Effective Covering Array Generation Using an Improved Particle Swarm Optimization

Zhao Li, Yuhang Chen, Yi Song, Kangjie Lu, and Jinwei Shen

***Abstract*—In the test case generation process of combinatorial testing, particle swarm optimization (PSO) is widely concerned for its simple implementation and fast convergence rate; however, it often falls into local optimum due to premature convergence. To attack this problem, a novel adaptive value measurement strategy is adopted by weighing the relationship between current test cases and historical test cases. The test case with the minimum average hamming distance is selected as the optimal test case, and the inertial weight linear differential decrease strategy is developed to ensure better inertial weight in different search stages, further to improve the capability of generating smaller covering arrays. In addition, we integrate the simulated annealing strategy into the improved PSO to improve the ability of particles jumping out of the local optimum, and an innovative approach for generating a better covering array is proposed. Experiments on 16 classical random strength covering arrays suggest that our approach outperforms six other techniques in terms of effectiveness.**

***Index Terms*—Combinatorial testing, covering array generation, improved particle swarm optimization (PSO), scale reduction, simulated annealing strategy.**

I. Introduction

A NORMAL software testing process needs to test each module, but the extremely high testing cost has greatly increased the overhead of software development [1]. In some scenarios, there are too many influencing factors and parameter values within the software under test, resulting in a huge number of test cases and excessive testing costs for full coverage testing. For instance, in a version of web server, users can configure up to 213 options, and each can be assigned at least two values; if a full coverage testing is conducted, at least 2^{213} test cases are required, which leads to the issue of combinatorial explosion. The combinatorial testing focuses on the software faults caused by the interactions among the inputs [2]−[4], [39]; to solve the above problem, fewer test cases are selected from the huge combination space to cover the interactions of the parameters, which can detect defects with a small number of test cases. As suggested by Kuhn *et al.* [5], more than three-quarters of software faults are caused by the interaction between any two parameters, and all software faults can be detected by the interactions among six parameters. Therefore, based on combinatorial testing, only a small set of test cases can be used to greatly reduce costs while obtaining higher fault detection effectiveness, which has also been further confirmed by the research on some open-source software [6], [7].

The idea of combinatorial testing originated from the design of experiment (DOE). In 1985, Robert utilized orthogonal Latin squares to generate covering array and test computer hardware and software; thus, the combinatorial testing technique was proposed. With the dramatic development, combinational testing has become an effective software testing technique [40], [41], which has been successfully applied to many fields, such as input parameters, configuration, event-driven software, software product lines, concurrent programs, security, and mobile application [8]–[14]. The combinatorial testing technique has also been involved in the ISO/IEC/IEEE 29119 [15], and nearly 800 research papers related to combinatorial testing have been published, which can be divided into several categories, including modeling, performance optimization, fault diagnosis, etc., especially test case generation.

Specifically, to achieve the goal of generating test cases with greater coverage and smaller scale, three strategies have been proposed by researchers, i.e., greedy algorithms, algebraic methods, and evolutional algorithms. Particle swarm optimization (PSO), a widely used evolutional approach, has been extensively employed to generate test cases in the field of combinatorial testing due to its reliability and promise [42]–[44]. However, the effectiveness of PSO is significantly limited since it typically falls into local optimum, which reduces the overall performance of combinatorial testing.

It can be inferred that when using PSO to generate test cases in combinatorial testing, it is difficult to obtain the optimal solution due to premature convergence. Aiming at solving this problem, we propose a novel approach to improve the inertial weight of PSO, integrate PSO with the simulated annealing technique, as well as further develop a covering arrays generation approach that can generate covering arrays with arbitrary strength. Experimental results demonstrate that our strategy can substantially enhance the effectiveness of the original combinatorial testing technique.

The remainder of this article is organized as follows. Background and related works are given in Section II. We

Manuscript received September 24, 2021; revised November 3, 2021; accepted November 29, 2021. Date of publication December 16, 2021; date of current version March 2, 2022. Associate Editor: Ruizhi Gao. (*Corresponding author: Jinwei Shen.*)

Zhao Li, Kangjie Lu, and Jinwei Shen are with the College of Mathematics and Computer, Guangdong Ocean University, Zhanjiang 524091, China (e-mail: zhaoli@gdou.edu.cn; lu_kangjie@163.com; jwshengz@163.com).

Yuhang Chen is with the College of Computer and Information Technology, China Three Gorges University, Yichang 443002, China (e-mail: 1359172090@qq.com).

Yi Song is with the School of Computer Science, Wuhan University, Wuhan 430072, China (e-mail: yisong@whu.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TR.2021.3132147.

Digital Object Identifier 10.1109/TR.2021.3132147

introduce testing case generation based on inertia weight optimizations, the generation steps of covering array, as well as our novel PSO-SA strategy in Section III, followed by illustrating the experiments to compare the effectiveness of PSO-SA with the baselines in Section IV. Section V concludes this article.

II. Background and Related Work

A. Combinatorial Testing

In 1979, there was already an introduction to the application scenarios of combinatorial testing in the book "The art of software testing," which include decision table, equivalence class division, causality diagram, analysis of marginal value, etc. However, the testers found that the influence of the interactions between the parameters in the system cannot be detected effectively by using the above techniques, and new approaches need to be developed to deal with this problem.

The earliest approach in the field of combinatorial testing was proposed by Mandl in 1985 [16], which first applied the DOE to software testing, and proposed the issue of test space explosion, that is to say, testing a large number of parameters and values is very difficult. Since exhaustive testing cannot be achieved and random testing lacked theoretical basis [17], Mandl innovatively proposed the concept of combinatorial coverage between two parameters and used orthogonal Latin square to generate the test case set with effectiveness similar to exhaustive testing for the first time.

In 1985, Tatsumi *et al.* [18] conducted an in-depth exploration on combinatorial testing. He systematically explored how to find various parameters and values from the software system and generate test cases based on DOE. He specified that the number of values assigned to each parameter is generally not the same, and the combination coverage among various parameters only need to be covered once, which was called a combination table by Tatsumi *et al.* [18]. Furthermore, a systematic modeling technique of combinatorial testing was proposed.

In 1990, researchers at Bell Labs developed a combinatorial testing cases generation tool called AETG, which can provide online testing services. This work has pushed the research and application of combinatorial testing to a dramatic peak.

B. Test Case Generation

Test cases are the main basis of software testing and also a key factor of software quality assurance. The research focus of software testing is how to generate a set of test cases with greater coverage and smaller scale.

Combinatorial testing has made some application achievements in practice; however, there is still some work to be improved. Test cases generation is also the core issue of combinatorial testing research. To acquire smaller covering arrays, researchers have been exploring for decades to develop more significant techniques. The test cases generation of combinatorial testing is an NP-hard issue, and the type of test case set can be orthogonal arrays, k-dimensional covering arrays, incremental covering arrays, or covering arrays with variable strength. Currently, the widely used testing case generation techniques mainly include greedy algorithm, algebraic method, and evolutionary algorithm.

1) Greedy algorithm: It is specified that a local optimal strategy can produce the global optimal solution. Applying the greedy algorithm to test cases generation of combinatorial testing was first proposed by Cohe *et al.* [19]. They proposed a heuristic technique for generating test data based on pairwise covering, which generates test data according to the actual test requirements to cover the pairwise or multiple parameters of the system. In Bell Labs, this heuristic technique was integrated into a test data generation tool called AETG. Other well-known greedy algorithms are the TCG that integrates the parameter values sorting strategy [20], and the test data generation technique for covering pair combination based on parameters order [21].
2) Algebraic method: Kobayashi [22] and Williams [23], respectively, proposed two algebraic methods for generating test cases of pairwise combination, which are generally better than greedy algorithm and heuristic search algorithm. They construct a pairwise combination covering array based on some basic orthogonal arrays and feature blocks, the core idea of which is the overlap of basic components. However, there are some limitations in using orthogonal arrays to generate test cases. For instance, the size of test cases is affected by the number of parameters and candidate values, and the number of test cases may dramatically increase if the number of parameters and candidate values increases, making it impossible to conduct comprehensive testing.
3) Evolutionary algorithm: The use of evolutionary algorithms in test generation is another popular research area in recent years [24]. The application of genetic algorithms in covering arrays generation was explored by Ghazi *et al.* [25] in 2003, which demonstrated the feasibility of using genetic algorithms to generate covering arrays. In 2012, Nie *et al.* [26] proposed to use genetic algorithm to optimize parameters to generate covering array, which made a great progress. PSO is an effective heuristic search algorithm, which is also used for generating test cases in combinatorial testing [27]. In addition, ant colony algorithm [45], simulated annealing algorithm [46], and tabu search algorithm [47] are also employed to generate test cases in combinatorial testing.

C. Covering Array Generation Based on Particle Swarm Optimization

PSO is a relatively new evolutionary search technique. Inspired by the foraging behaviors of birds, researchers proposed to ask leaders in the group to guide the search of each particle. The good effectiveness and strong memory made PSO a popular technique in communication, power system, biomedical, and other fields [28].

Ahmed *et al.* [29] proposed an efficient strategy for variable-strength test suite generation based on PSO. In particular, they successively proposed planar particle swarm test generator (PPSTG), particle swarm test generator (PSTG), and Variable-Strength Particle Swarm Test Generator (VS-PSTG) techniques,

to respectively generate three types of covering arrays: two-dimensional, multidimensional, and variable strength, with the goal of ensuring optimal test size reduction. Chen *et al.* [30] developed a test case reduction strategy, which integrated PSO into pairwise testing (a special case of combinatorial testing aiming to cover all the pairwise combinations), to further reduce the size of the covering array and analyze the impact of various settings on the size of test cases. These explorations above have acquired good achievements based on which the influence of various parameters on the algorithm performance and the size of covering arrays are taken into account. Shi *et al.* [31] discussed the influence of inertial weight and maximum velocity on the effectiveness of PSO, and then some researchers made linear or nonlinear adjustments to related parameters, such as inertial weight or learning factor. In addition, Premalatha and Natarajan [49] presented a hybrid model of PSO and genetic algorithm to prevent the suboptimal solutions, and the experimental results showed that the proposed approach indeed outperformed the standard PSO. Furthermore, in order to search better solution and hunt the global optimum, Liu and Qiu [50] attempted to combine PSO with the Levenberg−Marquardt algorithm or the conjugate gradient algorithm, which is proven to be more effective than the adaptive PSO algorithm.

The techniques above have improved the effectiveness of these algorithms. However, it is always difficult to find the optimal parameter configuration, and unsuitable parameters may be selected at a certain step of the algorithm, resulting in a decrease in the effectiveness. To attack the above problem, researchers have successively proposed strategies for adaptive or periodic adjustment of parameters. For the issue that PSO is trapped in a local optimal when facing complex problems, a variety of other techniques are integrated into PSO, such as crossover operator, mutation operator, and particle regeneration, which effectively enhance the effectiveness of PSO [32].

III. Our Technique

PSO is well-suited to the covering array generation of combinatorial testing, and an inertia weight differential decrement strategy is employed to build the combinatorial testing covering arrays due to PSO's series of advantages, including only a few parameters needed, simple principles, and fast convergence speed.

A. Testing Case Generation Based on Inertia Weight Optimizations

An advanced PSO, which is based on the inertia weight linear differential decrement strategy, differs from the classic PSO in the following aspects.

1) Compared with the traditional fixed inertia weight strategy, the particles have a stronger global search capability in the initial state, and a much stronger local search capability in the later stage also prevents particles from falling into local optimal solution.
2) Compared with the classic inertia weight linear decrement strategy, the inertia weight is the *n*-degree polynomial function of time in advanced PSO; thus, it changes slowly in the initial stage but rapidly in the later stage, which is advantageous for finding a global optimal solution quickly.

Algorithm 1: Calculation of Linear Differential Decrement Strategy of Inertia Weight ω.

1) $\frac{d\omega}{dt} = \frac{ng(\omega_{\max}-\omega_{\min})}{t_{\max}^n} gt$
2) $\int_{\omega(t)}^{\omega_{\max}} d\omega = \frac{ng(\omega_{\max-}\ \omega_{\min})}{t_{\max}^n} \int_0^t \vartheta d\vartheta$
3) $\omega = \omega_{\max-}(\omega_{\max} - \omega_{\min}) \times \frac{t^n}{t_{\max}^n}$.

Because the particles in PSO are randomly dispersed in solution space in the initial state, using a large inertia weight in the initial stage is important to boost the global search capability. However, as iterations increase in the later stage, the particles become closer to the optimal solution; so the local search ability needs to be strengthened gradually. Eberhart *et al.* [33] proposed an inertia weight linear decrement strategy based on this hypothesis, illustrated in the following formula:

$$\omega = \omega_{\max} - \left(\omega_{\max} - \omega_{\min} \times \frac{t}{t_{\max}}\right) \tag{1}$$

where t represents the current number of iterations, $t_{\max}$ is the maximum number of iterations, and $\omega_{\max}$ and $\omega_{\min}$ represent the maximum and minimum values of the inertia weight, respectively.

Because the formula has a constant scope, the change of speed is always kept at the same level. This leads to an unexpected phenomenon in the PSO iteration process; that is, in the initial iteration, if the particle is in a poor position, PSO is likely to fall into local optimal solution with the iteration and the declining speed.

To avoid this phenomenon and prevent particles from falling into local optimal solution, this article puts forward an inertia weight linear differential reduction strategy based on the traditional inertia weight linear decrement strategy, which is shown in the following formula:

$$\omega = \omega_{\max} - \left(\omega_{\max} - \omega_{\min} \times \frac{t^n}{t_{\max}^n}\right) \tag{2}$$

where the definitions of t, $t_{\max}$, $\omega_{\max}$, and $\omega_{\min}$ are the same as those in formula (1), and n is the exponent.

Formula (2) is obtained through Algorithm 1.

The inertia weight is an *n*-degree polynomial function of the iteration times t in the above formula, which is also negatively correlated to the iteration times. Fig. 1 shows a comparison of the function changes (the value of n in Figs. 1 and 2). It is obvious that the initial iteration stage of advanced PSO has a slow interval of changes, which is conducive to the global search of particles. However, the later iteration stage has accelerated speed of reduction, which can achieve a quick convergence and approach to the optimal value. The function graphs of the advanced and classical PSO approaches are shown in Fig. 1.

The fitness function guides the iterative updating process of PSO and thus determines whether the PSO algorithm can find the optimal solution. Hence, it is important to set a reasonable fitness function. This article uses *the combination number that*

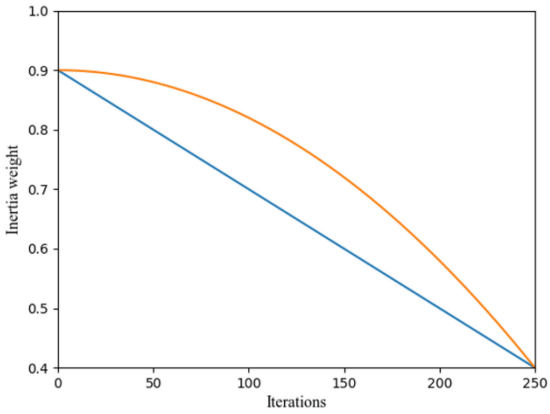

Fig. 1. Comparison of the two convergence strategies.

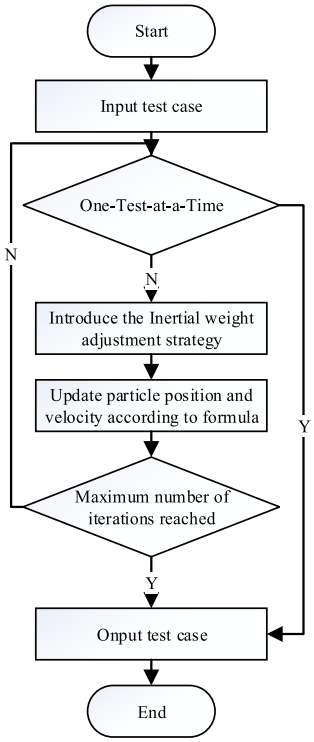

Fig. 2. Process of generating a single testing case.

can be covered by the current test case as the fitness function of PSO.

For example, software to be tested consists of four parameters, each of which has two valid values, namely, 0 and 1. The fitness function of the software when it is covered by two-way combinatorial testing is introduced as follows: suppose a testing case $T_1 = (1,1,0,0)$ has been generated, the two-way combination that can be covered is: (1, 1, _, _), (1, _, _, 0), (1, _, _, 0), (_, 1, _, 0), (_, _, 0, 0), (_, 1, 0, _), 6 in total, of which "_" indicates that the uncertainty of the value of the corresponding parameter. Six testing combinations can be covered by T_1; thus, the fitness value of T_1 will be 6 correspondingly.

B. Generation of Covering Array Based on Improved PSO

At the beginning of solving a problem, PSO first makes the rule that the particles are uniformly distributed in the whole solution space and makes the particles move at a certain speed in the solution space. Subsequently, each iteration causes each particle to constantly update the current state according to its own past optimal position, also known as personal best value (*pBest*), and the optimal position of the entire population, also known as global best value (*gBest*). By constantly searching in this way, each particle will gradually get closer to the personal optimal direction and the global optimal direction, so as to find the approximate optimal solution or the optimal solution of the problem.

Suppose that in a D-dimensional search space, the ith particle represents an efficient solution to the problem, and the velocity and position of the particle $p_{i,\,j}$ can be respectively represented by the vector set V_i and X_i, where $V_i = (v_{i,\,1}, v_{i,\,2}, \ldots, v_{i,\,d})$ and $X_i = (x_{i,\,1}, x_{i,\,2}, \ldots, x_{i,\,d})$. In each iteration, PSO uses the following two formulas to calculate the corresponding velocity and position of the particles:

$$v_{i,j}\ (k\ +1)\ = \omega \cdot\ v_{i,j}\,(k)\ + C_1 r_1\ [pBest_{i,j}\,(k) - x_{i,j}\,(k)] + C_2 r_2\,[gBest_j\,(k)\ -\ x_{i,j}\,(k)] \tag{3}$$

$$x_{i,j}\ (k\ +1) = x_{i,j}\ (k) + v_{i,j}\ (k\ +1)\,. \tag{4}$$

In the above formula, $pBest_{i,\,j}$ is the optimal position (personal optimal position) recorded in the history of particle $p_{i,\,j}$ under the J-dimensional solution space, and $gBest_j$ is the historical optimal position (global optimal position) of particles in the whole search space under the J-dimensional solution space. The velocity update process in formula (3) consists of three parts: the first part is ***inertia***, which represents the tendency of the particle to keep its own motion. ω is called the inertial weight. The second part is ***self-cognition***, which represents the learning and evolution ability of the particle based on its own historical optimal position. C_1 is the learning factor of the self-cognition of the particle, and r_1 is the random number of the interval [0,1]. The third part is ***collaboration***, which represents the information exchange among all the particles in the population. C_2 is the learning factor of the collaboration part, and r_2 is the random number of the interval [0,1]. Each particle in PSO can constantly move in the solution space under the control of these three parts; in others words, these parameters guide the whole population to transition between local search and global search.

Based on the above discussion, we can gain the steps of generating a single combinatorial testing case based on the inertia weight improvement strategy. The experimental parameters used in this article, including population size, iteration times, and learning factors, are set to commonly used values recommended by Wu [34]. Algorithm 2 provides the pseudo-code description of the test case generation process.

The process of generating a single testing case is shown in Fig. 2.

For example, given a coverage array, CA(2, 5, 3), in which the covering intensity is set to 2, the number of parameters is 5, and the number of options of each parameter is 3 (i.e., 0, 1, 2), we employ the mentioned Algorithm 2 to process it and a total of 12 test cases can be delivered: (0, 0, 1, 0, 1), (1, 2, 2, 1, 1),

Algorithm 2: Generation of a Test Case Using PSO, of Which the Inertial Weight is Optimized.

Input: params n, Corresponding values υ, Combination to be covered S.
Output: Optimal test cases *gBest*.

1. $it = 0$, $size = 80$, $iteration = 250$,
2. $C_1 = C_2 = 1.3$, $gBest =$ NULL
3. **for** each particle p_i **do**
4. Random initialization for position χ_i and velocity υ_i
5. **end for**
6. **while** $it <$ The preset maximum number of iterations **do**
7. **for** each particle p_i **do** //Adaptive value evaluation stage
8. Calculated adaptive value *fitness*(p_i)
9. **If**(*fitness*(p_i) = *C*(*n*, *t*))
10. **return** p_i
11. The particle with the maximum fitness value in the population was updated to *gBest*, and the historical optimal position of particle *i* was updated to $pBest_i$
12. **end for**
13. **for** each particle p_i **do** // Update location and speed
14. Introduce the inertial weight adjustment strategy
15. Use (3) and (4) to update particle velocity and position
16. Deal with maximum speed and boundary issues
17. **end for**
18. $it = it + 1$
19. **end while**
20. **return** *gBest*

Explanation of parameters.

T_0: Initial temperature.
T: The temperature in each iteration is generally $T = 0.9 \times T_0$.
x: Current solution.
x_1: Objective solution.
Δf: $f(x) - f(x_1)$.

(1, 1, 1, 2, 0), (0, 1, 0, 1, 2), (2, 0, 2, 2, 2), (2, 2, 0, 0, 0), (1, 0, 0, 1, 0), (2, 1, 1, 1, 1), (1, 2, 1, 0, 2), (0, 2, 2, 2, 0), (1, 1, 0, 2, 1), and (2, 1, 2, 0, 1). Similarly, if the covering intensity is set to 3, a total of 38 to 41 test cases[1] can be obtained by feeding CA(3, 5, 3) to Algorithm 2.

C. Simulated Annealing Strategy

The inertia weight adjustment strategy of PSO can effectively reduce the scale of the covering array while ensuring the running time of the algorithm. To further improve the performance of PSO combinatorial testing covering array generation, this article introduces a simulated annealing strategy to enhance the existing strategy. We will give the details concerning how to integrate a simulated annealing strategy into the existing PSO covering array generation in detail.

Simulated annealing (SA) is an algorithm inspired by thermodynamic behaviors [35] and simulates the physical process of metal cooling, the physical principle of which is that, for a metal that is at a high temperature and needs to currently cool down, the cooling process is also the process of reducing the internal energy in the object. In general, the greatest strategy is to quickly reduce the object's temperature. However, the rapid cooling rate makes establishing a stable crystalline state difficult for the object, and the object cannot minimize its internal energy. One potential idea is to enable slow cooling to make sure each particle in the metal has sufficient time to choose the best position, arrange closely and orderly and thus progressively build up a stable crystal state. Considering such a physics process and the mathematical algorithm comprehensively and applying the former to the issue of finding the optimal solution, the simulated annealing algorithm came into being. The simulated annealing algorithm starts from a higher initial temperature and, with the continuous decline of temperature parameters, randomly searches the global optimal solution of the objective function in the solution space based on the probability jump characteristics, that is, jumping out of the local optimal solution with a certain probability and finally approaching the global optimal solution.

Since simulated annealing can avoid local optimal solution problems by accepting a bad solution, it has achieved extensive success in solving complex optimization problems. A simulated annealing algorithm extends the local search algorithm and can converge to the global optimal solution under certain conditions. As a result, SA is regarded as a global optimal algorithm in theory.

The simulated annealing process starts with the initial feasible solution, sets an initial temperature T_0, and interferes with the algorithm's solution generation by controlling the change of the temperature *T*. Through the structure iteration, simulated annealing interferes with a candidate solution until the temperature *T* reaches a preset value lower than the stopping standard. In this process, a new candidate solution is selected from the neighborhood of the current solution through each iteration. Based on the comparison of these solutions, the best solution will be selected as the current new solution. In some scenarios, the algorithm can accept a bad solution with a certain probability to avoid local optimal value and continue to find a better solution.

The simulated annealing algorithm needs to set the following parameters.

For example, Fig. 3 shows a simple optimization process of simulated annealing. The red point is a local best point. When running to the red point, the algorithm will advance with a certain probability and accept the inferior solution until it finds the green point. At this time, the simulated annealing process ends and the temperature is also reduced to 0.

The simulated annealing algorithm determines whether to replace the optimal solution with the new solution generated by the selection according to the Metropolis criterion, as illustrated in the following formula:

$$P_i(n) \begin{cases} 1, E_i(n) \geq E_g \\ e^{\frac{E_i(n)-E_g}{T_i}}, E_i(n) < E_g \end{cases} \tag{5}$$

[1]These test cases are not listed here due to the limited space.

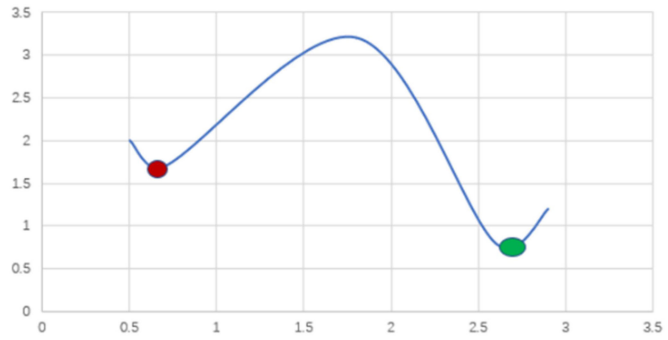

Fig. 3. Optimization of simulated annealing.

where E_i (n) represents the internal energy of the ith particle at the nth iteration, that is, the fitness value of the current particle i. E_g represents the most advantageous internal energy in the current population. T_i indicates the current temperature value, and each iteration process can result in the controllable linear decline of T_i to a certain extent. The optimization process of the simulated annealing method aims to achieve constantly slow cooling and search for new solutions. If the energy $E_i(n)$ in the current particle exceeds the most advantageous internal energy E_g of the current population, the probability of this particle entering the current position will be directly set to 1, that is, it will definitely enter. On the contrary, if the energy $E_i(n)$ in the current particle is less than the most advantageous internal energy E_g of the current population, the particle will enter the current position with a certain probability.

D. PSO-SA: A Novel Technique for Effectively Generating Covering Array

In the traditional PSO algorithms, the flying speeds v of particles are controlled in a certain range; thus, the particles are prevented from moving in a large deviation to avoid affecting the convergence speed and optimization results of the whole population. In the early stage of the algorithm execution, the convergence speed is high, and it might reach the extreme value quickly. However, after a period of execution, the speeds of all particles are near to 0, the capability of position updating becomes weak, and the convergence speed decreases or even stops. Hence, particles in PSO are prone to fall into local optimal. If a particle moves to a better position than the current position, which is also an optimal position for the global, the particle will send signals to the population. Consequently, the subsequent iteration process will be carried out based on this confusing location, and the optimization process may fall into the local optimal solution. This phenomenon is called "*premature convergence*."

PSO simulates the birds' behavior of hunting for food together. Although PSO does have a great advantage over other algorithms, there are still several limitations in its later stage, for instance, being prone to fall into local optimal solution, being prone to diverge, suffering from premature convergence, etc. These long-standing challenges remain the attention of researchers, and they find that the accuracy of PSO can be improved by modifying its parameters. However, for different optimization problems, various settings will lead to different results, and no optimal combination of parameters can be suitable for all cases. In other words, the best parameter selection in one optimization issue may not be the greatest choice in another issue.

To enable PSO to jump out of the local optimal position as much as possible, this article introduces a simulated annealing strategy so that particles can jump out of the local optimal solution by receiving the poor solution with a certain probability.

The simulated annealing algorithm is likely to obtain high-quality solutions at the cost of a slow convergence rate. In particular, when facing large-scale optimization challenges, the simulated annealing algorithm has extremely low efficiency. Therefore, it will take a lot of time to obtain the high-quality approximate optimal solution when using simulated annealing to solve the problem, which reduces its advantage significantly; thus, the simulated annealing algorithm fails to gain a wide practical application. To address this problem, the annealing rate can be adjusted appropriately to lower the running time of the algorithm.

There are two essential mechanisms for the simulated annealing algorithm: first, accepting the bad solution with a certain probability, and, second, simulating a cooling annealing process of high-temperature metals. These mechanisms allow SA to accept not only the solution that improves the fitness value of the objective function but also the solution that probably worsens the fitness value at a certain probability.

Many researchers and algorithm enthusiasts have made a series of improvements to PSO since its birth [48]. These attempts mainly seek to improve PSO speeds and the updating diversity of locations, maintain the evolutionary capability of the algorithm during the optimization process, thus avoid falling into the local optimum, and search the global optimal solution.

It is imperative to make the particles jump out of convergence and fly to other areas to search again when premature convergence occurs so that PSO can continue finding the global optimal solution. The aforementioned simulated annealing algorithm offers a solution to the challenge outlined above. SA algorithm takes effect before the particles accept the new solution, which enables the optimization process to not only accept a better solution but also accept a bad solution with a certain probability. Such capability can effectively alleviate the premature appearance of PSO.

To improve the performance of the whole optimization process, we propose a novel PSO algorithm based on simulated annealing strategy (PSO-SA), which integrates two algorithms through applying the simulated annealing algorithm's ability to accept bad solutions with a certain probability to PSO. The overview of the PSO-SA is shown in Fig. 4.

When the traditional PSO algorithm generates the covering array of combinatorial testing, the fitness function is defined as *the number of combinations that can be covered in the uncovered area*. This metric only considers the optimal test case in the current iteration process but ignores the relationship between the current generated test case and the existing test suite.

Table I provides two constructions of the covering array CA $(N; 2, 3^4)$. By calculating the fitness value, it can be found that the first three items of the two constructions can cover $C(4, 2) = 6$

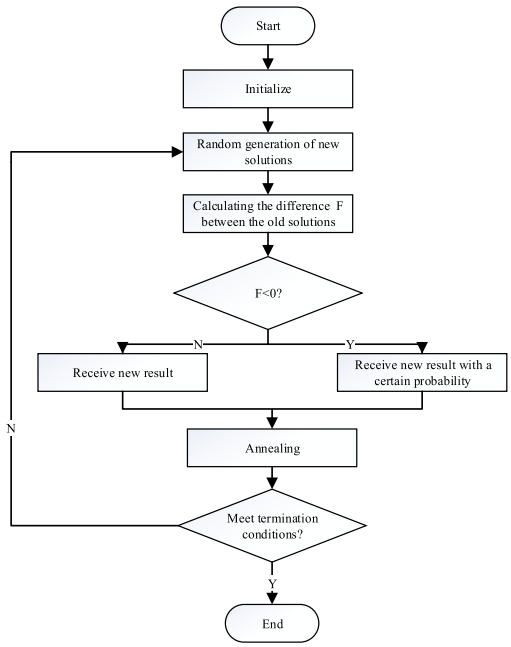

Fig. 4. Overview of PSO-SA strategy.

TABLE I
TWO CONSTRUCTIONS OF COVER TABLE CA (N; 2, 3^4) CONSTRUCTION 1

Construction 1				Construction 2			
0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1
0	2	2	2	2	2	2	2
1	0	1	2	-	-	-	-

different two-way combinations. But when generating the fourth testing case, the value between any two parameters in Method 2 has been covered. We randomly select a test case (1, 0, 1, 2). Because (1, _, 1, _) has been covered before, we fail to find a test case that can cover six different two-way combinations in the fourth case. In Construction 1, we can find (1, 0, 1, 2) to cover six combinations. The generation of combinatorial testing covering array aims to find the smallest covering array. Taking CA (N; 2, 3^4) as an example, its minimum covering array size is 9; so each testing case is required to cover six testing combinations. Construction 2 cannot get the minimum covering arrays, which is not expected to be seen in the combinatorial testing. Therefore, when taking into account the PSO final result *gBest*, we should also consider other factors in addition to the number of combinations that test cases can cover.

Given this problem, this article introduces a novel fitness value measurement strategy proposed by Wu *et al.* [34], to enable the test case to be more compact. The average Hamming distance is employed to measure the "tightness" between a test case t and an existing set *TS*. This article defines the Hamming distance $d_{1,2}$ between testing case t_1 and t_2 as *the number of parameters with different values between the two testing cases*, among which the number of test cases in the test suite *TS* is represented as $|TS|$. The average Hamming distance between t and *TS* is defined as follows:

$$H(t, TS) = \frac{1}{|TS|} \sum_{k \in TS} d_{tk}. \tag{6}$$

By improving the global optimal position, PSO selects the global optimal solution from the candidate solutions with the maximum coverage of the uncovered combinations (with the maximum fitness value), and its distance metric is the mean Hamming distance.

This section introduces a simulated annealing strategy to improve the PSO algorithm and proposes a covering array generation method based on this algorithm. The general logic framework of this approach consists of two components: the testing environment construction and the actual testing, as shown in Fig. 5.

The testing environment construction component leads the whole process of generating the combinatorial testing covering arrays. This component models the system under test (SUT) and converts the influencing factors of the actual testing system into mathematical parameters. Then, the static logic analysis is carried out on the SUT to clarify the covering intensity and the system constraints. Finally, a set S of combinations to be covered can be obtained. For example, through the construction of the testing environment, the covering is conducted on the combination CA (2, 10, 10), where the covering intensity value is 2, the number of parameters is 10, and each parameter's option is also 10.

The actual testing component is the crux of the combinatorial testing method. Based on the previous testing environment construction, the final covering array is generated by combining the PSO-SA algorithm with the testing case evolution strategy (one test at a time). Specifically, the particle position and velocity are first initialized, and a test case is generated by the PSO-SA. Then, the "one test at a time strategy" is adopted to remove the combinations that can be covered by the test case from *S*. Finally, the preceding steps are repeated until all combinations in S are covered or the maximum number of iterations is reached.

Generating a test case based on the PSO-SA includes the following steps:

Step 1: Initialize the position of all particles.
Step 2: Set the initial temperature $t_0 = f_{\max} - f_{\min}$, where $f_{\max}$ and $f_{\min}$ are the maximum and minimum fitness of the initial population respectively.
Step 3: Set *pBest* as the current position and *gBest* as the optimal particle position in the particle swarm to evaluate the particle fitness value.
Step 4: Update the particle position and speed according to the particle swarm formula.
Step 5: Generate a new position randomly for the particle and calculate the difference VC of the adaptability between the old and the new positions.
Step 6: If VC < 0, the particle enters the new position; if VC $=$ 0, conduct the global optimal position judgment. Otherwise, execute the next step.

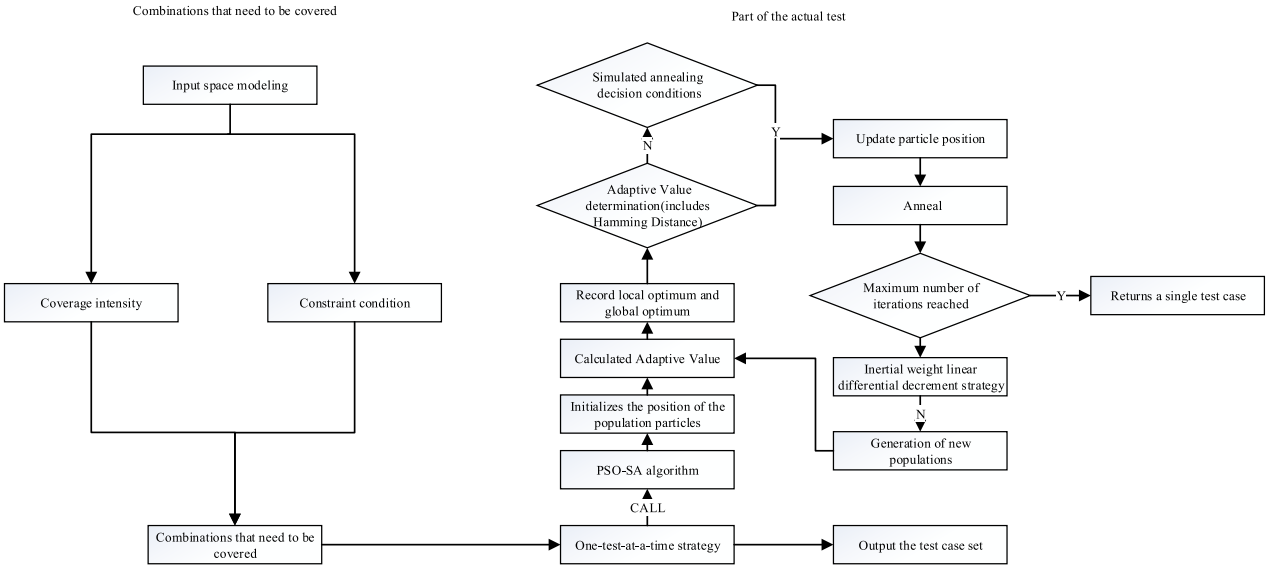

Fig. 5. Covering array generation framework.

Step 7: Generate the random number r between (0,1), if $r < \min$ [1, exp $(-\mathrm{VC}/t)$], the particle enters the new position, and then perform Step 4 again.

Step 8: Annealing operation: $t = 0.9 \times t$.

Step 9: Determine whether the optimization process reaches the maximum iteration times; if it does, go to Step 10; otherwise, go back to Step 3.

Step 10: Output the complete testing case.

IV. Case Studies

A. Effectiveness Evaluation of W-PSO

To verify the effectiveness of the PSO based on the inertia weight linear differential decrement strategy (W-PSO) to generate combinatorial testing cases, we randomly select 10 representative covering arrays in previous researches for analysis. The covering arrays are shown in Table I and include the covering array of any strength. Considering the randomness of PSO, we repeat each algorithm 20 times in each covering array and then report the average and minimum of the scale of the generated covering arrays.

We choose four PSO algorithms, PSO, PSO-1, adaptive particle swarm optimization (APSO), discrete particle swam optimisation (DPSO), as baselines to compare them with the W-PSO algorithm. The PSO algorithm adopts fixed inertia weight; PSO-1 adopts inertia weight linear decrement strategy; APSO algorithm [36] uses self-adaptive adjustment strategy proposed by Zhan for learning factors; DPSO algorithm [33] is a discrete PSO algorithm proposed by Wu.

The W-PSO (the square of iteration times) algorithm is compared with PSO (with the fixed inertia weight as 0.7), PSO-1 (inertia weight linear decrement strategy), APSO algorithm, and DPSO algorithm. This article analyzes the minimum and average scale of covering arrays generated by these five algorithms and verifies whether the W-PSO algorithm has better optimization effectiveness on the covering array generation for combinatorial testing. Table II shows the minimum and average scale of the first four algorithms to generate the covering array, as well as the minimum size of the DPSO algorithm to generate the covering array. The number in brackets represents the time (seconds) that the algorithm spends.

TABLE II
Ten Selected Covering Arrays $\mathrm{CA}_1(2, 6, 3)$

$CA_1(2, 6, 3)$	$CA_2(2, 10, 5)$
$CA_3(2, 20, 3)$	$CA_4(2, 10, 10)$
$CA_5(2, 9, 4^7 3^2)$	$CA_6(2, 9, 10^1 9^1 8^1 7^1 6^1 5^1 4^1 3^1 2^1)$
$CA_7(4, 6, 2^3 3^2 4^1)$	$CA_8(3, 10, 3^4 4^6)$
$CA_9(3, 8, 5^2 4^2 3^4)$	$CA_{10}(5, 7, 2^4 3^2 4^1)$

Table II shows the minimum and average scale of covering arrays generated by the five algorithms. Black font indicates the significant difference between the five algorithms in generating the smallest covering array. Based on the performance of these five algorithms, we can find that the W-PSO algorithm generates a smaller covering array compared with PSO, PSO-1, and APSO algorithm in 50% situations (that is, CA_4, CA_5, CA_7, CA_8, and CA_{10}) and has no worse than the existing results in 90% situations. Although the DPSO algorithm generates a smaller covering array scale than the W-PSO algorithm under large covering requirements, it takes almost 10 times as much time as the W-PSO algorithm, thus increasing the testing cost significantly. The W-PSO algorithm generates similar covering arrays to those generated by the DPSO algorithm in a very short time. Besides, W-PSO is more likely to generate smaller covering arrays when the covering requirement is greater than 4; as a result, it can be concluded that under the high covering requirements, the performance of W-PSO performs better than the traditional PSO and the PSO based on the inertia weight linear decrement strategy.

In addition, in the aspect of the average scale of generating covering arrays, W-PSO algorithm performs slightly better than PSO algorithm and PSO-1 algorithm in 80% (8/10) situations

TABLE III
COMPARISON OF COVERING ARRAYS

Coverage matrix	PSO		PSO-1		W-PSO		APSO		DPSO
	min	*mean*	*min*	*mean*	*min*	*mean*	*min*	*mean*	*min*
CA_1	14 (<1)	15.1	14 (<1)	14.7	14 (<1)	14.5	14 (<1)	14.8	14 (2)
CA_2	44(2)	44.3	43 (2)	44.2	43 (2)	44.7	45 (7)	45.3	43 (17)
CA_3	21	22.17	21 (3)	22	21 (3)	22.3	22 (4)	22.7	21 (27)
CA_4	158 (6)	158.6	158 (6)	159.7	**157 (6)**	158.6	175 (13)	177.5	**147 (65)**
CA_5	26 (<1)	27.3	26 (1)	27	**25 (1)**	26.7	27 (3)	27.6	26 (8)
CA_6	95(3)	99.2	**93 (3)**	96.7	94 (3)	97.4	96 (12)	100.5	93 (34)
CA_7	92(2)	94.7	92 (2)	93	**89 (3)**	92.7	91 (3)	94.3	81 (22)
CA_8	120 (16)	121.6	119 (17)	120.8	**117 (15)**	120.4	126 (31)	127.2	117 (147)
CA_9	131 (8)	132.2	131 (7)	133.3	131 (7)	134.5	131 (9)	135	133 (81)
CA_{10}	205 (8)	209.6	210 (9)	211	**204 (9)**	208	212 (11)	212.7	209 (70)

and better than APSO algorithm in 100% (10/10) situations; W-PSO algorithm is significantly better than APSO algorithm in 60% (6/10) situations of covering array generation.

B. Effectiveness Evaluation of PSO-SA

This article develops a series of optimizations and improvement strategies for the PSO algorithm of the generation of combinatorial testing covering arrays, including optimizing the inertia weight value strategy, introducing a simulated annealing strategy, and optimizing the same fitness value through the Hamming distance. To evaluate the effectiveness of the proposed strategy, we design the following experiments to compare our proposed strategy with many other counterparts.

Experiments in Section IV-B have demonstrated the optimized inertia weight linear differential reduction strategy's advantages compared with baselines. This section mainly researches the effectiveness of other approaches on the generation of combinatorial testing covering arrays. We will focus on two research questions (RQs):

RQ1: Does the proposed PSO-SA algorithm generate a smaller covering array?

RQ2: Whether the optimized global optimal position evaluation strategy optimizes the algorithm?

To explore these two RQs, in addition to the APSO algorithm in Section IV, PSO-CL [37] and PSO-DMS [38] algorithms are also implemented to generate covering arrays and compared with the proposed PSO-SA algorithm. The experiments are conducted on Windows 10 operating system, with the i7-6700k processor and 16-GB memory.

The main objective of generating combinatorial test cases is to build a test suite with strong covering capability, i.e., covering arrays. Thus, the primary challenge is how to generate smaller covering arrays without loss of covering capability. To compare the performance of the selected algorithms with our proposed PSO-SA, this article carries out extensive experiments in terms of the scale of generated covering arrays. This section complements six additional classical covering matrices with variable strength [34], as shown in Table IV. The scale of classical covering matrices with variable strength is the addition of covering arrays' scale generated under different covering intensities. To avoid the influence of random factors on the experimental results during execution, each covering matrix is executed 20 times independently, and the minimum and average sizes of the generated covering arrays are collected.

TABLE IV
SIX COVERING MATRICES WITH VARIABLE STRENGTH
VCA_{11} (2, 3^{15}, CA (3, 3^5))

VCA_{11} (2, 3^{15}, $CA(3, 3^5)$)
VCA_{12} (2, 3^{15}, $CA(3, 3^6)$)
VCA_{13} (2, 3^{15}, $CA(3, 3^7)$)
VCA_{14} (2, 3^{15}, $CA(4, 3^5)$)
VCA_{15} (3, 3^{15}, $CA(4, 3^7)$)
VCA_{16} (2, $4^4 5^2 6^2$, $CA(3, 4^3)$)

The experimental design of this section adds two groups of comparative experiments, namely, PSO-CL and PSO-DMS. Table V shows the experimental results of the minimum scale and time consumption of the covering array generated.

Table V shows the minimum scale of PSO algorithm, W-PSO algorithm, APSO algorithm, DPSO algorithm, PSO-CL algorithm, PSO-DMS algorithm, and PSO-SA algorithm in the 20 generations of combinatorial testing covering arrays, where the numbers in parentheses indicate the running time (in seconds). Overall, it can be seen that the PSO-SA algorithm performs better than other algorithms.

The covering arrays generated by PSO-SA are the minimum of seven algorithms in 56% (9/16) situations, and it is no worse than the other six algorithms in 87.5% (14/16) situations. For example, in terms of the comparison between PSO-SA and PSO, the PSO-SA algorithm is better than the PSO algorithm in 81% (13/16) situations and is not inferior to the PSO algorithm in 94% (15/16) situations. As for the comparison between PSO-SA and APSO, the PSO-SA algorithm is better than the APSO algorithm in 87.5% (14/16) situations and is 100% no better than the APSO algorithm. As for the comparison between PSO-SA and DPSO, the PSO-SA algorithm is superior to the DPSO algorithm in 62.5% (10/16) situations and is not inferior to the APSO algorithm in 81% (13/16) situations. In terms of the comparison between PSO-SA and PSO-CL, the PSO-SA algorithm is better than the PSO-CL algorithm in 81% (13/16) situations and is no better than the PSO-CL algorithm in 93.75% (15/16) situations. As for the comparison between PSO-SA and PSO-DMS, the PSO-SA algorithm is better than the PSO-DMS algorithm in 81% (13/16) situations and is no better than the APSO algorithm in 93.75% (15/16) situations.

TABLE V
MINIMUM SCALE COVERING ARRAY THAT CAN BE GENERATED BY EACH ALGORITHM

Coverage matrix	PSO	W-PSO	APSO	DPSO	PSO-CL	PSO-DMS	PSO-SA
CA_1	14 (<1)	14 (<1)	14 (<1)	14 (2)	14 (1)	14 (1)	14 (2)
CA_2	44 (2)	43 (2)	45 (7)	43 (17)	45 (8)	43 (3)	42 (18)
CA_3	21	21 (3)	22 (4)	21 (27)	22 (11)	21 (2)	21 (26)
CA_4	158 (6)	157 (6)	175 (13)	147 (65)	172 (16)	169 (5)	145 (61)
CA_5	26 (<1)	25 (1)	27 (3)	26 (8)	26 (2)	27 (1)	25 (8)
CA_6	95 (3)	94 (3)	96 (12)	93 (34)	98 (8)	96 (3)	91 (42)
CA_7	92 (2)	89 (3)	91 (3)	81 (22)	92 (5)	90 (3)	82 (29)
CA_8	120 (16)	117 (15)	126 (31)	117 (147)	118 (40)	118 (15)	115 (142)
CA_9	131 (8)	131 (7)	131 (9)	133 (81)	133 (20)	131 (9)	128 (79)
CA_{10}	205 (8)	204 (9)	212 (11)	209 (70)	208 (21)	208 (9)	199 (80)
VCA_{11}	57 (3)	56 (3)	60 (3)	62(65)	59 (7)	58 (3)	55 (63)
VCA_{12}	61 (3)	61 (3)	65 (4)	64 (70)	64 (8)	62 (4)	65 (67)
VCA_{13}	68 (3)	67 (3)	70 (4)	68 (77)	69 (8)	68 (5)	67 (65)
VCA_{14}	168 (30)	164 (29)	177 (33)	154 (267)	176 (120)	172 (30)	151 (251)
VCA_{15}	230 (36)	227 (34)	239 (41)	224 (328)	235 (135)	230 (36)	219 (330)
VCA_{16}	105 (1)	103 (1)	107 (1)	103 (11)	104 (1)	105 (1)	104 (12)

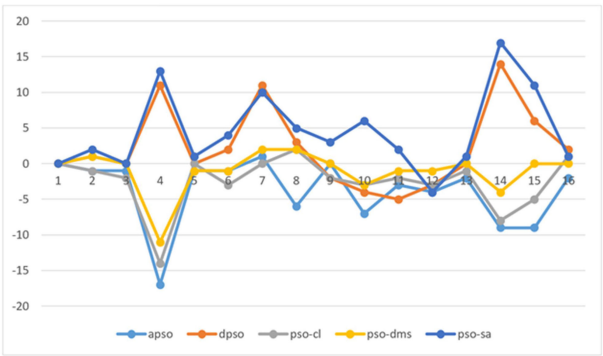

Fig. 6. Comparison among each algorithm.

From the viewpoint of time-consumption, the PSO algorithm and W-PSO algorithm cost the least time, and the PSO-SA algorithm consumes much more time. The reason is that the improved global optimal position evaluation and simulated annealing process need to consume more computing. Thus, this article can reasonably recommend the algorithm: PSO-SA algorithm can be used in the case of the minimum testing case set; the W-PSO algorithm can be used in the case of the need to get the better testing case set in a short time.

To demonstrate the PSO-SA algorithm's capability to generate combinatorial testing covering arrays more intuitively, we also compare PSO-SA with baselines through a line graph in Fig. 6, where the ordinate represents the difference between the minimum size of the covering array generated by each baseline algorithm on various covering matrices and that of the PSO algorithm.

Fig. 6 shows the capabilities of five algorithms to generate the minimum covering arrays. The comparison results of various algorithms show that the PSO-SA algorithm is better than other algorithms in covering matrix CA_4, CA_8, CA_{10}, VCA_{14}, and VCA_{15}, but it is inferior to the PSO algorithm in the covering matrix VCA_{12}. It indicates that the proposed PSO-SA algorithm can generate a smaller covering array in most cases. In other words, the limitations and challenges faced by traditional PSO techniques as well as the improvement brought by our novel approach are clearly illustrated in Fig. 6, and the contrasts further confirm the effectiveness of PSO-SA.

C. Threats to Validity

Despite the fact that promising results have been obtained by using our novel technique, there are still some factors that can influence our experiments in Section IV. For example, to generate combinatorial testing cases, we select ten representative covering arrays in previous researches for analysis, and the randomness could lower the generality of our evaluation. Besides, more projects (i.e., arrays) could be included in our datasets for the expansion of our experiments domain.

V. CONCLUSION

PSO has been intensively studied by researchers for its advantages of easy implementation and fast convergence in test case generations of combinatorial testing. However, PSO tends to fall into local optimization. To overcome this challenge, this article proposed PSO-SA, a novel PSO algorithm based on the SA strategy, to generate covering arrays of any covering intensity.

The article optimized the inertia weight of PSO, proposed and applied the inertia weight differential decrement strategy to the generation of combinatorial testing covering arrays, compared it with four PSO algorithms, and proved that the proposed strategy can generate a smaller covering array.

We applied the PSO-SA algorithm to the generation of combinatorial testing covering arrays. Through the multiple comparative experiments, we proved that this strategy can reduce the covering array scale effectively.

In future, we will continue enhancing combinatorial testing from many aspects, including further improving the capability of PSO to jump out of locally optimal solutions, reducing the scale of covering arrays, and better understanding the application scenarios of combinatorial testing.

REFERENCES

[1] N. Peng, G. Ji, and Z. Qin, "Survey on automatic test case generation algorithms for software testing," *Appl. Res. Comput.*, vol. 29, no. 2, pp. 401–405, 2012.

[2] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Comput. Surv.*, vol. 43, no. 2, pp. 11:1–11:29, 2011.

[3] D. Li, L. Hu, R. Gao, W. E. Wong, D. R. Kuhn, and R. N. Kacker, "Improving MC/DC and fault detection strength using combinatorial testing," in *Proc. Companion 17th IEEE Int. Conf. Software Quality, Reliab. Security*, Prague, Czech Republic, 2017, pp. 297–303.

[4] Z. Wang, Y. Zhang, P. Gao, and S. Shuang, "Comparing fault detection efficiencies of adaptive random testing and greedy combinatorial testing for Boolean-specifications," *Int. J. Performability Eng.*, vol. 17, no. 1, pp. 114–122, 2021.

[5] D. R. Kuhn and M. J. Reilly, "An investigation of the applicability of design of experiments to software testing," in *Proc. 27th Annu. NASA Goddard/IEEE Softw. Eng. Workshop*, Greenbelt, MD, USA, 2002, pp. 91–95.

[6] F. Medeiros *et al.*, "A comparison of 10 sampling algorithms for configurable systems," in *Proc. 38th Int. Conf. Softw. Eng.*, Austin, TX, USA, 2016, pp. 643–654.

[7] A. B. Sanchez *et al.*, "Variability testing in the wild: The Drupal case study," *Softw. Syst. Model.*, vol. 16, no. 1, pp. 173–194, 2017.

[8] B. Garn and D. E. Simos, "Eris: A tool for combinatorial testing of the Linux system call interface," in *Proc. IEEE 7th Int. Conf. Softw. Testing, Verification Validation Workshops*, Cleveland, OH, USA, 2014, pp. 58–67.

[9] J. Petke, "Testing django configurations using combinatorial interaction testing," in *Proc. Int. Symp. Search Based Softw. Eng.*, Bergamo, Italy, 2015, pp. 242–247.

[10] X. Yuan, M. B. Cohen, and A. M. Memon, "GUI interaction testing: Incorporating event context," *IEEE Trans. Softw. Eng.*, vol. 37, no. 4, pp. 559–574, Jul./Aug. 2011.

[11] R. E. Lopez-Herrejon *et al.*, "A first systematic mapping study on combinatorial interaction testing for software product lines," in *Proc. IEEE 8th Int. Conf. Softw. Testing, Verification Validation Workshops*, Graz, Austria, 2015, pp. 1–10.

[12] L. Yu *et al.*, "A combinatorial testing strategy for concurrent programs," *Softw. Testing Verification Rel.*, vol. 17, no. 4, pp. 207–225, 2007.

[13] N. Mirzaei *et al.*, "Reducing combinatorics in GUI testing of android applications," in *Proc. IEEE/ACM 38th Int. Conf. Softw. Eng.*, Austin, TX, USA, 2016, pp. 559–570.

[14] D. E. Simos *et al.*, "Combinatorial methods in security testing," *Computer*, vol. 49, no. 10, pp. 80–83, 2016.

[15] For Standardization I O.ISO/IEC/IEEE 29119 Software Testing Standard [EB/OL]. [Online]. Available: http://www.softwaretestingstandard.org/

[16] M. Robert, "Orthogonal Latin squares: An application of experiment design to compiler testing," *Commun. ACM*, vol. 28, no. 10, pp. 1054–1058, 1985.

[17] P. M. Bueno, W. E. Wong, and M. Jino, "Improving random test sets using the diversity oriented test data generation," in *Proc. 2nd Int. Workshop Random Testing Co-located 22nd IEEE/ACM Int. Conf. Automated Softw. Eng.*, Atlanta, GO, USA, 2007, pp. 10–17.

[18] K. Tatsumi, "Test case design support system," in *Proc. Int. Conf. Qual. Control*, Tokyo, Japan, 1987, pp. 615–620.

[19] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG system: An approach to testing based on combinatorial design," *Proc. IEEE Trans. Softw. Eng.*, vol. 23, no. 7, pp. 437–444, Jul. 1997.

[20] Y.-W. Tung and W. Aldiwan, "Automating test case generation for the new generation mission software system," in *Proc. IEEE Aerosp. Conf. Proc.*, vol. 1, Big Sky, MT, USA, 2000, pp. 431–437.

[21] K. C. Tai and Y. Lei, "A test generation strategy for pairwise testing," *IEEE Trans. Softw. Eng.*, vol. 28, no. 1, pp. 109–111, Jan. 2002.

[22] N. Kobayashi, T. Tsuchiya, and T. Kikuno, "A new method for constructing pair-wise covering designs for software testing," *Inf. Process. Lett.*, vol. 81, no. 2, pp. 85–91, 2002.

[23] A. W. Williams, *Software Component Interaction Testing: Coverage Measurement and Generation of Configurations*. Ottawa, ON, Canada: Univ. Ottawa, 2002.

[24] R. Gao, L. Hu, W. E. Wong, H.-L. Lu, and S.-K. Huang, "Effective test generation for combinatorial decision coverage," in *Proc. Companion 16th IEEE Int. Conf. Softw. Qual., Rel. Secur.*, Vienna, Austria, 2016, pp. 47–54.

[25] S. A. Ghazi and M. A. Ahmed, "Pair-wise test coverage using genetic algorithms," in *Proc. Congr. Evol. Comput.*, Canberra, ACT, Australia, 2003, pp. 1420–1424.

[26] Y. Liang and C. Nie, "The optimization of configurable genetic algorithm for covering arrays generation," *Chin. J. Comput.*, vol. 35, no. 7, pp. 1522–1538, 2012.

[27] C. Nie, H. Wu, Y. Liang, H. Leung, F. Kuo, and Z. Li, "Search based combinatorial testing," in *Proc. 19th Asia-Pacific Softw. Eng. Conf.*, Hong Kong, China, 2012, pp. 778–783.

[28] K. Li, L. Jia, and X. Shi, "IPSOMC: An improved particle swarm optimization and membrane computing based algorithm for cloud computing," *Int. J. Performability Eng.*, vol. 17, no. 1, pp. 135–142, 2021.

[29] B. S. Ahmed and K. Z. Zamli, "A variable strength interaction test suites generation strategy using particle swarm optimization," *J. Syst. Softw.*, vol. 84, no. 12, pp. 2171–2185, 2011.

[30] X. Chen, J. Qi, D. Chen, and Q. Gu, "Applying particle swarm optimization to pairwise testing," in *Proc. IEEE 37th Annu. Comput. Softw. Appl. Conf.*, Seoul, South Korea, 2010, pp. 107–116.

[31] J. Kari and Nurmela, "Upper bounds for covering arrays by Tabu search," *Discrete Appl. Math.*, vol. 138, no. 1-2, pp. 143–152, Mar. 2004.

[32] W. Huayao and N. Changhai, "Particle swarm optimization for covering array generation: Parameter optimization and adaptive algorithm," *J. Chinese Computer Syst.*, vol. 33, no. 10, pp. 2259–2267, 2012.

[33] R. C. Eberhart and Y. Shi, "Comparing inertial weights and constriction factor in particle swarm optimization," in *Proc. 2000 Congr. Evol. Comput.*, vol. 1, La Jolla, CA, USA, 2000, pp. 84–88.

[34] H. Wu, "Research on combinatorial testing and its fault detection ability," Ph.D. thesis, Nanjing University, 2018.

[35] H. Wu *et al.*, "A discrete particle swarm optimization for covering array generation," *IEEE Trans. Evol. Comput.*, vol. 19, no. 4, pp. 575–591, Aug. 2015.

[36] Z. H. Zhan *et al.*, "Adaptive particle swarm optimization," *IEEE Trans. Syst., Man, Cybern., Part B: Cybern.*, vol. 39, no. 6, pp. 1362–1381, Dec. 2009.

[37] J. Liang *et al.*, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 281–295, Jun. 2006.

[38] J. J. Liang and P. N. Suganthan, "Dynamic multi-swarm particle swarm optimizer," in *Proc. 2005 IEEE Swarm Intell. Symp., SIS 2005.*, Pasadena, Cal, USA, 2005, pp. 124–129.

[39] L. S. Ghandehari *et al.*, "A combinatorial testing-based approach to fault localization," *IEEE Trans. Softw. Eng.*, vol. 46, no. 6, pp. 616–645, Jun. 2020.

[40] H. Wu *et al.*, "An empirical comparison of combinatorial testing, random testing and adaptive random testing," *IEEE Trans. Softw. Eng.*, vol. 46, no. 3, pp. 302–320, Mar. 2020.

[41] R. Tzoref-Brill, "Advances in combinatorial testing," *Adv. Comput.*, vol. 112, pp. 79–134, 2019.

[42] S. Sengupta, S. Basak, and R. A. Peters, "Particle swarm optimization: A survey of historical and recent developments with hybridization perspectives," *Mach. Learn. Knowl. Extraction*, vol. 1, no. 1, pp. 157–191, 2019.

[43] J. C. Bansal, "Particle swarm optimization," in *Evolutionary and Swarm Intelligence Algorithms*. Cham, Switzerland: Springer, 2019, pp. 11–23.

[44] G. Xu *et al.*, "Particle swarm optimization based on dimensional learning strategy," *Swarm Evol. Comput.*, vol. 45, pp. 33–51, 2019.

[45] Q. Luo *et al.*, "Research on path planning of mobile robot based on improved ant colony algorithm," *Neural Comput. Appl.*, vol. 32, no. 6, pp. 1555–1566, 2020.

[46] N. Leite, F. Melício, and A. C. Rosa, "A fast simulated annealing algorithm for the examination timetabling problem," *Expert Syst. Appl.*, vol. 122, pp. 137–151, 2019.

[47] N. Hou *et al.*, "An efficient GPU-based parallel tabu search algorithm for hardware/software co-design," *Front. Comput. Sci.*, vol. 14, no. 5, pp. 1–18, 2020.

[48] D. Tian and Z. Shi, "MPSO: Modified particle swarm optimization and its applications," *Swarm Evol. Comput.*, vol. 41, pp. 49–68, 2018.

[49] K. Premalatha and A. M. Natarajan, "Hybrid PSO and GA for global maximization," *Int. J. Open Problems Comput. Math*, vol. 2, no. 4, pp. 597–608, 2009.

[50] J. Liu and X. Qiu, "A novel hybrid PSO-BP algorithm for neural network training," in *Proc. Int. Joint Conf. Comput. Sci. Optim.*, vol. 1, Sanya, China, 2009, pp. 300–303.