SCJ SOFT COMPUTING JOURNAL
ISSN: 2322-3707
University of Kashan

Beautiful Mind: a meta-heuristic algorithm for generating minimal covering array

Sajad Esfandyari[1,*], Leila Yousofvand[2], Vahid Rafe[2] and Einollah Pira[4*]

[1]Department of Computer Engineering, Faculty of Engineering, Malayer University, Malayer, Iran.
[2]Department of Computer Engineering, Faculty of Engineering, Lorestan University, Korramabad, Iran.
[3]Department of Computer Engineering, Faculty of Engineering, City, University of London, UK.
[4]Faculty of Information Technology and Computer Engineering, Azarbaijan Shahid Madani University, Tabriz, Iran.

Abstract. Combinatorial Testing (CT) is a widely used strategy in software testing that ensures coverage of parameter interactions using a minimal number of test cases, typically represented as Covering Arrays (CAs). A key challenge in generating effective CAs lies in the tendency of many search-based algorithms—especially meta-heuristic ones—to become trapped in local optima, leading to suboptimal solutions and high computational costs. Although various algorithms have been proposed to mitigate this issue, many still suffer from inefficiency and lack of consistency in producing high-quality test sets.

In this study, we introduce a novel meta-heuristic algorithm called **Beautiful Mind (BM)**, inspired by the human cognitive process. The algorithm combines elements of both rational intelligence and emotional reasoning to guide the search process more adaptively and escape local optima. BM incorporates a dynamic adjustment mechanism to improve search diversity and convergence. Comprehensive experiments were conducted to evaluate its performance, showing that BM not only produces more optimal covering arrays but also achieves greater consistency and accuracy compared to existing meta-heuristic methods.

Keywords: Beautiful Mind (BM), Combinatorial Testing (CT), Covering Array (CA), Meta-heuristic Optimization, Software Testing

1. Introduction

Software systems are becoming increasingly complex, comprising numerous interacting components, which makes software testing a crucial yet challenging and costly process [1]. One of the fundamental challenges in test case generation is detecting errors efficiently while minimizing computational costs. Combinatorial Testing (CT) is a well-established approach for identifying faults caused by interactions between different system components [2, 3].

In CT, interactions between software components form a vast search space, from which a Covering Array (CA) is constructed by selecting a minimal set of test cases

*Corresponding author. E-mails: Esfandyari@malayeru.ac.ir; Yousofvand.l@lu.ac.ir; v-rafe@araku.ac.ir; pira@azaruniv.ac.ir

that sufficiently cover all required parameter combinations [4]. The primary objective in CA generation is to minimize the number of test cases while ensuring full coverage of a given interaction strength (t-way testing). A well-constructed CA should be both efficient (i.e., minimal in size) and computationally feasible (i.e., generated within a reasonable time) [5]. Numerous approaches have been developed to optimize CA generation, which can be broadly categorized into mathematical and computational methods. Due to the limitations of mathematical approaches, most modern research focuses on computational techniques [6, 7].

Computational methods for CA generation fall into two primary categories [8]: greedy algorithms and meta-heuristic algorithms. While greedy algorithms provide deterministic and often fast solutions, they tend to be less efficient than meta-heuristic methods. Meta-heuristic algorithms, on the other hand, offer more flexible and adaptable search mechanisms, enabling them to find near-optimal solutions. However, these algorithms often suffer from premature convergence, where they get trapped in local optima, or face performance degradation over successive iterations. Several solutions have been proposed to address these challenges, achieving varying degrees of success. Nevertheless, improving efficiency often comes at the cost of reduced performance. Therefore, there remains a need to develop a novel meta-heuristic algorithm that effectively balances efficiency and performance in CA generation [9].

Meta-heuristic algorithms typically follow an iterative process in which a population of potential solutions is generated and refined using specific operators [10]. The effectiveness of these algorithms depends on their search strategies, selection mechanisms, and adaptation to different problem domains. No single meta-heuristic algorithm universally outperforms all others; instead, different algorithms excel in different scenarios. In the context of t-way testing, selecting an appropriate meta-heuristic algorithm is critical to ensuring an optimal CA with minimal test cases while maintaining high coverage. Given the limitations of existing methods, there is a clear motivation to explore new meta-heuristic approaches tailored to the challenges of CA generation [11, 12].

In this study, we propose a novel meta-heuristic algorithm, Beautiful Mind (BM), inspired by human problem-solving behavior, particularly the cognitive processes of professors and students. The BM algorithm integrates intelligence and adaptive learning mechanisms to enhance search efficiency and overcome local optima. By leveraging an innovative problem-solving model, BM aims to generate optimal CAs more effectively than existing meta-heuristic approaches.

The structure of this paper is as follows: Section II introduces Combinatorial Testing (CT) and highlights the limitations of existing CA generation techniques. Section III describes the Beautiful Mind algorithm, detailing its problem-solving model and optimization strategies. Section IV evaluates the proposed approach through comparative analysis and empirical experiments. Finally, Section V concludes the paper and outlines future research directions.

2. Combinatorial Testing

Consider a system with k parameters, where the i-th parameter has a discrete set of values V_i. Each test case for this system can be represented as a k-tuple $(x_1, x_2, x_3, \ldots, x_k)$, where $x_i \in V_i$ and $1 \leq i \leq k$ [13].

In exhaustive testing, all possible combinations of system parameters must be tested. For instance, if a system has $k = 5$ parameters and each parameter can take one of three distinct values, the total number of test cases required for exhaustive testing would be:

$3 \times 3 \times 3 \times 3 \times 3 = 81$. As the number of parameters increases, the number of required test cases grows exponentially, making exhaustive testing impractical for large systems. Hence, a more efficient approach is necessary to reduce the number of test cases while ensuring sufficient coverage.

2.1. t-Way Testing and Covering Array (CA). A more practical alternative is t-way testing, which considers only $t - wise$ interactions of parameters rather than all possible combinations ($1 < t < k$). The set of test cases generated using this approach forms a Covering Array (CA). A CA is defined as an $N \times K$ array, where N represents the number of test cases, satisfying the following properties [14]:

1. Each column i ($1 \leq i \leq k$) contains values from the corresponding set Vi.
2. Every $N \times t$ subarray covers all possible $t-way$ combinations of parameter values at least once.

For example, consider a system with two parameters, A and B, each having two possible values: 0 and 1. For $t = 2$, all four possible combinations (00, 01, 10, 11) must appear at least once in the columns representing A and B in the CA. A CA is typically denoted as $CA(N;\ t,\ v1,\ v2,\ ...,\ vk)$, where vi represents the number of possible values for each parameter. If all parameters have the same number of values ($v1 = v2 = \ldots = vk = v$), it can be abbreviated $as\ CA(N;\ t,\ k,\ v)$ or $CA(N;\ t,\ v^k)$.

2.2. Example of a Covering Array. To better illustrate the concept of a CA, consider $CA(N;\ 2,\ 4,\ 2)$, where $t = 2$, $k = 4$, and each parameter can take two values (0 or 1). If we were to use exhaustive testing, we would require $2^4 = 16$ test cases. However, using a covering array, we only need N = 5 test cases while still ensuring that all two-way combinations among the parameters are covered at least once.
In this example, the six necessary two-way interactions (AB, AC, AD, BC, BD, $and\ CD$) must each include the four possible value pairs (00, 01, 10, and 11). The minimum number of test cases required to satisfy this coverage constraint is five (Fig. 3).

The primary challenge in CA generation is to minimize the number of test cases while ensuring complete t-wise coverage. Efficient CA generation is crucial for reducing testing costs while maintaining the effectiveness of software verification.

2.3. Related work.

2.3.1. *Mathematical Methods.* This section reviews mathematical methods that directly generate Covering Arrays (CAs) within the context of Combinatorial Testing. Mathematical methods are primarily employed for specific configurations due to their inherent limitations in scalability and adaptability to diverse problem scenarios. These approaches rely on precise calculations and deterministic formulations to construct solutions, making them effective for structured and well-defined parameter spaces but less practical for large-scale or highly dynamic systems.

Among the prominent methods in this category are TConfig [15] and Combinatorial Test Services (CTS) [16]. TConfig focuses on generating optimal configurations by leveraging mathematical models to ensure full coverage within a predefined parameter space. It is particularly effective when the number of parameters and their interactions are limited and manageable. Similarly, CTS adopts a mathematical framework to tackle combinatorial problems, offering systematic and efficient test case generation techniques.

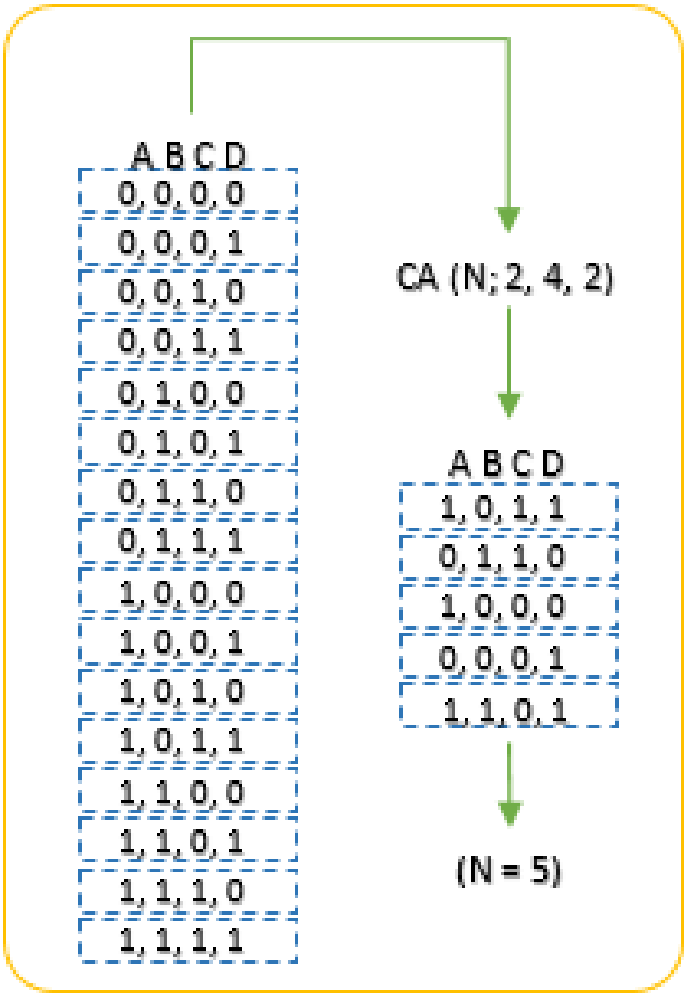

FIGURE 1. Example of CA.

CTS is well-regarded for its ability to generate compact test suites while maintaining high parameter coverage.

However, despite their efficiency in specific contexts, both TConfig and CTS struggle with large-scale problems due to computational complexity and the exponential growth of parameter interactions. Consequently, these methods are often complemented by computational techniques such as meta-heuristic algorithms, which provide greater flexibility and scalability. However, the time complexity of these methods grows rapidly with the number of parameters and interaction strength (t), limiting their use to small-scale CA problems.

2.3.2. *Computational Methods.* Computational methods typically employ greedy algorithms or meta-heuristic approaches to explore the search space efficiently. Some of the most widely used greedy strategies include Jenny [17], In-Parameter-Order-General (IPOG) [18], Automatic Efficient Test Generator (AETG) [19], Classification-Tree Editor eXtended Logics (CTE-XL) [20], Pairwise Independent Combinatorial Testing (PICT) [21], Test Vector Generator (TVG) [22], and Graph-Based Greedy Algorithm (GBGA) [23].

However, the most prominent solutions for covering array (CA) generation are based on meta-heuristic algorithms [24–33] and multi-stage algorithms [34–36]. Below, we review some notable contributions in this domain.

The first major implementation of meta-heuristic algorithms for CA generation was introduced in 2001 by Stardom [37], which employed Simulated Annealing (SA), Genetic Algorithm (GA), and Tabu Search (TS) for $t = 2$. The results indicated that SA outperformed the other two algorithms in terms of solution quality. However, due to their complex structures, these algorithms excelled in minimizing the array size but suffered from high computational costs.

In 2004, Cohen [38] extended SA for $t = 3$, and Shiba et al. [39] leveraged GA and Ant Colony Algorithm (ACA) to enhance CA generation for $t = 3$.

Later, in 2012, Ahmed et al. [40] introduced an improved CA generation approach capable of handling $t \leq 6$ by reducing the complexity of previous algorithms. While this method showed weaker performance than SA, GA, and ACA for $t \leq 3$, it outperformed greedy methods in larger configurations. Additionally, by modifying the data structure, the proposed PSTG algorithm significantly increased both test case search speed and test suite generation efficiency.

A significant advancement came in 2013 when HSS [41] introduced a new algorithm capable of handling interaction strengths up to $t = 15$. Although test suite generation time was not explicitly evaluated, HSS demonstrated high efficiency. Similarly, in [42], the Cuckoo Search Algorithm was applied to CA generation, achieving faster execution times than PSTG while maintaining comparable efficiency.

One of the main challenges in CA generation using meta-heuristic algorithms is avoiding local optima. In [11], a robust solution was proposed to address this issue, demonstrating superior results in reducing the array size compared to traditional meta-heuristic algorithms, albeit at the cost of increased computation time.

Further advancements were made in [6], where Genetic Algorithms (GA) were enhanced with optimized data structures and refined search mechanisms, significantly boosting CA generation speed. This approach successfully produced test suites up to $t = 20$, achieving high performance but still falling short of Differential Evolution-based PSO (DPSO).

In [8], the GS algorithm was introduced as a model-checking tool for CA generation. This method not only improved upon previous approaches but also featured automatic extraction of system-related information through model analysis.

The most recent development in this field is the GALP algorithm, introduced in 2021 [43]. This method integrates GA and PSOSLV [44], achieving notable improvements in both efficiency and performance. GALP represents a promising direction for future research in CA generation, balancing computational cost with optimal test coverage.

3. Implementation

This section explores the implementation of the "Beautiful Mind" algorithm as an innovative solution for the problem of test suite generation. By simulating human cognitive and emotional behaviors, this algorithm aims to provide an efficient and accurate approach. The details of the algorithm are introduced below.

3.1. Beautiful Mind algorithm. IQ stands for "Intelligence Quotient", which is a score used to measure the mental ability of people of the same age. IQ is designed in such a way that 100 is a normal score. According to Robert McCall, a psychologist from the

University of Pittsburgh, about two-thirds of the world' s population is in the 16-point range of this normal score. That is a score between 84 and 116 on the IQ test is possible for the majority of people and is considered a norm. In an interview with Live Science, McCall said that only 2.5 percent of people can score above 130; however, since there are different versions of the IQ test, to obtain more accurate statistics, the type of test should be determined.

Individuals' IQs are not necessarily fixed and can alter in relation to experiences, living environment, parents, social network members, relationships with others, and in general, according to what happens in a person' s life.

Another component that affects human learning is emotional intelligence. Emotional intelligence is a set of abilities that help us recognize and regulate emotions in ourselves and others. Many scientists believe that the effect of EQ (Emotional intelligence Quotient) on learning is even higher than IQ, but there is no specific parameter for this type of intelligence.

In this paper, we intend to simulate the behavior of several university students in reaching an answer with the help of a supervisor and an advisor as a meta-heuristic algorithm. The procedure of this algorithm is as follows:

In the Beautiful Mind algorithm (Algorithm 1), the problem-solving process begins by defining the primary parameters, including the number of students, supervisors, and advisors, along with the IQ and EQ values. Once these parameters are set, the initial population is generated randomly. In this algorithm, each student, supervisor, and advisor represents a potential solution, and these solutions serve as possible answers to the problem.

The IQ parameter plays a key role in this model. The algorithm assumes IQ values within the range of 84 to 200. To normalize these values, they are divided by 100, resulting in a range of 0.8 to 2. The proposed model defines three distinct IQ categories: students' IQ (IQ_ST), supervisors' IQ (IQ_SU), and advisors' IQ (IQ_AD). Similarly, emotional intelligence (EQ), which is crucial for learning and decision-making, is modeled using three values within the range of [0.0, 1.0]. These values include students' EQ (EQ_ST), supervisors' EQ (EQ_SU), and advisors' EQ (EQ_AD).

Students play a central role in generating initial solutions. They leverage their IQ and EQ levels to explore potential solutions. Supervisors act as guides, using their expertise and experience to direct students toward more promising areas of the solution space. Advisors, on the other hand, provide general feedback and evaluate the overall direction of the search, ensuring that students' progress aligns with the problem' s requirements.

The interaction between these three groups—students, supervisors, and advisors—drives the iterative improvement of solutions and prevents the algorithm from getting stuck in local optima. This iterative process refines the solutions in each step, eventually leading to a near-optimal answer for the given problem.

The Beautiful Mind algorithm uniquely combines cognitive and emotional aspects of problem-solving, emphasizing collaborative interactions and adaptive learning to tackle complex optimization problems effectively. This innovative approach highlights the importance of balancing exploration and exploitation, making it a promising tool for various optimization challenges.

In Algorithm 1, each "Student_Info[j]" represents a candidate covering array. The algorithm iteratively improves these candidates based on guidance from "Supervisors" and "Advisors", whose influence is weighted by their cognitive (IQ) and emotional (EQ)

Algorithm 1. Pseudo code of the Beautiful Mind algorithm
Input:

- $NumOfSupervisors \in \{1, 2\}$.
- $NumOfAdvisors \in \{0, 1, 2\}$.
- $NumOfStudents = N$ (population size).
- t, k, v (parameters of the covering array: strength, number of factors, values per factor).

Initialize:

- Generate an initial population of students, each representing a candidate CA: $Student\_Info[j]$, $j \in [1, N]$.
- Assign random IQ and EQ values to each student, supervisor, and advisor.
- $Best\_Info \leftarrow$ the best initial student (minimum CA size or highest coverage).

Main Loop:

(1) For each Supervisor $i = 1, \ldots, NumOfSupervisors$:

- For each Student $j = 1, \ldots, NumOfStudents$, update

$$\begin{aligned} Student\_Info[j] \leftarrow Student\_Info[j] &+ IQ\_Student[j]\, EQ\_Student[j] \\ &\times (Best\_Info - Student\_Info[j]) \\ &+ IQ\_Supervisor[i]\, EQ\_Supervisor[i] \\ &\times (Best\_Info - Supervisor\_Info[i]). \end{aligned}$$

- Update $Best\_Info$ with the best updated student.
- Set $Supervisor\_Info[i] \leftarrow Best\_Info$.

(2) For each Advisor $i = 1, \ldots, NumOfAdvisors$, perform the analogous update using $IQ\_Advisor[i]\, EQ\_Advisor[i]$ and $Advisor\_Info[i]$.

Output: $Best\_Info \leftarrow$ final optimized covering array $CA(N; t, k, v)$.

FIGURE 2. Pseudo code of the Beautiful Mind algorithm.

capacities. The "Best_Info" tracks the most optimal CA found so far, and the algorithm aims to minimize the number of rows N while maintaining complete t-wise coverage.

For example, consider t = 2, k = 3, v = 2. A candidate solution in Student_Info [1] has IQ = 1.2, EQ = 0.8, and the CA matrix is as follows:

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

In this matrix, each row represents a test case and each column corresponds to a parameter (A, B, and C). The vector [1-2-0-0-0-0-0-0] encodes the combination of parameters (A = 1, B = 2), where '1' and '2' denote the chosen values, and the remaining zero entries indicate uncovered value pairs. After each iteration of the algorithm, this representation is updated based on newly covered interactions until all required t-way combinations are satisfied.

The multiplication of IQ and EQ values represents the compounded influence of cognitive ability and adaptability on refining the solution. This formulation parallels weight assignment strategies in swarm intelligence algorithms, where combined factors modulate the step size toward promising solutions.

In the context of the CA generation problem, each student, supervisor, and advisor in the Beautiful Mind algorithm represents a potential test suite, i.e., a candidate Covering Array (CA). The quality of these candidate solutions is shaped by their corresponding IQ and EQ values, which influence their ability to explore the search space and refine their configurations. Through the interactions between students (explorers), supervisors (guides), and advisors (evaluators), the algorithm iteratively improves these test suites. Ultimately, the best-performing individual represents a near-optimal CA that satisfies all t-way interaction requirements. This mapping ensures that all elements of the algorithm are directly tied to the core components of the combinatorial testing objective. Table I illustrates the detailed mapping between the algorithm components and their respective roles in the CA generation task.

TABEL I. Mapping Table (Algorithm Parameters and Problem Components)

TABLE 1. Mapping Table (Algorithm Parameters and Problem Components)

Algorithm Component	Mapped to CA Problem Component
Student_Info[j]	A candidate solution representing a test suite (i.e., a Covering Array)
Supervisor_Info[i]	A guiding solution that influences the improvement of student solutions
Advisor_Info[i]	A monitoring and evaluation component for ensuring alignment with search goals
IQ_Student, IQ_Supervisor, IQ_Advisor	Cognitive strength of each individual, influencing learning and refinement
EQ_Student, EQ_Supervisor, EQ_Advisor	Emotional intelligence, affecting adaptability and collaboration during search
Best_Info	The current best candidate CA with minimal test cases and full t-wise coverage
Update(Best_Info)	Selection of the best solution based on coverage and array size
Iteration Process	Optimization loop aiming to enhance test suite quality and reduce array size
Initial Population	Randomly generated set of test suites (candidate covering arrays)

3.2. CA generation. In this section, we describe the steps of implementing the CA generation strategy using the beautiful mind algorithm.

Step 1: Covering matrix generation

One of the important issues in CA generation is how to store uncovered cases. By searching through these cases, the weight of a test case can be calculated for meta-heuristic algorithms. How to store the uncovered test cases can dramatically increase the speed of weight calculation. In this study, we employ the bit structure used in [6]. This data structure is composed of $\binom{n}{t}$ rows each of which represents a combination. Suppose a system has four parameters A, B, C, and D (n = 4) and t = 2, the first row is assigned to the combination of A (1) and B (2). The number of columns in each row equals the product of the number of values assigned by these two parameters, and the initial value of each cell in the first stage is zero (i.e. not covered). It should be noted that to identify each combination in the corresponding row, the values of the combination are stored in t columns. For instance, the first row of the above example, assuming that parameter A has three values (0, 1, 2) and parameter B has two values (0, 1), equals [1-2-0-0-0-0-0-0], in which the 1s and 2s represent the combination of A (1) and B (2) and the number of cells in a row is $t + v_A * v_B$.

Step 2: Test case weight calculation

To explain the weight calculation process for a test case, we suppose a system with four parameters A (0, 1), B (0, 1, 2), C (0, 1), and D (0, 1, 2) where t = 2. The data structure for this system is depicted in Fig. 4. Now we are going to calculate the weight of the test case (A = 0, B = 2, C = 1, D = 1). To calculate the weight of a test case, we obtain the decimal equivalent of the corresponding combination and then examine the value of that cell in the data structure. If this value is zero, one unit is added to the weight of that test case and the value of the cell will change to one. For example, the

decimal equivalent of the first row, which is related to the combination of A and B (A = 0, B = 2) (Fig 4. I), is 2. Since the corresponding cell value is zero, this value will change to one and one unit will be added to the test case weight. This procedure is continued until the last line. Numbers such as 12, 13, 14 represent parameter pair identifiers (e.g., 12 = parameters 1 and 2). Red '1' entries indicate that the corresponding combination has been covered by the selected test case. For example, for the test case {A=0, B=2, C=1, D=1}, the algorithm finds the relevant parameter pairs, computes their indices, and updates the coverage matrix from 0 to 1.

The fitness function evaluates each candidate CA as:

$$Fitness = \alpha \times Coverage + \beta \times \frac{1}{N}. \tag{1}$$

where Coverage is the percentage of covered t-way combinations, N is the current number of test cases, and α, β are weighting coefficients. After each update of Student_Info[j], the fitness function is computed, and Best_Info is updated with the candidate having the highest fitness.

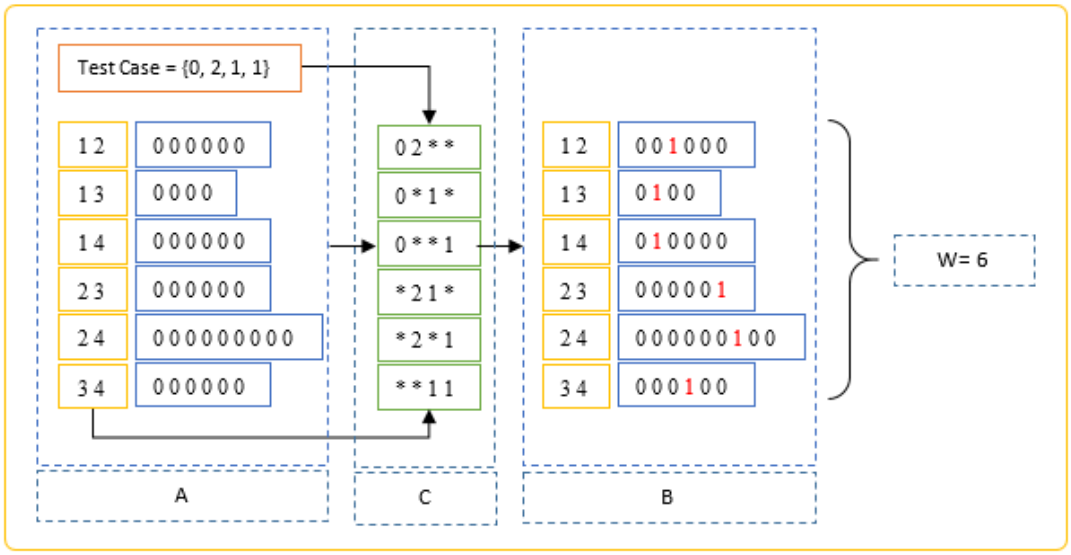

FIGURE 3. Example of the data structure for a system with four parameters and weight calculation of an instance test case.

Step 3: CA generation using BM algorithm

Each test case in the BM algorithm is the mentality of a professor or student. In fact, a test case is a solution that a student or professor presents and this solution will change over time by studying and reviewing until it reaches the desired result. In the first step, the weight of all test cases is the same and equals the number of rows in the covering matrix. Therefore, the first test case (first row) of the CA is selected randomly. After selecting the first test case, the covering matrix is updated (values of the covered cells change to one). Then the BM algorithm is applied to pick the second test case. Besides, the initial population is generated randomly, and the weight of each test case is calculated. Afterward, this population enters the BM algorithm loops to increase its weight; finally, the heaviest test case is added to the test suite (CA). This process will continue until all test cases are covered (all zeros in the covering matrix become ones).

To make the CA generation process reproducible, we summarize below the exact step-by-step procedure used in our implementation when constructing the test suite incrementally.

1. Initialize the coverage matrix M using Step 1, set all cells to 0 (uncovered), and set the test suite CA as an empty set.

2. Because M is initially all zeros, any test case covers exactly one cell per row in M and therefore all test cases have the same initial weight. To break this symmetry, select the first test case uniformly at random from the full factorial space, append it to CA, and update M by setting the corresponding cells to 1.

3. While M still contains at least one 0 (i.e., some required t-way interactions are uncovered), repeat the following steps.

3.1. Generate an initial population of candidate test cases (students) at random.

3.2. For each candidate test case x, compute its weight w(x) as the number of currently-uncovered cells that would be covered by x. This is done by scanning all rows of M (all t-way parameter combinations), computing the column index of the value tuple induced by x (as in Step 2), and counting a hit whenever the corresponding cell in M equals 0.

3.3. Compute the candidate fitness using (1), where Coverage is evaluated after hypothetically adding x to the current partial suite and $N = |CA| + 1$.

3.4. Apply the BM update loop (Algorithm 1) for Max_iter iterations using the above fitness evaluation and keep Best_Info as the best candidate found.

3.5. Append Best_Info (the heaviest test case) to CA and permanently update M by setting all newly covered cells to 1.

3.6. If Best_Info provides $w(x) = 0$ (no new coverage), re-initialize part of the population and continue, to avoid stagnation.

4. Terminate when all entries of M are 1. The resulting CA is the final test suite that achieves full t-way coverage.

In the worked example of Fig. 4, the test case {A=0, B=2, C=1, D=1} covers one value tuple for each parameter pair (AB, AC, AD, BC, BD, and CD), yielding $W = 6$ newly covered cells at the beginning of the construction.

4. Evaluation

The purpose of the evaluation in the CA is to compare the number of test cases for a particular configuration. In other words, the smaller the number of test cases in the test suite (provided that it covers all the states of the CA), the more powerful the algorithm. In this solution, we have added computational approaches (Jenny, TConfig, PICT, and IPOG) to the evaluation tables, along with the meta-heuristic ones (CPSO, PSTG, CS, GS, and BM).

To evaluate the effectiveness of the proposed Beautiful Mind (BM) algorithm, we compared it with four well-established meta-heuristic algorithms that are widely used in the context of Covering Array (CA) generation:

- CPSO (Cooperative Particle Swarm Optimization): An enhanced version of the PSO algorithm that employs multiple swarms working cooperatively to improve convergence and search diversity.
- PSTG (Pairwise Swarm Test Generation): A PSO-based method specifically designed for efficient generation of pairwise ($t = 2$) test suites.
- CS (Cuckoo Search): A meta-heuristic algorithm inspired by the brood parasitism of cuckoos, using Lévy flight steps for effective global search.
- GS (Genetic Strategy): A genetic algorithm that simulates natural evolution through selection, crossover, and mutation to evolve high-quality test suites.

These algorithms were selected due to their prevalence in existing literature, effectiveness in prior studies, and accessibility for implementation. Including these techniques ensures a comprehensive and fair comparison with the proposed BM algorithm.

In the evaluation tables, each row represents a configuration that gives the number of test cases produced by the algorithm. The smaller number in each row indicates the superiority of that algorithm over its competitors, which we have highlighted that cell. The specification of the hardware platform is: Windows 7, 2.20GHz core ^{TM} i7QM CPU, and 6GB RAM. Its implementation has been done in the Eclipse IDE (JDK 1.8). The parameters of the proposed solution are shown in Table II.

Table II. Parameters of the proposed solution

TABLE 2. Parameters of the proposed solution

parameters	values
NumOfSupervisor	1-2
NumOfAdvisor	0-2
NumOfStudent	5-100
IQ_Supervisor	0.8-2.0
IQ_Advisor	0.8-2.0
IQ_student	0.8-2.0
EQ	0.0-1.0

4.1. Evaluation with interaction strength 2. The first evaluation in this paper concerns 13 configurations with interaction strength 2, which is illustrated in Table 2. The first CA configuration is (N; 2, 2^7) in which the Jenny algorithm with 8 test cases performed the worst among the available solutions. TConfig, PICT, IPOG, CPSO, PSO, CS, and HSS, each managed to produce a test suite with 7 test cases. After all, GS and the proposed solution with 6 test cases were the best for this configuration. In the CA configuration (N; 2, 3^3), the two algorithms TConfig and PICT with 10 and other solutions with 9 test cases could generate the test suite. In the CA configuration (N; 2, 3^4), Jenny and PICT obtained the worst results with 13 test cases. TConfig with 10 test cases is the second worst one, and other solutions with 9 test cases have the best efficiency. In CA (N; 2, 3^5), the CPSO, CS, GS, HSS, and the proposed solution with 11 test cases are the best among all. In CA (N; 2, 3^6), AI-based solutions except for CPSO, succeeded in producing test suites with 13 test cases. The next configuration is CA (N; 2, 3^7) where CS, GS, HSS, and the proposed solution got the best results with 14 test cases. In the CA configuration (N; 2, 3^8), all solutions except Jenny, TConfig, and PICT performed the best with 15 test cases. In the next configuration which has 9 parameters (CA (N; 2, 3^9)), three solutions namely IPOG, GS, and BM were able to produce the best results with 15 test cases. In CA

(N; 2, 3^{10}) which includes 10 parameters, only the IPOG algorithm was able to produce a minimum test suite with 15 test cases. For configurations with 11 and 12 parameters, GS and BM solutions with 16 test cases were more efficient than other algorithms. Finally, in the two configurations CA (N; 2, 4^7) and CA (N; 2, 5^7), the proposed solution had the best results among the existing solutions with 24 and 35 test cases, respectively.

Table 2. Comparing the efficiency of various solutions for configurations with interaction strength 2

TABLE 3. Table 2. Comparing the efficiency of various solutions for configurations with interaction strength 2

	Jenny N	TConfig N	PICT N	IPOG N	CPSO N.Best	PSO N.Best	CS N.Best	GS N.Best	HSS N.Best	BM N.Best
CA (N; 2, 27)	8	7	7	7	7	7	7	**6**	7	**6**
CA (N; 2, 33)	**9**	10	10	**9**	**9**	**9**	**9**	**9**	**9**	**9**
CA (N; 2, 34)	13	10	13	**9**	**9**	**9**	**9**	**9**	**9**	**9**
CA (N; 2, 35)	14	14	13	15	**11**	12	**11**	**11**	**11**	**11**
CA (N; 2, 36)	15	15	14	15	14	**13**	**13**	**13**	**13**	**13**
CA (N; 2, 37)	16	15	16	15	15	15	**14**	**14**	**14**	**14**
CA (N; 2, 38)	17	17	16	**15**	**15**	**15**	**15**	**15**	**15**	**15**
CA (N; 2, 39)	18	17	17	**15**	16	17	16	**15**	16	**15**
CA (N; 2, 310)	19	17	18	**15**	16	17	17	16	17	16
CA (N; 2, 311)	17	20	18	17	**16**	17	18	**16**	17	**16**
CA (N; 2, 312)	19	20	19	21	17	18	18	**16**	17	**16**
CA (N; 2, 47)	28	28	27	29	25	26	25	24	25	**24**
CA (N; 2, 57)	37	40	40	45	36	37	37	36	36	**35**

4.2. Evaluation with interaction strength 3. Table 3 is devoted to evaluating 14 configurations with interaction strength 3. In the first configuration CA (N; 3, 2^7) as well as CA (N; 3, 2^9) and CA (N; 3, 210), five solutions including CPSO, CS, GS, HSS, and the proposed solution produced the best results with 12, 16, and 16 test cases, respectively. In the CA configuration (N; 3, 3^4), the PSO, GS, HSS, and the proposed solution overcame their competitors with 27 test cases. In the CA configuration (N; 3, 3^7), the AI-based solutions achieved better results, and among them, three solutions namely CS, HSS, and the proposed solution, with 48 test cases, managed to produce a test suite. Next in the configurations CA (N; 3, 3^6), CA (N; 3, 3^8), CA (N; 3, 3^9), CA (N; 3, 3^{10}), CA (N; 3, 3^{11}), and CA (N; 3, 3^{12}), the proposed solution is the most efficient approach with 39, 52, 56, 59, 62, and 65 test cases, respectively. Eventually, in CA (N; 3, 4^7), IPOG and the proposed solution are the most efficient among them all with 112 test cases.

Table 3. Comparing the efficiency of various solutions for configurations with interaction strength 3

TABLE 4. Table 3. Comparing the efficiency of various solutions for configurations with interaction strength 3

	Jenny N	TConfig N	PICT N	IPOG N	CPSO N.Best	PSO N.Best	CS N.Best	GS N.Best	HSS N.Best	BM N.Best
CA (N; 3, 27)	14	16	15	16	**12**	13	**12**	**12**	**12**	**12**
CA (N; 3, 28)	14	18	17	18	16	16	16	14	15	**12**
CA (N; 3, 29)	17	20	17	20	**16**	17	**16**	**16**	**16**	**16**
CA (N; 3, 34)	34	32	34	32	30	**27**	28	**27**	**27**	**27**
CA (N; 3, 35)	40	40	43	41	38	39	38	38	38	**37**
CA (N; 3, 210)	18	20	18	20	**16**	17	**16**	**16**	**16**	**16**
CA (N; 3, 36)	51	48	48	46	42	45	43	43	41	**39**
CA (N; 3, 37)	51	55	51	55	49	50	**48**	49	**48**	**48**
CA (N; 3, 38)	58	58	59	56	53	54	53	54	53	**52**
CA (N; 3, 39)	62	64	63	63	58	58	58	58	58	**56**
CA (N; 3, 310)	65	68	65	66	61	62	62	61	62	**59**
CA (N; 3, 311)	65	72	70	70	63	64	66	63	63	**62**
CA (N; 3, 312)	68	77	72	73	68	67	70	67	66	**65**
CA (N; 3, 47)	124	122	124	**112**	115	116	117	116	115	**112**

4.3. Evaluation with interaction strength 4. Table 4 deals with the evaluation of configurations with interaction strength 4. In the first three configurations including CA (N; 4, 2^7), CA (N; 4, 2^8), and CA (N; 4, 2^9), the proposed solution performed more efficiently than its competitors. In CA (N; 4, 2^{10}) and CA (N; 4, 3^5) configurations, the GS strategy obtained the best results by producing a test suite with 25 and 88 test cases, respectively. In the two configurations CA (N; 4, 3^6) and CA (N; 4, 3^8), the two strategies GS and BM with 129 and 171 test cases produced the best output. The HSS strategy with 222 test cases is stronger than others in CA (N; 4, 3^{11}). At last, in the other three remaining configurations including CA (N; 4, 3^9), CA (N; 4, 3^{10}), and CA (N; 4, 3^{12}), the GS strategy achieved the best outcomes.

Table 4. Comparing the efficiency of various solutions for configurations with interaction strength 4

TABLE 5. Table 4. Comparing the efficiency of various solutions for configurations with interaction strength 4

	Jenny N	TConfig N	PICT N	IPOG N	CPSO N.Best	PSTG N.Best	CS N.Best	GS N.Best	HSS N.Best	BM N.Best
CA (N; 4, 27)	31	36	32	35	24	29	27	27	27	**24**
CA (N; 4, 28)	37	38	35	39	32	33	31	30	30	**29**
CA (N; 4, 29)	37	41	41	41	33	34	31	33	30	**25**
CA (N; 4, 210)	39	45	43	46	37	34	28	**25**	34	35
CA (N; 4, 35)	109	97	100	97	94	96	94	**88**	94	91
CA (N; 4, 36)	140	141	142	141	132	133	132	**129**	131	**129**
CA (N; 4, 37)	169	166	168	167	153	155	154	152	152	**151**
CA (N; 4, 38)	187	190	189	192	174	175	173	**171**	172	**171**
CA (N; 4, 39)	206	213	211	210	191	195	195	**187**	191	189
CA (N; 4, 310)	221	235	231	233	211	210	211	**206**	208	208
CA (N; 4, 311)	236	258	249	251	226	222	229	223	**222**	223
CA (N; 4, 312)	252	272	269	272	242	244	253	**236**	242	237

4.4. Evaluation with greater interaction strengths. The last evaluation in this paper is for configurations with an interaction strength greater than 4, the results of which are provided in Table 8. In this table, "NS" (which stands for Not Succeeded) means that the relevant strategy was unable to produce a test suite for that particular configuration, and also "> day" implies that the strategy did not produce a test suite in less than one day. As can be seen, the TConfig strategy could produce test suites in less than 24 hours just for two configurations CA (N; 5, 3^7) and CA (N; 6, 3^8). Also, both PSO and CS strategies produce test suites only for the two above-mentioned configurations and do not support configurations with an interaction strength greater than 5. This evaluation reveals that the proposed solution performed the best and is the most efficient among its competitors in all cases.

Table 5. Comparing the efficiency of various solutions for configurations with interaction strengths greater than 4

TABLE 6. Table 5. Comparing the efficiency of various solutions for configurations with interaction strengths greater than 4

	Jenny N	TConfig N	PICT N	IPOG N	CPSO N.Best	PSO N.Best	CS N.Best	GS N.Best	HSS N.Best	BM N.Best
CA (N; 5, 37)	458	477	452	466	441	441	434	431	431	**429**
CA (N; 6, 38)	1466	1515	1455	1409	1397	1402	1401	1398	1399	**1389**
CA (N; 7, 39)	4746	¿day	4618	NS	4422	NS	NS	4437	4420	**4410**
CA (N; 8, 310)	14999	¿day	14599	NS	13925	NS	NS	13907	13912	**13894**
CA (N; 9, 311)	47009	¿day	45521	NS	43587	NS	NS	43808	43808	**43539**
CA (N; 10, 312)	147004	¿day	141990	NS	135498	NS	NS	136096	1396090	**135380**
CA (N; 11, 312)	305797	¿day	278993	NS	268173	NS	NS	267630	267628	**267803**
CA (N; 12, 214)	9422	¿day	9112	NS	8882	NS	NS	8890	8885	**8857**
CA (N; 13, 214)	13251	¿day	12441	NS	11588	NS	NS	10251	11210	**10895**
CA (N; 14, 215)	26579	¿day	25036	NS	23889	NS	NS	23377	23384	**20895**
CA (N; 15, 216)	53977	¿day	51127	NS	45838	NS	NS	46575	45535	**41818**

In all experiments, the initial population consisted of 30 students (candidate solutions), with 2 supervisors and 1 advisor. The algorithm typically converged within 50–100 iterations, depending on the complexity of the configuration (i.e., t, k, and v). The cognitive

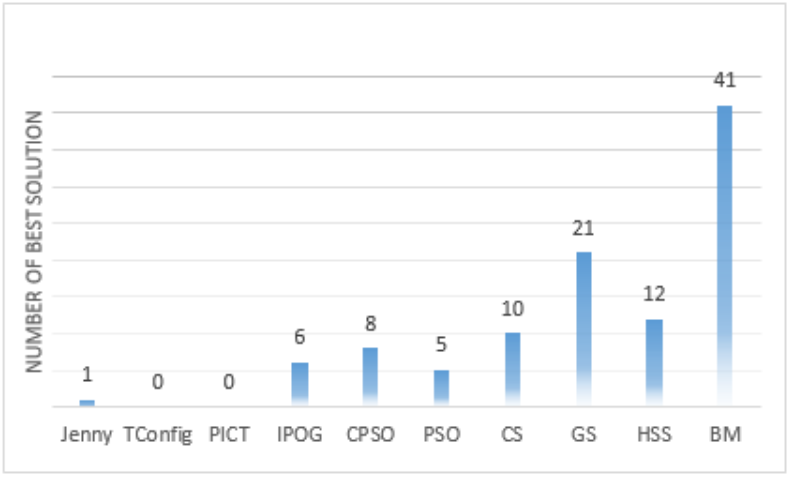

FIGURE 4. The number of case studies in which each solution in Tables 2–5 outperforms its competitors.

and emotional weights helped maintain a balance between intensification and diversification, improving convergence stability.

The next evaluation, using the Wilcoxon test, pertains to Tables 2–5, with Table 6 dedicated to this assessment. Table 6 consists of four parts. The first part corresponds to Table 2. As shown, in the first, second, and third parts—corresponding to Tables 3, 4, and 5, respectively—BM has a statistically significant difference compared to other algorithms except for DPSO, and its superiority is clearly evident.

To precisely compare BM with DPSO, the Ranks section can be referred to, which indicates that BM outperforms DPSO. The fourth part corresponds to Table 5. Since many results for the strategies in this part are unavailable, these missing values are considered as "Miss" in the Wilcoxon test. The test partially disregards the corresponding values for these missing entries, which can negatively impact the final result. Despite this, BM shows statistically significant differences with three strategies—Jenny, PICT, and DPSO. However, it does not have a statistically significant difference with three other strategies—TConfig, IPOG, and PSTG. This lack of significance is attributed to the fact that these three strategies were only able to generate test sequences in two cases.

4.5. Time Comparison of the Proposed Solution. The next comparison pertains to the execution time for generating the CA. This part of the evaluation focuses on comparing the proposed solution with AI-based approaches. As previously mentioned, one of the factors influencing the CA generation time in the proposed solution is the determination of algorithm parameters. For this comparison, the values of popsize, crossover rate, and mutation rate are set to 50, 0.8, and 0.8, respectively. The comparison considers the time required to generate a single test case. For instance, the DPSO algorithm generates a test sequence with 14 test cases in 30 milliseconds. Therefore, the time required for one test case is calculated as 2.14 milliseconds (30/14). The results of this comparison are presented in Table 7. As observed, the proposed strategy significantly outperforms the AI-based strategies in terms of efficiency.

Table 7. Time Comparison of the Proposed Solution

TABLE 7. Comparison of the Proposed Solution Using the Wilcoxon Test

Table	Comparison	BM$¡$	BM$¿$	BM$=$	Z	Asymp. Sig. (2-tailed)
Table 2	BM-Jenny	11	0	1	-2.831	.005
Table 2	BM-TConfig	12	0	0	-2.971	.003
Table 2	BM-PICT	12	0	0	-2.965	.003
Table 2	BM-IPOG	7	1	4	-2.043	.041
Table 2	BM-PSTG	7	0	5	-2.2781	.023
Table 2	BM-HSS	6	0	6	-2.2787	.024
Table 2	BM-DPSO	3	1	8	-0.0000	1.00
Table 3	BM-Jenny	15	0	0	-2.947	.002
Table 3	BM-TConfig	15	0	0	-2.941	.002
Table 3	BM-PICT	15	0	0	-2.952	.002
Table 3	BM-IPOG	15	0	0	-2.965	.002
Table 3	BM-PSTG	14	0	1	-2.840	.005
Table 3	BM-HSS	9	0	5	-2.935	.004
Table 3	BM-DPSO	5	3	7	-.184	.854
Table 4	BM-Jenny	12	0	0	-2.521	.012
Table 4	BM-TConfig	12	0	0	-2.524	.012
Table 4	BM-PICT	12	0	0	-2.524	.012
Table 4	BM-IPOG	12	0	0	-2.524	.012
Table 4	BM-PSTG	12	0	0	-2.536	.011
Table 4	BM-HSS	11	0	1	-2.536	.019
Table 4	BM-DPSO	3	2	7	-.680	.496
Table 5	BM-Jenny	11	0	0	-2.934	.003
Table 5	BM-TConfig	2	0	0	-1.342	.180
Table 5	BM-PICT	12	0	0	-3.059	.002
Table 5	BM-IPOG	2	0	0	-1.352	.180
Table 5	BM-PSTG	2	0	0	-1.342	.180
Table 5	BM-HSS	11	0	0	-2.934	.003
Table 5	BM-DPSO	5	0	0	-2.023	.043

TABLE 8. Time Comparison of the Proposed Solution

	CPSO time	DPSO time	GS time	**BM time**
CA (2, 37)	0.28	2.14	0.41	**0.12**
CA (3, 37)	0.40	4.79	0.68	**0.18**
CA (4, 37)	0.82	9.86	0.66	**0.21**
CA (3, 38)	0.94	6.92	0.96	**0.29**
CA (3, 39)	1.35	9.82	1.21	**0.41**
CA (3, 310)	1.82	13.72	2.02	**0.67**
CA (3, 47)	0.53	4.82	0.63	**0.23**
CA (3, 57)	0.57	7.96	0.57	**0.23**
CA (3, 67)	0.57	5.15	0.45	**0.22**
CA (3, 77)	0.58	3372	0.51	**0.22**

5. Conclusion and future work

Artificial intelligence and meta-heuristic solutions hold a vital place in computer science. Numerous meta-heuristic algorithms have been developed to address a wide range

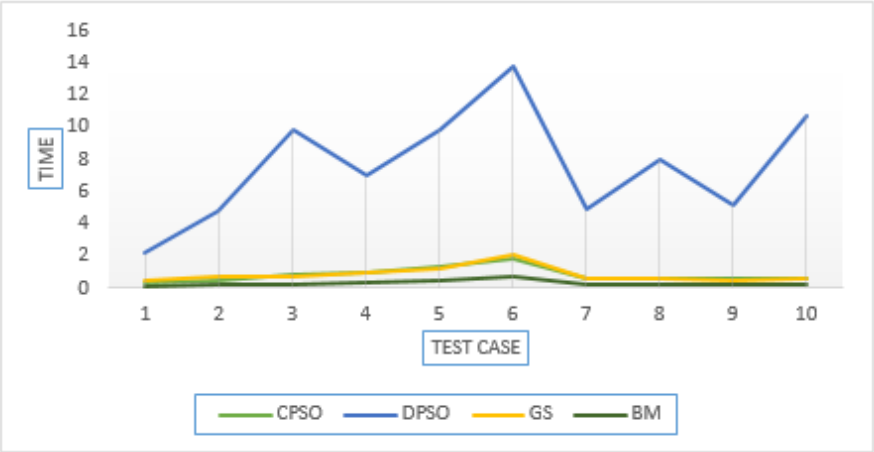

FIGURE 5. Time comparison of the proposed solution.

of problems, offering solutions to many NP-hard problems in the shortest possible time. The emergence of diverse challenges and the critical importance of time efficiency in solving these problems have driven the continuous creation of new algorithms. The field of meta-heuristics is ever-evolving, with new algorithms introduced each year, many of which demonstrate superior performance in specific domains compared to their predecessors.

One significant application of meta-heuristic algorithms in recent years has been optimizing the generation of test cases in Combinatorial Testing (CT). A common challenge in generating test suites in CT is the tendency to get trapped in local optima. This motivated us to propose a novel meta-heuristic solution for test suite generation, which we named "Beautiful Mind" (BM). Unlike conventional meta-heuristic algorithms, BM simulates human cognitive behavior, leveraging both intellectual and emotional factors to reach optimal solutions.

As demonstrated by the evaluation results, the proposed BM algorithm delivers significantly better outcomes compared to its strong competitors. Additionally, with minor adaptations, this solution has the potential to be effectively applied to other problems, producing highly favorable results.

Declarations

Ethical Approval. This article does not contain any studies with human participants or animals performed by any of the authors.

Competing interests. The authors have no relevant financial or non-financial interests to disclose.

Funding. The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Availability of data and materials. Not applicable.

Authors' contributions. Sajad Esfandyari: Conceptualization, Methodology, Software, Writing–original draft. Vahid Rafe: Supervision, Writing–review & editing. Einollah Pira: Supervision, Writing–review. All authors read and approved the final manuscript.

References

[1] C. Nie and H. Leung, "A survey of combinatorial testing," CM Comput. Surv., vol. 43, no. 2, pp. 11.1-11.29, 2011.

[2] D. Kuhn and M. Reilly, "An investigation of the applicability of design of experiments to software testing," in 27th Annual NASA Goddard/IEEE Software Engineering Workshop, 2002. Proceedings, 2002.

[3] L. Yousofvand, S. Soleimani, V. Rafe and S. Esfandyari, "Automatic program bug fixing by focusing on finding the shortest sequence of changes," Artificial Intelligence Review, vol. 57, no. 2, p. 39, 2024.

[4] S. Esfandyari, L. Yousofvand, E. Pira and V. Rafe, "optimal production of the test suite by the combinatorial testing method by applying changes in the gravitational search algorithm for the uniform strength cover array," TABRIZ JOURNAL OF ELECTRICAL ENGINEERING, vol. 54, no. 3, pp. 269-279, 2024.

[5] S. Esfandyari and V. Rafe, "Correction to: GALP: a hybrid artificial intelligence algorithm for generating covering array," Soft Computing, 2021.

[6] S. Esfandyari and V. Rafe, "A tuned version of genetic algorithm for efficient test suite generation in interactive t-way testing strategy," Information and Software Technology, vol. 94, pp. 165-185, 2018.

[7] M. Farokhian, J. Shoaei and S. Esfandyari, "GWC: A tool for automatic web data extraction," in 13th Symposium on Advances in Science and Technology: Sustainable Land of Computer and Information Technology, 2018.

[8] S. Esfandyari and V. Rafe, "Extracting Combinatorial Test parameters and their values using model checking and evolutionary algorithms," Applied Soft Computing, vol. 91, pp. 1-19, 2020.

[9] L. Yousofvand, S. Soleimani and V. Rafe, "Automatic bug localization using a combination of deep learning and model transformation through node classification," Software Quality Journal, vol. 31, no. 4, pp. 1045-1063, 2023.

[10] E. Pira, V. Rafe and S. Esfandyari, "A three-phase approach to improve the functionality of t-way strategy," Soft Computing, pp. 1-21, 2023.

[11] H. Wu, C. Nie, F.-C. Kuo, H. Leung and C. J. Colbourn, "A Discrete Particle Swarm Optimization for Covering Array Generation," IEEE Transactions on Evolutionary Computation, vol. 19, no. 4, pp. 575-591, 2015.

[12] L. Yousofvand, A. Fathi and F. Abdali-Mohammadi, "Person identification using ECG signal's symbolic representation and dynamic time warping adaptation," Signal, Image and Video Processing, vol. 13, pp. 245-251, 2019.

[13] S. Esfandyari, D. Giveki and M. Farokhzadi, "A Novel Approach to Enhancing Defense: Metaheuristic Algorithms for Optimal Structuring of Covering Arrays and Efficient Test Suite Generation," Aerospace Defense, vol. 2, no. 4, pp. 58-77, 2024.

[14] E. Pira, V. Rafe and S. Esfandyari, "Minimum Covering Array Generation Using Success-History and Linear Population Size Reduction based Adaptive Differential Evolution Algorithm," TABRIZ JOURNAL OF ELECTRICAL ENGINEERING, vol. 52, no. 2, pp. 77-89, 2022.

[15] A. W. Williams, "Determination of Test Configurations for Pair-Wise Interaction Coverage," in IFIP TC6/WG6.1 13th International Conference on Testing Communicating Systems: Tools and Techniques, Kluwer, B.V., Deventer, The Netherlands, 2000, pp. 59–74, 2000.

[16] A. Hartman, "Software and Hardware Testing Using Combinatorial Covering Suites," Springer, vol. 34, 2005.

[17] B. Jenkins, "Jenny download web page," Bob Jenkins' Website, 2019. [Online]. Available: http://burtleburtle.net/bob/math/jenny.html.

[18] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun and J. Lawrence, "IPOG: a general strategy for t-way software testing," in 4th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, IEEE Computer Society, Tucson, AZ, 2007.

[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," IEEE Transactions on Software Engineering, vol. 23, no. 7, pp. 437 - 444, 1997.

[20] M. Grochtmann and K. Grimm, "Classification Trees for Partition Testing," SOFTWARE TESTING, VERIFICATION AND RELIABILIIY, vol. 3, no. 2, pp. 63-82, 1993.

[21] J. Czerwonka, "Pairwise testing in real world: practical extensions to test case generator," in 24th Pacific Northwest Software Quality Conference, IEEE Computer Society, Portland, OR, USA, 2006.

[22] J. Arshem, "TVG download page," 2019. [Online]. Available: http://sourceforge.net/projects/tvg.

[23] J. Torres-JimenezJose and C. Perez-Torres, "A greedy algorithm to construct covering arrays using a graph representation," Information Sciences, vol. 477, pp. 234-245, 2019.
[24] L. Gonzalez-Hernandez, N. Rangel-Valdez and J. Torres-Jimenez, "Construction of mixed covering arrays of variable strength using a tabu search approach," in International Conference on Combinatorial Optimization and Applications, Berlin, Heidelberg, 2010.
[25] E. Rodriguez-Tello and J. Torres-Jimenez, "Memetic algorithms for constructing binary covering arrays of strength three," in International Conference on Artificial Evolution (Evolution Artificielle), Berlin, Heidelberg, 2009.
[26] J. Bracho-Rios, J. Torres-Jimenez and E. Rodriguez-Tello, "A new backtracking algorithm for constructing binary covering arrays of variable strength," in Mexican International Conference on Artificial Intelligence, Berlin, Heidelberg, 2009.
[27] H. Avila-George, J. Torres-Jimenez, L. Gonzalez-Hernandez and V. Hernández, "Metaheuristic approach for constructing functional test-suites," IET software, vol. 7, no. 2, pp. 104-117, 2013.
[28] A. K. Alazzawi, H. M. Rais and S. Basri, "HABC: Hybrid Artificial Bee Colony For Generating Variable T-Way Test Sets," Journal of Engineering Science and Technology, vol. 7, no. 13, 2019.
[29] A. K. Alazzawi, H. M. Rais, S. Basri and Y. A. Alsariera, "Pairwise Test Suite Generation Based on Hybrid Artificial Bee Colony Algorithm," in Advances in Electronics Engineering, pp. 137-145, 2020.
[30] A. k. Alazzawi1, H. M. Rais, S. Basri, Y. A. Alsariera and L. F. Capretz, "HABCSm: A Hamming Based t-way Strategy Based on Hybrid Artificial Bee Colony for Variable Strength Test Sets Generation," International Journal of Computer Communications & Control, vol. 16, no. 5, pp. 1-18, 2021..
[31] Z. Abbasi, S. Esfandyari and V. Rafe, "Covering array generation using teaching learning base optimization algorithm," Tabriz Journal of Electrical Engineering, vol. 48, no. 1, pp. 161-171, 2018.
[32] S. Esfandyari and V. Rafe, "Using the Particle Swarm Optimization Algorithm to Generate the Minimum Test Suite in Covering Array with Uniform Strength," Soft Computing Journal, vol. 8, no. 2, pp. 66-79, 2021.
[33] s. esfandyari and v. rafe, "A Hybrid solution for Software testing to minimum test suite generation using hill climbing and bat search algorithms," Tabriz Journal of Electrical Engineering, vol. 46, no. 3, pp. 25-35, 2016.
[34] I. Izquierdo-Marquez, J. Torres-Jimenez, B. Acevedo-Juárez and H. Avila-George, "A greedy-metaheuristic 3-stage approach to construct covering arrays," Information Sciences, vol. 460, pp. 172-189, 2016.
[35] J. Torres-Jimenez, H. Avila-George and I. Izquierdo-Marquez, "A two-stage algorithm for combinatorial testing," Optimization Letters, vol. 11, no. 3, pp. 457-469, 2017.
[36] J. Torres-Jimenez, D. O. Ramirez-Acuna, B. Acevedo-Juárez and H. Avila-George, "New upper bounds for sequence Covering Arrays using a 3-stage approach," Expert Systems with Applications, vol. 207, p. 118022, 2022.
[37] J. Stardom, "Metaheuristics and the Search for Covering and Packing Array," Thesis (M.Sc.), Simon Fraser University, 2001, 2001.
[38] M. Cohen, "Designing Test Suites for Software Interaction Testing," PhD Thesis Department of Computer Science, pp. University of Auckland, New Zealand, 2004.
[39] T. Shiba, T. Tsuchiya and T. Kikuno, "Using artificial life techniques to generate test cases for combinatorial testing," in 28th Annual International Computer Software and Applications Conference, Hong Kong, China, 2004.
[40] B. S. Ahmed, K. Z. Zamli and C. P. Lim, "Application of Particle Swarm Optimization to uniform and variable strength covering array construction," Applied Soft Computing, vol. 12, no. 4, p. 1330–1347, 2012.
[41] A. R. A. Alsewari and K. Z. Zamli, "Design and implementation of a harmony-search-based variable-strengtht-way testing strategy with constraints support," Information and Software Technology, vol. 54, no. 6, p. 553–568, 2012.
[42] B. S. Ahmed, T. Sh. Abdulsamad and M. Y. Potrus, "Achievement of minimized combinatorial test suite for configuration-aware software functional testing using the Cuckoo Search algorithm," Information and Software Technology, vol. 66, p. 13–29, 2015.
[43] s. esfandyari and V. Rafe, "GALP: a hybrid artificial intelligence algorithm for generating covering array," soft computing, vol. 25, p. 7673–7689, 2021.

[44] J. TenreiroMachado, S. M. Pahnehkolaei and A. Alfi, "Complex-order particle swarm optimization," Communications in Nonlinear Science and Numerical Simulation, vol. 92, 2021.