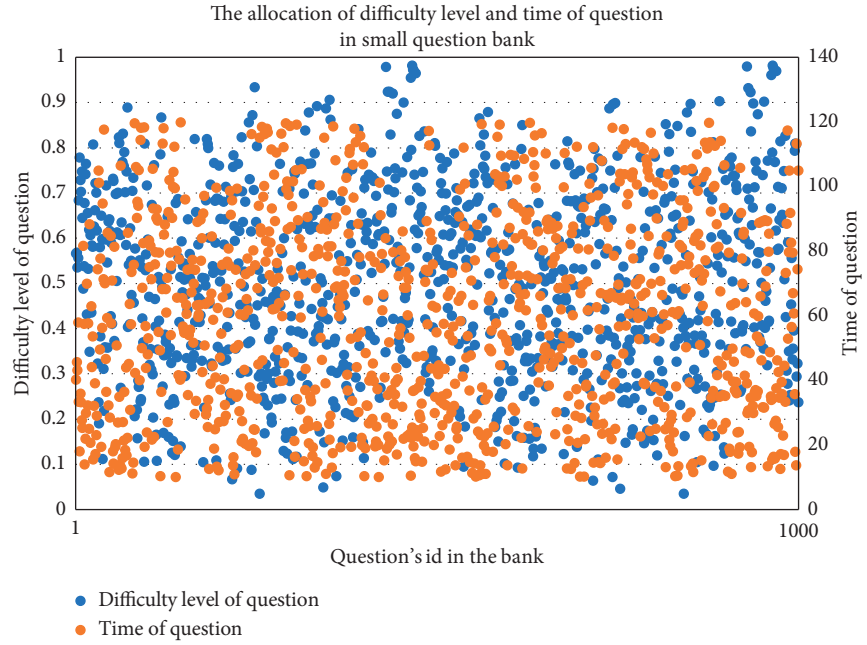

Figure 2: The allocation of the difficulty level and time of question in the small question bank.

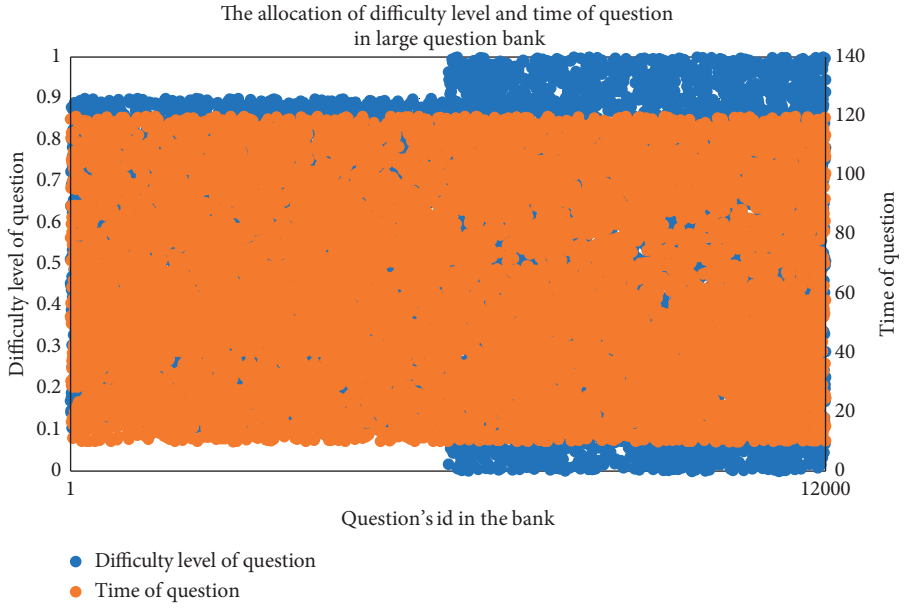

Figure 3: The allocation of the difficulty level and time of question in the large question bank.

Table 4: Experimental parameters.

The required level of difficulty (ODR)	0.5
The total test time (TR)	5400 (seconds)
The value of α	[0.1, 0.9]
The number of required questions in a test	100
The number of questions in each section in the test	10

Table 4: Continued.

The number of simultaneously generated tests in each run	100
The number of questions in the bank	1000 and 12,000
The PSO's parameters	The number of particles in each swarm: 10 Random value $r1$, $r2$ are in [0,1] The percentage of P_{best} individuals which receive position information from G_{best} ($C1$): 5% The percentage of G_{best} which moves to final goals ($C2$): 5% The percentage of **migration** δ: 10% The percentage of **migration probability** γ: 5% The stop condition: either when the tolerance fitness <0.001 or when the number of movement loops >1000
The SA's parameters	Initial temperature: 100 Cooling rate: 0.9 Termination temperature: 0.01 Number of iterations: 100

Table 5: Experimental results in the small question bank.

Algorithms	Weight constraint (α)	Number of runs	Successful times	Average runtime for extracting tests (second)	Average number of iteration loops	Average fitness	Average duplicate (%)	Standard deviation
Parallel multiswarm multiobjective PSO (parallel MMPSO)	0.1	50	11	61.3658	999.75	0.003102	2.43	0.0007071
	0.2	50	445	47.9793	981.03	0.003117	2.64	0.0014707
	0.3	50	425	35.8007	957.53	0.004150	2.73	0.0021772
	0.4	50	530	30.5070	928.10	0.004850	2.80	0.0027973
	0.5	50	774	29.5425	877.65	0.004922	2.85	0.0033383
	0.6	50	1410	22.6973	754.82	0.003965	2.91	0.0034005
	0.7	50	2900	14.9059	470.13	0.002026	2.97	0.0022461
	0.8	50	3005	16.7581	488.31	0.001709	3.01	0.0017271
	0.9	50	3019	28.5975	619.34	0.001358	3.04	0.0009634
Parallel multiswarm multiobjective PSO with SA (parallel MMPSO with SA)	0.1	50	4	142.2539	999.98	0.003080	2.98	0.0006496
	0.2	50	2912	132.3828	900.42	0.001265	3.27	0.0007454
	0.3	50	3681	111.9513	650.42	0.001123	3.38	0.0008364
	0.4	50	3933	100.0204	474.91	0.001085	3.44	0.0009905
	0.5	50	4311	91.7621	318.75	0.000938	3.48	0.0008439
	0.6	50	4776	84.7441	161.23	0.000746	3.53	0.0005124
	0.7	50	4990	81.1127	76.75	0.000666	3.54	0.0002421
	0.8	50	4978	84.6747	131.32	0.000679	3.52	0.0002518
	0.9	50	4937	98.8690	339.89	0.000749	3.41	0.0002338
Migration parallel multiswarm multiobjective PSO (migration parallel MMPSO)	0.1	50	575	51.3890	959.29	0.002138	5.19	0.0008091
	0.2	50	1426	33.2578	837.87	0.002119	5.60	0.0011804
	0.3	50	1518	25.3135	779.76	0.002587	5.82	0.0017130
	0.4	50	1545	21.0524	745.36	0.002977	5.89	0.0021845
	0.5	50	1650	17.9976	710.92	0.003177	5.95	0.0025374
	0.6	50	1751	16.0573	680.97	0.003272	5.92	0.0028531
	0.7	50	2463	12.9467	540.21	0.002243	5.94	0.0022161
	0.8	50	3315	12.7420	402.75	0.001374	5.90	0.0012852
	0.9	50	3631	19.1735	439.27	0.001067	5.85	0.0006259
Migration parallel multiswarm multiobjective PSO with SA (migration parallel MMPSO with SA)	0.1	50	816	139.6821	952.42	0.002183	3.82	0.0009039
	0.2	50	3641	111.4438	638.08	0.001039	5.19	0.0005336
	0.3	50	3958	98.9966	463.53	0.000984	5.36	0.0006349
	0.4	50	4084	92.6536	357.13	0.000973	5.30	0.0007475
	0.5	50	4344	88.3098	255.46	0.000898	5.10	0.0007442
	0.6	50	4703	83.4776	144.00	0.000758	4.90	0.0004939
	0.7	50	4874	81.2981	84.57	0.000697	4.70	0.0003271
	0.8	50	4955	84.2094	106.68	0.000683	4.45	0.0002609
	0.9	50	4937	94.1685	267.30	0.000746	4.19	0.0002345

Table 6: Experimental results in the large question bank.

Algorithms	Weight constraint (α)	Number of runs	Successful times	Average runtime for extracting tests (second)	Average number of iteration loops	Average fitness	Average duplicate (%)	Standard deviation
Parallel multiswarm multiobjective PSO (parallel MMPSO)	0.1	50	2931	23.50	888.85	0.001137	0.95	0.000476
	0.2	50	4999	14.05	484.33	0.000725	1.03	0.000219
	0.3	50	4997	9.56	296.80	0.000689	1.04	0.000234
	0.4	50	4999	5.99	190.55	0.000676	1.04	0.000233
	0.5	50	5000	3.61	121.24	0.000668	1.05	0.000236
	0.6	50	5000	2.77	79.32	0.000663	1.05	0.000235
	0.7	50	5000	3.22	92.33	0.000669	1.05	0.000238
	0.8	50	5000	4.92	173.19	0.000673	1.04	0.000231
	0.9	50	5000	10.98	384.13	0.000738	1.02	0.000213
Parallel multiswarm multiobjective PSO with SA (parallel MMPSO with SA)	0.1	50	3055	99.75	890.40	0.001095	0.96	0.000432
	0.2	50	5000	84.90	469.23	0.000709	1.04	0.000224
	0.3	50	5000	74.43	275.54	0.000686	1.05	0.000230
	0.4	50	5000	73.03	168.91	0.000668	1.06	0.000237
	0.5	50	5000	69.88	99.92	0.000663	1.07	0.000235
	0.6	50	5000	67.34	61.42	0.000662	1.07	0.000236
	0.7	50	5000	53.43	69.02	0.000661	1.07	0.000235
	0.8	50	5000	70.06	132.27	0.000676	1.06	0.000236
	0.9	50	5000	79.58	319.36	0.000734	1.03	0.000211
Migration parallel multiswarm multiobjective PSO (migration parallel MMPSO)	0.1	50	2943	33.52	886.90	0.001144	0.95	0.000482
	0.2	50	4995	19.33	488.60	0.000724	1.02	0.000219
	0.3	50	4998	12.43	295.69	0.000688	1.04	0.000231
	0.4	50	5000	8.57	190.20	0.000667	1.04	0.000238
	0.5	50	4999	6.02	120.88	0.000665	1.05	0.000240
	0.6	50	5000	4.44	78.89	0.000669	1.04	0.000234
	0.7	50	5000	4.98	92.53	0.000668	1.05	0.000234
	0.8	50	5000	7.94	171.98	0.000669	1.04	0.000236
	0.9	50	5000	15.76	383.09	0.000738	1.02	0.000209
Migration parallel multiswarm multiobjective PSO with SA (migration parallel MMPSO with SA)	0.1	50	3122	102.00	888.48	0.001091	0.96	0.000436
	0.2	50	5000	85.68	469.50	0.000716	1.04	0.000222
	0.3	50	5000	77.35	276.03	0.000678	1.05	0.000235
	0.4	50	5000	73.19	167.84	0.000674	1.06	0.000234
	0.5	50	5000	69.62	99.66	0.000665	1.06	0.000238
	0.6	50	5000	67.83	61.33	0.000660	1.07	0.000237
	0.7	50	5000	64.19	69.02	0.000666	1.07	0.000237
	0.8	50	5000	61.46	133.45	0.000673	1.06	0.000234
	0.9	50	5000	71.62	319.68	0.000731	1.03	0.000213

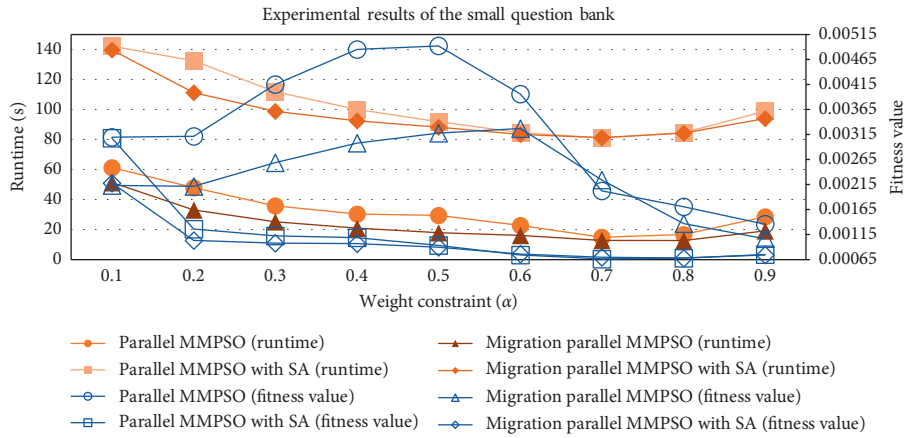

Figure 4: Experimental results of the runtime and fitness value are in Table 4.

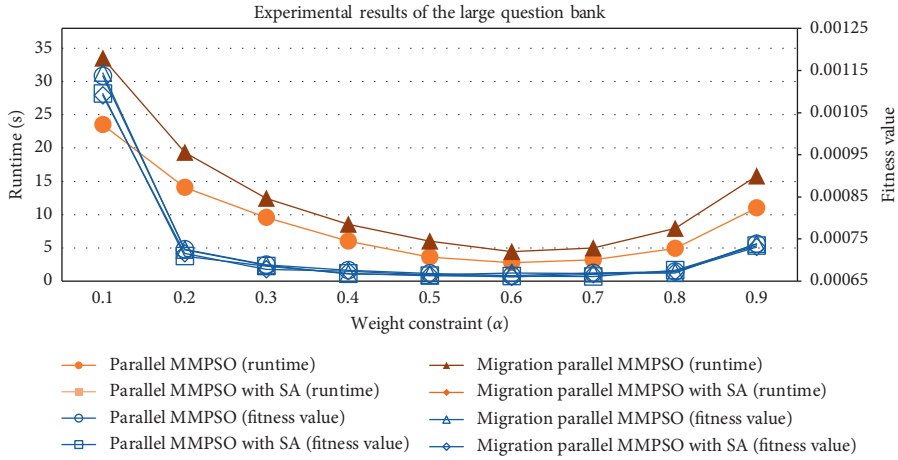

Figure 5: Experimental results of the runtime and fitness value are in Table 5.

where z is the number of experimental runs. $\overline{y}$ is the average fitness of all runs. y_i is a fitness value of run i^{th}.

The standard deviation is used to assess the stability of the algorithms. If its value is low, then the generated tests of each run do not have much difference in the fitness value. The weight constraint α is also being examined as it balances the objective functions. In our cases, a change in α can shift the importance towards the test duration constraint, the test difficulty constraint, or the balance between those two. We can select α to suit what we require, emphasizing more on either the test duration or the test difficulty.

5.3. Experimental Results. The experiments are executed with the parameters following Ridge and Kudenko [36] in Table 3, and the results are presented in Table 5 (run on computer 4-CPUs) and Table 6 (run on computer 16-CPUs). The comparisons of runtime and fitness of the small and large question bank are presented in Figures 4 and 5. Regarding Tables 5 and 6, each run extracts 100 tests simultaneously, and each test has a fitness value. Each run also requires several iteration loops to successfully extract 100 candidate tests. The average runtime for extracting tests is the average runtimes of all 50 experimental runs. The average number of iteration loops is the average of all required loops of all 50 runs. The average fitness is the average of all fitness values of 5000 generated tests. The average duplicate indicates the average number of duplicate questions among 100 generated tests of all 50 runs. The average duplicate is also used to indicate the diversity of tests. The lower the value, the more diverse the tests.

When α is at the lower range [0.1, 0.5], the correctness for difficulty value of each generated test is emphasized more than that of the total test time. Based on the average fitness value, all algorithms appear to have a harder time generating tests at the lower range [0.1, 0.5] compared with at the higher range [0.5, 0.9]. Additionally, when α increases, the runtime starts to decrease, the fitness gets better (i.e., the fitness values get smaller), and the numbers of loops required for generating tests decrease. Apparently, satisfying the requirement for the test difficulty requirement is harder than satisfying the requirement for total test time. The experiment results also show that integrating SA gives a better fitness value without SA. However, runtimes of algorithms with SA are longer as a trade-off for better fitness values.

All algorithms can generate tests with acceptable percentages of duplicate questions among generated tests. The duplicate question proportions between generated tests depend on the sizes of the question bank. For example, if the question bank's size is 100, we need to generate 50 tests in a single run and each test contains 30 questions, and then, some generated tests should contain similar questions of the other generated tests.

Based on the standard deviation in Tables 5 and 6, all MMPSO algorithms with SA are more stable than those without SA since the standard deviation values of those with SA are smaller. In other words, the differences in fitness values between runs are smaller with SA than without SA. The smaller standard deviation values and smaller average fitness values also mean that we less likely need to rerun the MMPSO with SA algorithms many times to get the generated tests that better fit the test requirements. The reason is that the generated tests we obtain at the first run are likely close to the requirements (due to the low fitness value) and the chance that we obtain those generated tests with less fit to requirements is low (due to the low standard deviation value).

6. Conclusions and Future Studies

Generation of question papers through a question bank is an important activity in extracting multichoice tests. The quality of multichoice questions is good (diversity of the level of difficulty of the question and a large number of questions in question bank). In this paper, we propose the use of MMPSO to solve the problem of generating multi-objective multiple k tests in a single run. The objectives of the tests are the required level of difficulty and the required total test time. The experiments evaluate two algorithms, MMPSO with SA and normal MMPSO. The results indicate that MMPSO with SA gives better solutions than normal

MMPSO based on various criteria such as diversities of solutions and numbers of successful attempts.

Future studies may focus on investigating the use of the proposed hybrid approach [37, 38] to solve other NP-hard and combinatorial optimization problems, which focus on fine-tuning the PSO parameters by using some type of adaptive strategies. Additionally, we will extend our problem to provide feedback to instructors from multiple-choice data, such as using fuzzy theory [39], and PSO with SA for mining association rules to compute the difficulty levels of questions.

Data Availability

The data used in this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] F. Ting, J. H. Wei, C. T. Kim, and Q. Tian, "Question classification for E-learning by artificial neural network," in *Proceedings of the Fourth International Conference on Information, Communications and Signal Processing*, pp. 1575–1761, Hefei, China, 2003.

[2] 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, no. 1, pp. 616–624, 2009.

[3] T. Bui, T. Nguyen, B. Vo, T. Nguyen, W. Pedrycz, and V. Snasel, "Application of particle swarm optimization to create multiple-choice tests," *Journal of Information Science & Engineering*, vol. 34, no. 6, pp. 1405–1423, 2018.

[4] M. Yildirim, "Heuristic optimization methods for generating test from a question bank," *Advances in Artificial Intelligence*, pp. 1218–1229, Springer, Berlin, Germany, 2007.

[5] M. Yildirim, "A genetic algorithm for generating test from a question bank," *Computer Applications in Engineering Education*, vol. 18, no. 2, pp. 298–305, 2010.

[6] B. R. Antonio and S. Chen, "Multi-swarm hybrid for multimodal optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1759–1766, Brisbane, Australia, June 2012.

[7] N. Hou, F. He, Y. Zhou, Y. Chen, and X. Yan, "A parallel genetic algorithm with dispersion correction for HW/SW partitioning on multi-core CPU and many-core GPU," *IEEE Access*, vol. 6, pp. 883–898, 2018.

[8] M. E. C. Bento, D. Dotta, R. Kuiava, and R. A. Ramos, "A procedure to design fault-tolerant wide-area damping controllers," *IEEE Access*, vol. 6, pp. 23383–23405, 2018.

[9] C. Han, L. Wang, Z. Zhang, J. Xie, and Z. Xing, "A multi-objective genetic algorithm based on fitting and interpolation," *IEEE Access*, vol. 6, pp. 22920–22929, 2018.

[10] J. Lu, Z. Chen, Y. Yang, and L. V. Ming, "Online estimation of state of power for lithium-ion batteries in electric vehicles using genetic algorithm," *IEEE Access*, vol. 6, pp. 20868–20880, 2018.

[11] X.-Y. Liu, Y. Liang, S. Wang, Z.-Y. Yang, and H.-S. Ye, "A hybrid genetic algorithm with wrapper-embedded approaches for feature selection," *IEEE Access*, vol. 6, pp. 22863–22874, 2018.

[12] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, pp. 1942–1948, Perth, Australia, 1995.

[13] G. Sen, M. Krishnamoorthy, N. Rangaraj, and V. Narayanan, "Exact approaches for static data segment allocation problem in an information network," *Computers & Operations Research*, vol. 62, pp. 282–295, 2015.

[14] G. Sen and M. Krishnamoorthy, "Discrete particle swarm optimization algorithms for two variants of the static data segment location problem," *Applied Intelligence*, vol. 48, no. 3, pp. 771–790, 2018.

[15] M. Q. Peng, Y. J. Gong, J. J. Li, and Y. B. Lin, "Multi-swarm particle swarm optimization with multiple learning strategies," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pp. 15-16, New York; NY, USA, 2014.

[16] R. Vafashoar and M. R. Meybodi, "Multi swarm optimization algorithm with adaptive connectivity degree," *Applied Intelligence*, vol. 48, no. 4, pp. 909–941, 2017.

[17] X. Xia, L. Gui, and Z.-H. Zhan, "A multi-swarm particle swarm optimization algorithm based on dynamical topology and purposeful detecting," *Applied Soft Computing*, vol. 67, pp. 126–140, 2018.

[18] B. Nakisa, M. N. Rastgoo, M. F. Nasrudin, M. Zakree, and A. Nazri, "A multi-swarm particle swarm optimization with local search on multi-robot search system," *Journal of Theoretical and Applied Information Technology*, vol. 71, no. 1, pp. 129–136, 2015.

[19] M. Li and F. Babak, "Multi-objective particle swarm optimization with gradient descent search," *International Journal of Swarm Intelligence and Evolutionary Computation*, vol. 4, pp. 1–9, 2014.

[20] S. P. Kapse and S. Krishnapillai, "An improved multi-objective particle swarm optimization based on utopia point guided search," *International Journal of Applied Metaheuristic Computing*, vol. 9, no. 4, pp. 71–96, 2018.

[21] T. Cheng, M. F. P. J. Chen, Z. Yang, and S. Gan, "A novel hybrid teaching learning based multi-objective particle swarm optimization," *Neurocomputing*, vol. 222, pp. 11–25, 2016.

[22] H. A. Adel, L. Songfeng, E. A. A. Yahya, and A. G. A. Arkan, "A particle swarm optimization based on multi objective functions with uniform design," *SSRG International Journal of Computer Science and Engineering*, vol. 3, pp. 20–26, 2016.

[23] D. M. Alan, T. Gregorio, H. B. Z. Jose, and T. L. Edgar, "R2-based multi/many-objective particle swarm optimization," *Computational Intelligence and Neuroscience*, vol. 2016, Article ID 1898527, 10 pages, 2016.

[24] S. P. Kapse and S. Krishnapillai, "An improved multi-objective particle swarm optimization algorithm based on adaptive local search," *International Journal of Applied Evolutionary Computation*, vol. 8, no. 2, pp. 1–29, 2017.

[25] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[26] Z. H. Zhan, J. Li, and J. Cao, "Multiple populations for multiple objectives: a coevolutionary technique for solving multiobjective optimization problems," *IEEE Transactions on Cybernetics*, vol. 43, no. 2, pp. 445–463, 2013.

[27] N. ShiZhang and K. K. Mishra, "Improved multi-objective particle swarm optimization algorithm for optimizing watermark strength in color image watermarking," *Applied Intelligence*, vol. 47, no. 2, pp. 362–381, 2017.

[28] X. Liansong and P. Dazhi, "Multi-objective optimization based on chaotic particle swarm optimization," *International Journal of Machine Learning and Computing*, vol. 8, no. 3, pp. 229–235, 2018.
[29] Y. Xiang and Z. Xueqing, "Multi-swarm comprehensive learning particle swarm optimization for solving multi-objective optimization problems," *PLoS One*, vol. 12, no. 2, Article ID e, 2017.
[30] V. Mokarram and M. R. Banan, "A new PSO-based algorithm for multi-objective optimization with continuous and discrete design variables," *Structural and Multidisciplinary Optimization*, vol. 57, no. 2, pp. 509–533, 2017.
[31] Z. B. M. Z. Mohd, J. Kanesan, J. H. Chuah, S. Dhanapal, and G. Kendall, "A multi-objective particle swarm optimization algorithm based on dynamic boundary search for constrained optimization," *Applied Soft Computing*, vol. 70, pp. 680–700, 2018.
[32] R. Liu, J. Li, J. Fan, C. Mu, and L. Jiao, "A coevolutionary technique based on multi-swarm particle swarm optimization for dynamic multi-objective optimization," *European Journal of Operational Research*, vol. 261, no. 3, pp. 1028–1051, 2017.
[33] T. Nguyen, T. Bui, and B. Vo, "Multi-swarm single-objective particle swarm optimization to extract multiple-choice tests," *Vietnam Journal of Computer Science*, vol. 6, no. 2, pp. 147–161, 2019.
[34] D. Hunt, "W. A. Lewis on "economic development with unlimited supplies of labour"" *Economic Theories of Development: An Analysis of Competing Paradigms*, pp. 87–95, Harvester Wheatsheaf, New York, NY, USA, 1989.
[35] A. Kharrat and M. Neji, "A hybrid feature selection for MRI brain tumor classification," *Innovations in Bio-Inspired Computing and Applications*, pp. 329–338, Springer, Berlin, Germany, 2018.
[36] E. Ridge and D. Kudenko, "Tuning an algorithm using design of experiments," *Experimental Methods for the Analysis of Optimization Algorithms*, pp. 265–286, Springer, Berlin, Germany, 2010.
[37] P. Riccardo, I. Umberto, L. Giampaolo, and L. Stefano, "Global/local hybridization of the multi-objective particle swarm optimization with derivative-free multi-objective local search," in *Proceedings of the Congress of the Italian Society of Industrial and Applied Mathematics (SIMAI)*, pp. 198–209, At Milan, Italy, 2017.
[38] S. Sedarous, S. M. El-Gokhy, and E. Sallam, "Multi-swarm multi-objective optimization based on a hybrid strategy," *Alexandria Engineering Journal*, vol. 57, no. 3, pp. 1619–1629, 2018.
[39] H. S. Le and H. Fujita, "Neural-fuzzy with representative sets for prediction of student performance," *Applied Intelligence*, vol. 49, no. 1, pp. 172–187, 2019.