Integration of Grey Wolf Optimizer Algorithm with Combinatorial Testing for Test Suite Generation

Muhamad Asyraf Anuar, Rosziati Ibrahim, Mazidah Mat Rejab, Nurezayana Zainal
Department of Software Engineering, Universiti Tun Hussein Onn Malaysia, Johor, Malaysia

***Abstract*—Combinatorial Testing (CT) is a software testing technique designed to detect defects in complex systems by efficiently covering diverse combinations of input parameters within given time and resource constraints. A common strategy in CT is t-way testing, which ensures that all possible interactions among any t parameters are tested at least once. The Grey Wolf Optimization Algorithm (GWOA) is a nature-inspired metaheuristic that has been successfully applied to various optimization problems. In this study, we introduce the Combinatorial Grey Wolf Optimization Algorithm (CGWOA), which integrates GWOA with CT to enhance test suite generation. Effectiveness of CGWOA is evaluated through experiments on a real-world software system, where it is found that the number of test cases was reduced by 98%, from 3000 to 40, while still ensuring complete 2-way interaction coverage. Experimental results demonstrate that CGWOA consistently produces smaller test suites compared to pure computation methods such as Jenny, IPOG, IPOG-D and TConfig, as CGWOA consistently outperformed, especially in handling both lower and higher interaction strengths. In scenarios with binary parameters, CGWOA delivered the smallest test suites, while in more complex configurations, even in MCA settings, it showed impressive scalability, outperforming the other algorithms. Statistical analysis using the Wilcoxon signed-rank test revealed that the proposed approach significantly outperforms existing methods, with all p-values less than 0.02 after applying the Holm correction. The experimental results demonstrate that the proposed CGWOA approach advances software testing by efficiently minimizing the number of test cases required to achieve complete test coverage.**

Keywords*—Grey wolf optimizer algorithm; combinatorial testing; metaheuristics; t-way testing

I. Introduction

Combinatorial Testing (CT) has been widely recognized for its effectiveness in identifying software defects [1], as demonstrated by numerous empirical studies. Its success depends on a well-structured set of test cases, known as a Covering Array (CA), which ensures that all interactions among the parameters influencing the behavior of the System Under Testing (SUT) are covered [2]. CT detects failures by examining interactions between parameters in the testing process. It has been used to automate test suite generation, aiming to produce a reliable product that is usable across a wide range of inputs. However, the high cost of CT can be a significant obstacle to its widespread adoption. Developers spend approximately 50% of their time identifying and fixing bugs [3] which highlights the need for more efficient and cost-effective testing strategies.

As software systems grow in complexity and size, the number of possible input combinations become overwhelmingly large, making complete coverage challenging with traditional CT methods. To address this, researchers and practitioners are exploring cost-reduction strategies, such as leveraging machine learning to automate test case generation [4] or prioritizing test cases based on their defect detection potential [5]. Despite these challenges, CT remains a crucial approach for ensuring software quality and minimizing defects in complex systems.

To tackle CT challenges, various heuristic search techniques have been introduced to optimize test case selection and reduce the cost of CT. CT employs a covering array as a test suite to systematically cover parameter combinations, aiming to balance the number of test cases with their effectiveness in detecting failures [3]. Among these approaches, the GWOA has gained attention for its ability to balance exploration and exploitation, where it first explores the search space and then converges towards the best solutions. GWOA is a metaheuristic optimization method [6] inspired by the hunting behavior of grey wolves and has been applied to various optimization problems.

This study proposes an innovative CT approach leveraging GWOA, combining the strengths of both techniques to maximize test coverage while minimizing the number of test cases. It aims to investigate the effectiveness of integrating the Grey Wolf Optimizer Algorithm with Combinatorial Testing to reduce test suite size while ensuring complete t-way interaction coverage across diverse parameter configurations. The goal is to assess whether the proposed CGWOA approach offers a more scalable and efficient solution than traditional combinatorial testing techniques.

A. Combinatorial Testing

Combinatorial testing is a software testing technique that systematically covers combinations of input parameters to detect interaction faults efficiently. Instead of testing every possible configuration, which can be infeasible, it focuses on ensuring that all t-way combinations are tested at least once. A single test suite can be designed to cover all t-wise combinations of these parameters [1] while minimizing the total number of test cases.

This study addresses the combinatorial optimization challenge of t-way interaction test generation, where t denotes the interaction strength or combination degree. The goal is to generate minimal yet effective test suites that ensure full t-wise coverage across complex software configurations. In recent years, various methodologies have been proposed to tackle this problem, leading to numerous computation-based solutions. Despite these advances, many existing approaches still suffer from limitations in adaptability [7], especially when dealing with systems involving diverse parameter types and values.

B. Overview of T-way Testing

A test suite is structured as an N × k array, where N is the number of test cases and k is the number of parameters. Each row represents a unique combination of parameter values. The suite is built by first enumerating all t-way value combinations (t-tuples) and then selecting the minimal number of test cases required to cover them. These test cases are used to assess the system under different interaction scenarios, enabling the identification of faults that arise from specific parameter pairings. The following formal definitions clarify key terms relevant to this approach:

- T-way testing is a combinatorial testing method that ensures all possible combinations of t input parameters are exercised at least once. It provides a balance between test coverage and resource efficiency.
- A Covering Array (CA), denoted as CA(N; t; v^p), is a matrix covering all t-way combinations where each parameter has v values.
- A Mixed Covering Array (MCA), denoted as MCA(N; t; v_1^p_1, v_2^p_2, ..., v_i^p_i), generalizes the CA to support varying numbers of values per parameter.
- A t-tuple is a t-length subset of parameter values that must appear at least once across all test cases.

Mathematically, t-way testing can be described using Orthogonal Arrays (OAs) [8], where each subarray contains all ordered t-sized subsets of parameter values. OAs are typically represented using parameters such as N (number of test cases), k (number of parameters), v (number of values per parameter), and t (interaction strength). However, a key limitation of OAs is their requirement for uniform value counts across all parameters.

To address this issue, Covering Arrays (CAs) [9] were introduced. CA, denoted as CA(N; t, k, v), guarantees that every t-combination of values is covered at least once in a test suite. For example, CA(N; 2, 5, 4) covers all pairwise interactions among five parameters, each with four possible values. While CAs offer improved flexibility over OAs, they still assume a uniform number of values per parameter. For real-world systems where parameters often have varying values, the Mixed Covering Array (MCA) provides a more general solution. An MCA is defined as MCA(N, t, C), where C represents the configuration of parameter-value groupings, such as ($v_1^{k_1}$, $v_2^{k_2}$, ..., $v_n^{k_n}$). For instance, MCA(7, 3, 4^6, 2^4) refers to a 3-way interaction test suite with 7 test cases, covering 6 parameters with 6 values each and 4 parameters with 2 values each.

To illustrate the principles of t-way testing, consider a hypothetical configuration scenario involving a mobile e-commerce application where the system consists of four key input parameters: Payment Method, Login Authentication, Delivery Option, and Notification Type. The Payment Method parameter supports three values of Credit Card (C), Digital Wallet (W), and Cash on Delivery (D) while the other three parameters each support two values of Password (P) or Biometric (B) for Login Authentication, Standard (S) or Express (E) for Delivery Option, and Email (E) or SMS (S) for Notification Type, as shown in Table I.

TABLE I. INPUT PARAMETERS AND CORRESPONDING VALUES

Payment	Auth	Delivery	Notification
Credit Card (C)	Password (P)	Standard (S)	Email (E)
Digital Wallet (W)	Biometric (B)	Express (E)	SMS (S)
Cash on Delivery (D)			

Assuming a 2-way interaction strength (t = 2), the goal is to construct a test suite in which all possible pairs of values across all parameter combinations are covered at least once. Exhaustively testing every possible configuration would require 3 × 2 × 2 × 2 = 24 test cases. However, by applying combinatorial optimization, the number of required test cases can be significantly reduced without sacrificing 2-way coverage. Based on the configuration above, the following test suite of 12 test cases achieves complete 2-way coverage with a significantly reduced number of tests compared to exhaustive enumeration, as shown in Table II.

TABLE II. 2-WAY COVERING TEST SUITE

Test Case	Payment	Auth	Delivery	Notification
TC1	C	P	S	E
TC2	W	B	E	S
TC3	D	P	E	S
TC4	C	B	S	S
TC5	W	P	S	E
TC6	D	B	S	E
TC7	C	P	E	E
TC8	W	B	S	S
TC9	D	P	S	S
TC10	C	B	E	S
TC11	W	P	E	S
TC12	D	B	E	E

Based on the configuration of the mobile e-commerce application above, the system can be formally represented as a Mixed Covering Array as follows:

$$MCA\ (12;2;3^1 2^3)$$

This denotes a test suite containing 12 test cases that achieves full pairwise coverage across one 3-valued parameter and three 2-valued parameters. The reduced test count demonstrates the efficiency of combinatorial testing in minimizing redundant combinations while preserving interaction coverage. This test suite ensures that every pairwise combination of parameter values appears in at least one test case. For example, the pair (Wallet, Biometric) is covered in TC2 and TC8, while (Credit Card, Express) appears in TC7 and TC10. This approach demonstrates how combinatorial testing can yield significant reductions in testing effort while maintaining robust fault detection capabilities.

C. Metaheuristic Algorithm

Metaheuristic algorithms are a diverse class of nature-inspired, derivative-free optimization techniques that excel in solving complex real-world problems where traditional analytical methods fall short [10]. These algorithms are generally classified into four main categories of swarm-based,

evolutionary, physics-based, and human-based approaches [11]. Evolutionary algorithms (EAs), for example, are inspired by Darwinian natural selection, with genetic algorithms being a well-known instance that simulates biological evolution [12].

Over the past two decades, metaheuristic algorithms have been extensively adapted, modified, and integrated with other intelligent techniques for various engineering applications, ranging from Proton Exchange Membrane Fuel Cell (PEMFC) design [13] to software testing, including their role in Test Case Prioritization [14][15]. A survey of 14 metaheuristic algorithms introduced between 2000 and 2020 [16] highlights the research trends, the hybridization of metaheuristics, advancements in parallel metaheuristics, unresolved challenges, and emerging research opportunities of metaheuristic algorithms.

Metaheuristics are particularly well-suited for problems involving complex constraints or mixed-integer variables [17], where solutions are refined within predefined limits and constraints are effectively managed to transform constrained problems into unconstrained ones. While challenges remain in fully understanding their convergence properties and guaranteeing global optimality across diverse problem domains, ongoing research continues to enhance their efficiency and reliability. As advancements in metaheuristic design progress, these algorithms are becoming increasingly adaptable and robust, reinforcing their potential as powerful optimization tools.

Metaheuristics serve as a powerful approach for tackling optimization problems that are too complex or large to be solved precisely. They are widely applicable across various optimization challenges, including software testing, an essential field in IT that encompasses diverse testing strategies, methodologies, and techniques, including metaheuristic-based approaches [18]. By providing a flexible and efficient means of identifying high quality, near optimal solutions, metaheuristics prove valuable in scenarios, where traditional optimization methods may be impractical or ineffective.

The key contributions of this study are threefold. First, it presents a comprehensive overview of CT techniques and the Grey Wolf Algorithm (GWOA), outlining their strengths and limitations. Second, it provides a detailed explanation of the proposed approach that integrates CT with GWOA, emphasizing its key features and advantages. Third, the effectiveness of the proposed approach is evaluated by comparing its performance with pure computation CT techniques to assess its efficiency under various configurations.

D. Grey Wolf Optimizer Algorithm

Grey Wolf Optimizer Algorithm (GWOA) is an optimization technique inspired by the social structure and hunting strategies of grey wolves. In this method, wolves are organized into four hierarchical levels of alpha, beta, delta, and omega, reflecting a ranking from the strongest to the weakest candidate solutions. The hunting process is mainly directed by the top three wolves of alpha, beta, and delta, while the rest adjust their positions based on these leaders [19].

To utilize GWOA, the input data must first be properly defined and formatted, typically as an array or matrix. The optimization begins with a randomly generated initial population of grey wolves, each representing a potential solution within the search space. These wolves are ranked according to their fitness scores, with the three best classified as alpha (α), beta (β), and delta (δ), while the others, referred to as omega (ω), update their positions in response to the movements of the leading wolves. The key phases of the hunting process of tracking, encircling, and attacking prey are represented mathematically. After identifying the prey, the grey wolves begin to encircle it, which is mathematically modeled using the following equations:

$$\vec{D} = \left|\vec{C} \times \overrightarrow{X_p}(t) - \vec{X}(t)\right| \quad (1)$$

$$\vec{X}(t+1) = \overrightarrow{X_p}(t) - \vec{A} \times \vec{D} \quad (2)$$

Eq. (1) models the step size taken by a wolf towards one of the leading wolves (i.e., α, β, or δ), and Eq. (2) defines the final updated position of the omega wolf. Here, t is the t[th] iteration, $\overrightarrow{X_p}(t)$ are the position vectors of the prey, and the coefficient vector of $\vec{A}$ and $\vec{C}$ calculated as:

$$\vec{A} = 2\vec{a} \times \overrightarrow{r_1} - \vec{a} \quad (3)$$

$$\vec{C} = 2 \times \overrightarrow{r_2} \quad (4)$$

To simulate encircling behavior, the vectors $\vec{A}$ and $\vec{C}$ are computed using Eq. (3) and Eq. (4), where $\vec{a}$ decreases linearly from 2 to 0, and $\overrightarrow{r_1}$ and $\overrightarrow{r_2}$ are random vectors in [0, 1]. $\vec{A}$ controls step size and direction, while $\vec{C}$ adjusts the influence of the leader's position, balancing exploration and exploitation.

To simulate hunting behavior, it is assumed that the α, β, and δ wolves possess superior knowledge about the prey's probable location. Therefore, their positions are treated as the best solutions discovered so far, and the remaining wolves (i adjust their positions based on those of the leading wolves. This adaptive behavior is mathematically expressed through the following equations:

$$\vec{D_\alpha} = |\vec{C_1} \times \vec{X_\alpha} - \vec{X}|, \vec{D_\beta} = |\vec{C_2} \times \vec{X_\beta} - \vec{X}|, \vec{D_\delta} = |\vec{C_3} \times \vec{X_\delta} - \vec{X}| \quad (5)$$

$$\vec{X_1} = \vec{X_\alpha} - \vec{A_1} \times \vec{D_\alpha}, \vec{X_2} = \vec{X_\beta} - \vec{A_2} \times \vec{D_\beta}, \vec{X_3} = \vec{X_\delta} - \vec{A_3} \times \vec{D_\delta} \quad (6)$$

$$\vec{X}(t+1) = (\vec{X_1} + \vec{X_2} + \vec{X_3})/3 \quad (7)$$

In this formulation, $\vec{X_\alpha}$, $\vec{X_\beta}$, and $\vec{X_\delta}$ represent the best positions identified by the α, β, and δ wolves, respectively. The vectors $\vec{C_1}$, $\vec{C_2}$, and $\vec{C_3}$ are randomly generated coefficients, while $\vec{X}$ denotes the current position of the search agent under evaluation. Likewise, $\vec{A_1}$, $\vec{A_2}$, and $\vec{A_3}$ are randomly distributed coefficient vectors, and t represents the current iteration number. Eq. (5) to Eq. (7) describe how the position of each search agent is updated based on the influence of the leading wolves.

The fundamental principle of the GWO algorithm is that omega wolves adjust their positions during the search process according to the locations of the α, β, and δ wolves, thereby simulating the surrounding and attacking of prey. This strategy

allows omegas to diverge from one another while collectively converging towards the prey. When |A|<1, convergence towards prey is emphasized, promoting exploitation. However, this may lead to premature convergence and local optima stagnation. To mitigate this, values of $\vec{A}$ greater than 1 or less than −1 are introduced to encourage divergence from the prey and enhance global exploration in search of better solutions. Additionally, the vector $\vec{C}$ contributes to exploration behavior. When $\vec{C}$>1, the influence of the prey on the search agent is amplified. Conversely, when $\vec{C}$ <1, this influence is reduced, thereby modifying the distance calculation in Equation (1). It is important to note that the parameter $\vec{A}$ decreases linearly throughout the optimization process to promote exploitation in later iterations.

In contrast, $\vec{C}$ remains randomly distributed throughout the process, balancing exploration and exploitation. This dynamic adjustment of $\vec{A}$ and $\vec{C}$ has proven to be an effective strategy for avoiding stagnation in local optima. The flowchart of the GWO algorithm (Algorithm 1) is presented in Fig. 1.

Algorithm 1: GWOA Algorithm

Input: Objective function f(x)

Output: X_alpha: Best-found solution

1. Initialize the grey wolf population X = {X_1, X_2, ..., X_N} randomly
2. Evaluate the fitness of each search agent using f(x)
3. Identify the best three solutions as Alpha (X_alpha), (X_beta), and Delta (X_delta)
4. Set iteration counter t = 0
5. **while** t < MaxIter do
6. a = 2 - 2 * (t / MaxIter)
7. **for** each wolf X_i do
8. **for** each dimension d = 1 to D do
9. Generate r_1, $r_2 \in [0, 1]$
10. A = 2 * a * r_1 - a
11. C = 2 * r_2
12. D_alpha = |C * X_alpha − X_i|
13. D_beta = |C * X_beta − X_i|
14. D_delta = |C * X_delta − X_i|
15. X_1 = X_alpha − A * D_alpha
16. X_2 = X_beta − A * D_beta
17. X_3 = X_delta − A * D_delta
18. X_i = (X_1 + X_2 + X_3) / 3
19. end for
20. end for
21. Evaluate new fitness values
22. Update X_alpha, X_beta, X_delta
23. t = t + 1
24. end while

1.

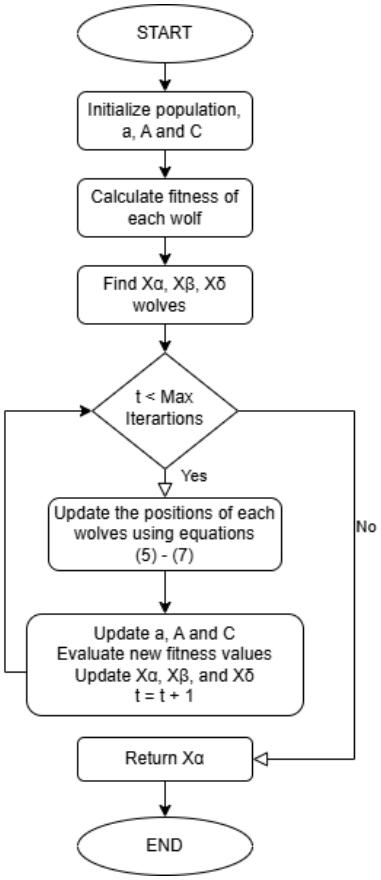

Fig. 1. Flowchart of grey wolf optimization algorithm.

One of the key advantages of GWOA is its ability to balance exploration and exploitation. The algorithm initially explores a wide range of solutions, but as iterations progress, it shifts focus toward refining the best solutions found. Studies have shown that GWOA performs competitively against other optimization algorithms in terms of convergence speed and solution accuracy [20], [21], [22]. Studies have demonstrated that GWOA is effective in cancer detection [23], scheduling [24], and engineering design [25], among other fields [26]. For instance, research has shown that GWOA improved the decision tree model through hyperparameter optimization for diabetes classification [27].

However, like other metaheuristic algorithms, GWOA does not guarantee the ability to pinpoint the exact position of the global optimal solution in every case. The algorithm may converge to a local optimum or fail to find the best solution within the given number of iterations, particularly if the search space is highly complex or if parameter tuning is not optimized. Despite these challenges, GWOA remains a valuable tool for solving problems with large search spaces and intricate constraints due to its ability to balance exploration and exploitation while requiring minimal prior knowledge of the problem domain.

The remainder of this study is structured as follows: Section II presents related work on combinatorial testing and the application of metaheuristic algorithms in test generation. Section III describes the proposed Combinatorial Grey Wolf

Optimizer Algorithm, including its design, parameters, and workflow. Section IV discusses the experimental setup, results, and performance analysis of CGWOA compared to existing combinatorial testing methods. Finally, Section V concludes the study and outlines potential directions for future research.

II. Related Work

In existing research, various heuristic search algorithms have been proposed to guide the process of Combinatorial Testing (CT) and reduce its cost. These algorithms aim to enhance the efficiency of CT by prioritizing the most relevant combinations of input values and parameters. Grey Wolf Optimization Algorithm (GWOA) has been successfully applied to various optimization problems as it has demonstrated strong exploration-exploitation capabilities. Given these characteristics, GWOA has the potential to significantly improve the efficiency of CT by optimizing test suite generation. The following section reviews related work on CT and GWOA, focusing on previous efforts to integrate metaheuristic optimization techniques with CT. The key contributions and limitations of these approaches are discussed, highlighting their relevance to the proposed study. Additionally, an overview of the current state of CT and GWOA research is provided, along with the main research gaps this work aims to address.

Several studies have compared GWOA with other metaheuristic algorithms, highlighting its effectiveness in various optimization tasks. For instance, Pace demonstrated that GWOA outperforms other algorithms in optimizing Transient Electromagnetic (TDEM) data [22]. Similarly, a 2024 study showed that GWOA delivers superior results in optimizing battery performance and lifespan compared to alternative approaches [21]. Due to its strong foundation, GWOA has been the basis for numerous enhancements for optimization problems. In 2022, the Cross-Dimensional Coordination Grey Wolf Optimizer was introduced, outperforming 16 well-known algorithms in benchmark functions and engineering optimization tasks [28]. More recently, in 2025, the Memory-Based Grey Wolf Optimizer (mGWO) was developed, incorporating the personal best history of the wolves, crossover, and greedy selection to improve search efficiency, solution accuracy, and convergence rate [29]. Beyond direct enhancements, researchers have also explored hybridizing GWOA with other optimization techniques to further improve its performance. Ahmad et al. combined GWOA with the Scout Bee Operator function from the Artificial Bee Colony algorithm, achieving superior results compared to multiple other algorithms [30]. Additionally, a 2022 study explored the integration of GWOA with Whale Optimization Algorithm and Harmony Search [31] to optimize combinatorial testing, successfully reducing the size of combinatorial test suites.

Metaheuristic algorithms have been extensively used in CT to optimize test case generation and selection. Algorithms such as Genetic Algorithms (GA), Simulated Annealing (SA), Particle Swarm Optimization (PSO), Harmony Search (HS), and many more have demonstrated their effectiveness in minimizing test suites while maintaining high t-way coverage [32]. GWOA has also been investigated for its application in CT. Prior research has explored various methods of integrating GWOA with CT, including its use in optimizing test suite generation and improving the overall efficiency of the testing process.

However, further research is needed to fully harness the potential of GWOA in CT and to explore additional optimization strategies for improving test effectiveness. A 2024 study explores the use of dynamic TWGH for scalable client-server test suite generation [33]. Another study introduces SCHOP, an approach that enhances the Harmony Search algorithm by integrating seeding and constraint support while employing a one-parameter-at-a-time strategy [34]. Additionally, 2024 saw the introduction of the Binary Growth Optimizer (BGO), a modified version of the Growth Optimizer algorithm, specifically adapted for binary optimization to address the set-covering problem [35]. Other metaheuristics approaches include ROBDD-IPSO (Improved Particle Swarm Optimization) [36], BM (Beautiful Mind) [37], Wingsuit Flying Search Optimization Algorithm (WFSO) [38], Arquitetura Multi-Agente para Metaheurísticas (AMAM) [39], Random-Key Greedy Randomized Adaptive Search Procedure (RK-GRASP) [40], and Pairwise Test Case Generation in Harmony Search Algorithm with Seeding, Constraint Mechanism (PHOSC) [41], Hybridization of Whale Optimization Algorithm and Long Short Term Memory (WOA-LTSM) [42] and Enhanced Harmony Search (EHS) algorithm [43]. Table III summarizes the findings of the recent study.

TABLE III. Recent Study Summary

Study	Baseline Method	Dataset/Project	Evaluation Criteria
[33]	Dynamic TWGH	Client-Server System	Test Suite Size
[34]	SCHOP	Bank Islam Online Login System.	Test Suite Size
[35]	BGO	Set-Covering Problem	Test Suite Size
[36]	ROBDD-IPSO	Real-Time Boiler System	Benchmark Functions
[37]	BM	Benchmark Covering Array	Test Suite Size
[38]	WFSO	Synthetic Test Suites	Test Suite Size
[39]	AMAM	Vehicle Routing Problem with Time Windows	Test Suite Size
[40]	RK-GRASP	Tree Of Hubs Location Problem	Test Suite Size
[41]	PHOSC	UBA Mobile-App login screen	Test Cases
[42]	WOA-LTSM	Kaggle Dataset	Test Suite Size
[43]	EHS	Benchmark Covering Array	Test Suite Size

III. Proposed CGWOA

Failures caused by system interactions can be effectively detected through combinatorial testing, a powerful technique for ensuring software reliability. Traditional methods often struggle to identify optimal solutions for complex optimization problems, especially when dealing with large input spaces. To address this challenge, swarm intelligence and population-based algorithms have been recommended to be applied to optimize test suite generation. Metaheuristic search algorithms, such as the Grey Wolf Optimizer (GWO), offer an effective approach for generating compact and efficient t-way test suites.

The T-tuple table is a key component in applying combinatorial testing (CT) with the Grey Wolf Optimizer Algorithm (GWOA) in this study, ensuring that all necessary t-wise interactions among input parameters are covered. It is initially populated with all possible T-tuples, where each tuple represents a combination of parameter values that must be tested. During the optimization process, test cases are evaluated based on how many uncovered T-tuples they contain, and selected test cases remove these tuples from the table once covered. This iterative process continues until all T-tuples have been accounted for. The T-tuple table thus serves as a guide, preventing redundant test cases and ensuring an efficient balance between test reduction and interaction testing.

The proposed CGWOA method is influenced by several key parameters. The population size affects diversity and search breadth as larger sizes increase exploration but also computation time. The parameter a, which linearly decreases from 2 to 0, governs the balance between exploration and exploitation over time, guiding the wolves from broad searching to focused refinement. The random coefficients r_1 and r_2 introduce stochastic behavior that prevents premature convergence and enhances global search ability. Algorithm 2 presents the CGWOA algorithm.

Algorithm 2: CGWOA Algorithm

Input: Parameters P with their respective values

Output: Test suite covering all t-way combinations

1. Generate all t-way parameter combinations from P
2. Generate all value combinations for each t-way parameter combination
3. Store all the parameter combinations and value combinations pairs in T-tuple table
4. Initialize test suite T
5. **while** T-tuple_table is not empty do
6. Initialize grey wolf population $X = \{X_1, X_2, ..., XN\}$ as random test cases
7. Evaluate fitness for each wolf. X_i based on covered t-tuples
8. Identify the Alpha (Xα), Beta (Xβ), Delta (Xδ) wolves
9. Set iter = 0
10. **while** iter < MaxIter do
11. a = 2 − 2 * (iter / MaxIter)
12. **for** each wolf X_i do
14. Generate $r_1, r_2 \in [0, 1]$
15. A = 2 * a * r_1 − a
16. C = 2 * r_2
17. D_α = |C * X_α[d] − X_i[d]|
18. D_β = |C * X_β[d] − X_i[d]|
19. D_δ = |C * X_δ[d] − X_i[d]|
20. X_1 = Xα[d] − A * Dα
21. X_2 = Xβ[d] − A * Dβ
22. X_3 = Xδ[d] − A * Dδ
23. X_i[d] = round((X_1 + X_2 + X_3) / 3)
24. end for
25. Evaluate new fitness of X_i
26. Update Xα, Xβ, Xδ based on current fitness
27. iter = iter + 1
28. end while
29. Add Xα to the test suite T
30. Remove all covered tuples of Xα from T-tuple_table
31. end while

To begin the CGWOA process, the input parameters and their possible values are first defined. These parameters, which can be variables such as numbers or categorical values, are typically structured into dictionaries or arrays. Then, all t-strength interactions of t-tuples between the parameters are generated and stored in a tuple table. At the start of the optimization process, a population of random test cases of wolves is initialized. Each test case represents a possible solution and consists of randomly selected values for each parameter. The test cases are then evaluated to determine how many t-tuples they cover. The fitness function is calculated based on how much of the test case covers the required T-tuples. The test cases are then ranked based on their fitness, with the top three solutions being designated as alpha, beta, and delta wolves.

The main optimization process proceeds in iterations. For each iteration, the algorithm adjusts the position of each wolf in the population. This is done using the standard GWOA equations. These equations involve randomly generated coefficients, and a linearly decreasing parameter named 'a' which helps control the balance between exploration and exploitation of the search space. The algorithm computes distances from each wolf to the positions of the Alpha, Beta, and Delta wolves and uses these distances to calculate three potential new positions. These are averaged to form a new candidate position for the wolf. If the test case at the new position covers more T-tuples than the current one, the wolf is moved to that new position. Once all wolves have been updated and evaluated, the test case corresponding to the Alpha wolf is selected. If this test case contributes any new t-tuples that are not yet covered, it is added to the final test suite, and those t-tuples are removed from the remaining table. The algorithm continues this process until all T-tuples have been covered. The top-performing test case in each iteration is selected for inclusion in the final test suite, and the t-tuples it addresses are eliminated from the global tuple set to avoid duplication. This cycle continues until either all T-tuples are covered, or the maximum iteration limit is reached. In brief, the algorithm starts by generating a random population of test cases and evaluating their fitness based on uncovered t-tuples. Test cases with higher coverage are added to the suite, while others are refined through iterative updates using the GWOA mechanism. By guiding candidate solutions toward the best-performing wolves, the algorithm gradually achieves full t-tuple coverage. Fig. 2 presents the CGWOA workflow step-by-step.

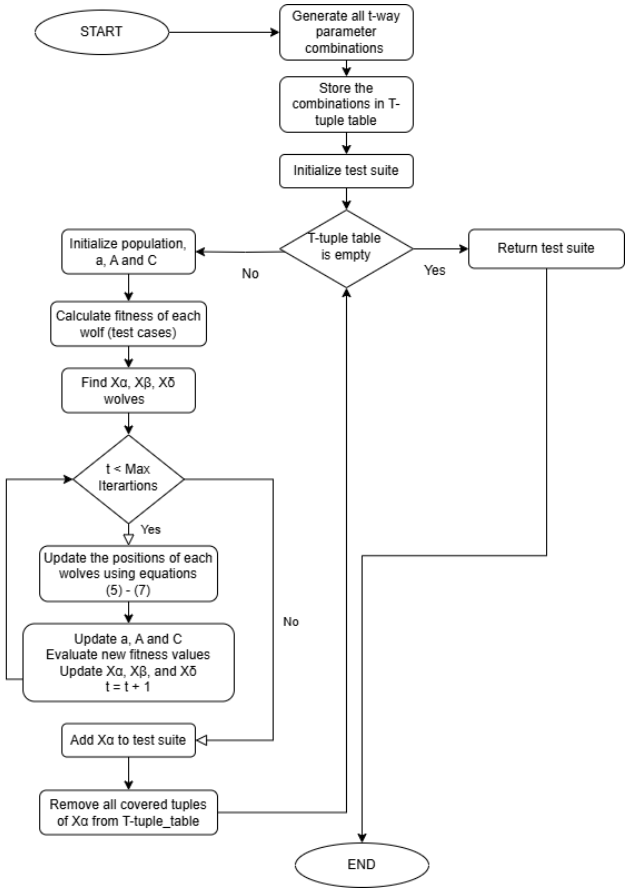

Fig. 2. Flowchart of combinational grey wolf optimization algorithm.

IV. Results and Discussions

This section presents the performance of the proposed CGWOA approach through test suite generation on real-world software configurations, benchmarking against pure computation combinatorial testing methods, and statistical evaluation to assess the significance of observed differences in performance.

A. Test Suite Generation on Real World Software Configuration

To demonstrate the application of the proposed Combinatorial Grey Wolf Optimizer Algorithm (CGWOA), a case study was conducted using real-world parameter configurations derived from the Microsoft Word Page Setup Dialog Box. The CGWOA approach was applied by iteratively selecting test cases that eliminated the largest number of uncovered t-tuples from a dynamic t-tuple table. As each test case was added to the suite, the corresponding covered interactions were removed from the table, and the process continued until full 2-way coverage was achieved. Fig. 3 illustrates the Microsoft Word Page Setup dialog [44], which includes six key parameters relevant to document formatting of Section Start (A), Header and Footer (B), From Edge Header (C), From Edge Footer (D), Vertical alignment (E), and Apply To (F). The full range of configurable options is summarized in Table IV.

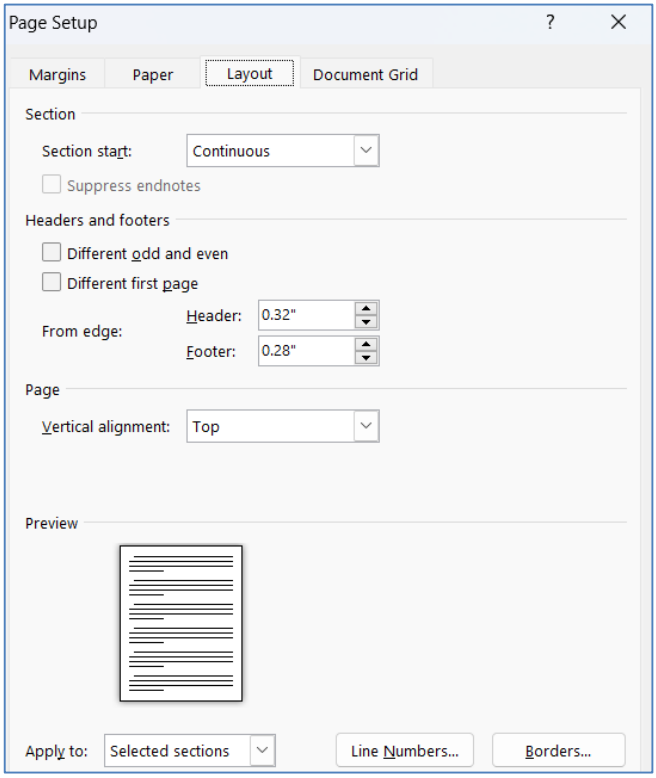

Fig. 3. MS Word page setup dialog box.

TABLE IV. MS Word Page Setup Dialog Box Parameters Value

Input Values					
Section Start (A)	**Headers and Footers (B)**	**From Edge Header (C)**	**From Edge Footer (D)**	**Vertical Alignment (E)**	**Apply To (F)**
Continuous	Different Odd and Even	-10	-10	Top	Selected Sections
New Column	Different First Page	0	0	Center	Whole Document
New Page		10	10	Justified	
Even Page		*null*	*null*	Bottom	
Odd Page		NaN	NaN		

Before applying CGWOA, the total number of possible test case combinations was calculated to be 2000, based on the following computation:

A = {Continuous, new column, new page, even page, odd page}

B = {Different odd and even, Different first page}

C = $\{-10, 10, 0, null, NaN\}$}

D = $\{-10, 10, 0, null, NaN\}$

E = {Top, Center, Justified, Bottom}

F = {Selected Sections, Whole Document}

$$\therefore |test\ cases| = |A| \times |B| \times |C| \times |D| \times |E| \times |F|$$
$$= 5 \times 2 \times 5 \times 5 \times 4 \times 2$$
$$= 2000$$

However, executing all 2000 test cases is computationally expensive and inefficient, necessitating an optimization strategy for test suite reduction while maintaining essential coverage. To achieve this objective, a t-tuple table was constructed to facilitate pairwise combinatorial testing. This table systematically enumerates all T-tuples that consist of all possible combinations of two parameters and their values that need to be included in the final test suite. The optimization process begins by generating the full set of t-tuples and subsequently selecting test cases that maximize tuple coverage while minimizing redundancy.

The original exhaustive test set contained numerous test cases with overlapping t-tuples, leading to significant redundancy. CGWOA addressed this by using the t-tuple table as a structured mechanism to track essential interactions and guide the search toward high-utility test cases. Table V illustrates a substantial reduction of the test suite from 2000 to a mere 40 cases, a 98% reduction with all coverage. The algorithm's ability to continuously update its coverage goals and remove redundant combinations ensured that each selected test case contributed meaningfully to the overall efficiency and completeness of the suite.

TABLE V. PERCENTAGE OF TEST CASES REDUCTION

Condition	Test Cases	Percentage Reduction
Exhaustive Combination	2000	Full Test Cases Count
With CGWOA	40	98%

The results indicate that GWOA provides a highly efficient method for combinatorial test case selection, significantly reducing the test suite size while preserving essential coverage. The integration of the t-tuple table ensures that all required 2-way combinations are methodically incorporated, while the optimization process effectively minimizes unnecessary test case execution. This efficiency underscores CGWOA's direct applicability in software quality assurance.

These promising results highlight the potential of CGWOA in generating compact and efficient test suites while maintaining full interaction coverage. However, to fully evaluate its effectiveness, it is essential to compare CGWOA's performance against other well-established combinatorial testing strategies. In the next section, we assess the performance of CGWOA in comparison with techniques such as Jenny, IPOG, IPOG-D, PICT, and TConfig. By examining these strategies across various test configurations and strengths, we can better understand the relative advantages and limitations of CGWOA in the context of combinatorial test case generation. This comparison will provide a comprehensive view of how CGWOA performs in diverse scenarios and its potential for improving testing efficiency in real-world applications.

B. CGWOA Benchmarking Against Pure Computation Combinatorial Testing Techniques

To assess the performance of the proposed CGWOA, a comparative evaluation was conducted against several established pure computation combinatorial test generation techniques, namely Jenny [45], IPOG [46], IPOG-D [47], TConfig [48], and PICT [49]. Jenny is a fast, backtracking-based tool for generating minimal t-wise test suites, suitable for small to medium configurations. IPOG incrementally constructs t-wise tests by covering interactions in stages, offering a good balance between efficiency and scalability. IPOG-D extends IPOG with a divide-and-conquer strategy, making it more suitable for large systems and higher interaction strengths. TConfig, associated with using mathematical methods to generate covering arrays for efficient t-wise testing, is particularly effective with uniform parameter configurations. PICT, developed by Microsoft, employs a greedy, constraint-solving approach to generate efficient pairwise and n-wise test suites, widely adopted in industrial testing scenarios.

These algorithms were chosen due to their extensive use in real-world applications, including implementation in well-established companies like IBM and integration into widely adopted open-source projects [49][50]. The inclusion of these benchmark methods ensures a comprehensive and balanced comparison, thereby facilitating an objective analysis of CGWOA's capability to construct minimal yet high-coverage test suites.

The experiments were conducted on a laptop running a 64-bit version of Windows 11, featuring an AMD Ryzen 5 processor operating at 3.30 GHz and 8 GB of RAM. The proposed method was developed in Python. Testing was organized into the following three datasets:

- Evaluating the CGWOA approach against existing methods using CA (t, v^7), where the number of parameters is fixed, but the number of values varies. Additionally, interaction strength ranges from 2 to 6.
- Evaluating the CGWOA approach compared to other methods using five different Mixed Covering Array (MCA) configurations.
- Evaluating the CGWOA approach against current techniques using CA (t, 3^p), where the number of parameters varies while the number of values remains constant and the interaction strength t ranges from 2 to 5.

Although reduction percentages were computed for all combinatorial testing techniques, they were uniformly high,

exceeding 99% across all parameter and strength configurations. Due to this saturation effect, the inclusion of detailed reduction statistics was deemed unnecessary, as such values offered limited discriminatory insight into the relative performance of the methods which is why the analysis focuses on absolute test suite sizes, which provide a more meaningful basis for evaluating the effectiveness and efficiency of each technique in practical settings.

Firstly, the performance of CGWOA was assessed across various interactions, with increasing levels of parameter cardinality and benchmarked across configurations of CA (N; t, 3^P). Based on Table VI, for t = 2, CGWOA consistently matched or outperformed the compared tools in terms of minimizing test suite size. It demonstrated particularly stable and competitive performance as the number of parameters increased. The growth in test suite size with increasing P remained modest, indicating good scalability for pairwise testing. When moving to t = 3, CGWOA continued to demonstrate efficient scaling behavior. The growth in test size remained well-controlled, which highlights the optimizer's ability to effectively manage the combinatorial explosion typical at this level.

TABLE VI. TEST SUITE SIZE PERFORMANCE FOR CA(T, 3^P)

t	p	CGWOA	Pure Computation Strategies				
			Jenny	TConfig	PICT	IPOG-D	IPOG
2	3	9	9	10	10	15	9
	4	9	13	10	13	15	9
	5	12	14	14	13	15	15
	6	14	15	15	14	15	15
	7	15	16	15	16	15	15
	8	15	17	17	16	15	15
	9	16	18	17	17	15	15
	10	16	19	17	18	21	15
	11	16	17	20	18	21	17
	12	18	19	20	19	21	21
3	4	31	34	32	34	27	32
	5	39	40	40	43	45	41
	6	45	51	48	48	45	46
	7	50	51	55	51	50	55
	8	53	58	58	59	50	56
	9	57	62	64	63	71	63
4	5	96	109	97	100	162	97
	6	132	140	141	142	162	141
	7	156	169	166	168	226	167
5	6	313	348	305	310	386	305
	7	441	458	477	452	678	466
6	7	975	1089	921	1015	1201	921
	8	1409	1466	1515	1455	1763	1493

As interaction strength increased further to t = 4 and t = 5, the benefits of CGWOA became increasingly evident. The differences were especially noticeable as the number of parameters rose, indicating that CGWOA scaled more efficiently with problem complexity. At the highest interaction strength examined, t = 6, CGWOA continued to show strong performance as it maintained smaller test suite sizes than others, with only a few occasionally matching its output.

TABLE VII. TEST SUITE SIZE PERFORMANCE FOR MCA

Configurations	CGWOA	Jenny	Tconfig	PICT	IPOG-D	IPOG
MCA (N; 4, 3^4 4^5)	444	457	463	NA	NA	499
MCA (N; 4, 5^1 3^8 2^2)	291	303	324	NA	NA	302
MCA (N; 4, 8^2 7^2 6^2 5^2)	4323	4580	4776	NA	NA	4317
MCA (N; 4, 6^5 5^4 3^2)	2475	3033	3273	NA	NA	NA
MCA (N; 4, 10^1 9^1 8^1 7^1 6^1 5^1 4^1 3^1 2^1)	5883	6138	5492	NA	NA	5495

The results from Table VII show that CGWOA consistently achieves adequate test suite sizes across all five Mixed Covering Array configurations. For example, in MCA(N; 4, 3^4 4^5) and MCA(N; 4, 5^1 3^8 2^2), it generates 444 and 291 test cases, outperforming all other methods. Even in more complex scenarios such as MCA(N; 4, 8^2 7^2 6^2 5^2) and MCA(N; 4, 6^5 5^4 3^2), CGWOA maintains superior performance, with smaller test sizes than Jenny, Ipog, and Tconfig. In the largest configuration, MCA(N; 4, 10^1 9^1 8^1 7^1 6^1 5^1 4^1 3^1 2^1), it again produces the most compact suite with 5883 tests. These results highlight CGWOA's efficiency in handling mixed-level parameter sets.

Table VIII presents the performance comparison of CGWOA with established combinatorial test generation tools across configurations of CA(N; t, v^7), where the interaction strength t ranges from 2 to 6 and the number of values per parameter v increases from 2 to 5. While interaction strength influences test suite size, the results show that the increase in v plays a more dominant role in driving complexity.

When t is fixed at 2, all tools generate relatively small test suites, and the differences among them are minor. However, as v increases, CGWOA consistently maintains its efficiency, either matching or outperforming other tools, and demonstrates stable, controlled growth in test suite size. As t increases to 3, the interaction complexity deepens, but again, the rise in v results in the most significant expansion in test suite size. CGWOA continues to provide competitive results, especially at higher values of v, where other tools begin to show sharper increases. At t = 4, the impact of growing v becomes even more pronounced.

It can be observed that the differences between CGWOA and traditional tools are amplified, especially in higher-v scenarios, as can be seen when t reaches 5, the increase in v continues to dominate the growth trend. While all methods experience substantial expansion in test suite size, CGWOA manages to generate significantly leaner suites across all v values. Finally for v = 6, all techniques show growth in test suite size, but CGWOA consistently produces smaller suites across all v levels.

These are visually summarized in Fig. 4, which highlights how test suite size increases with higher values of v.

Overall, CGWOA demonstrates consistent and effective performance across varying degrees of test complexity. Its ability to scale gracefully with increasing interaction strength and parameter diversity makes it a promising approach for efficiently generating minimal yet comprehensive test suites.

The experimental results across all test scenarios consistently demonstrate the effectiveness of the proposed CGWOA approach in generating compact covering arrays. CGWOA outperforms pure computation methods such as Jenny, IPOG, IPOG-D, PICT, and Tconfig in the majority of configurations. It achieves smaller test suite sizes across varying interaction strengths and parameter combinations, indicating its robustness and scalability.

TABLE VIII. TEST SUITE SIZE PERFORMANCE FOR CA(T, v^7)

CA (t, v^7)		CGWOA	Pure Computation Strategies				
t	p		Jenny	TConfig	PICT	IPOG-D	IPOG
2	2	7	8	7	7	8	8
	3	15	15	15	16	15	17
	4	25	28	28	27	32	28
	5	37	37	40	40	45	42
3	2	12	14	16	15	14	19
	3	52	54	55	51	50	57
	4	119	124	112	124	114	208
	5	222	236	239	241	252	275
4	2	30	31	36	32	40	48
	3	159	169	166	168	226	185
	4	493	517	568	529	704	509
	5	1174	1248	`1320	1279	1858	1349
5	2	54	57	56	57	80	128
	3	440	458	477	452	678	608
	4	1830	1938	1792	1933	2816	2560
	5	5480	5895	N/A	5814	9198	8091
6	2	71	87	64	72	96	64
	3	978	1087	921	1015	1201	1281
	4	5629	6127	N/A	5847	5120	4096
	5	21608	23492	N/A	22502	24808	28513

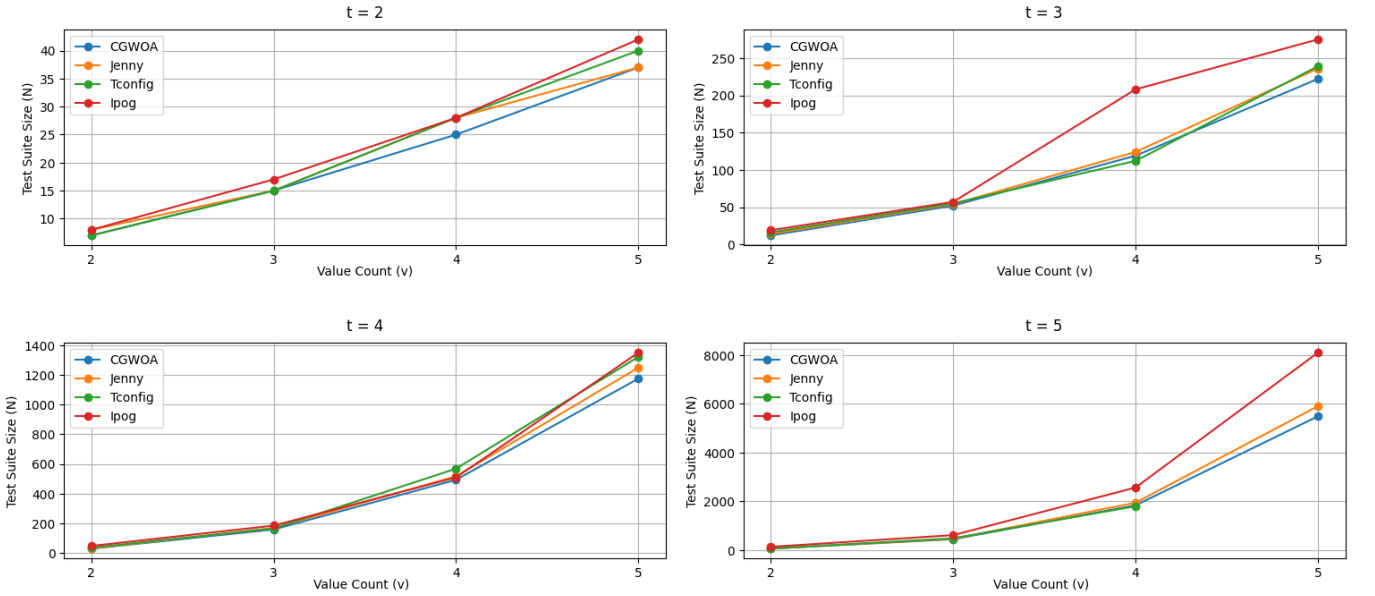

Fig. 4. Line graph of test suite size performance for CA(N;t, v7).

C. Statistical Evaluation

To further evaluate the performance of the proposed CGWOA, a statistical comparison was conducted against the pure computation combinatorial test generation methods. To evaluate whether the differences in test suite sizes between CGWOA and other methods were statistically meaningful, we applied the Wilcoxon signed-rank test. This is a non-parametric statistical test used to compare two related samples. The test was conducted at a 95% confidence level (α = 0.05), making it appropriate for determining whether the observed differences are likely to be due to chance.

The comparison was based on the test suite size data shown in Table VI, which includes multiple configurations of parameter counts and interaction strengths. Because several t-strength comparisons were performed, we used the Holm–Bonferroni method to correct for the increased risk of Type I errors (false positives). The Holm–Bonferroni procedure works by ranking the p-values from smallest to largest and adjusting the significance threshold for each test accordingly. This helps ensure that the conclusions remain statistically valid even when multiple comparisons are made. The adjusted significance level for each test (α Holm) was calculated following the method described in Eq. (8))[51]:

$$\alpha \text{ Holm} \frac{\alpha}{M-i+1} \quad (8)$$

where, α is the initial significance level (0.05), M is the total number of hypotheses tested, and *i* is the rank order of each p-value.

As shown in Table VI, CGWOA outperformed all competing strategies of Jenny, PICT, IPOG-D, TConfig, and IPOG, in the majority of test configurations. Based on this data, the Wilcoxon signed-rank test was applied, and the results are presented in Table IX.

TABLE IX. WILCOXON SIGNED-RANK TEST ANALYSIS

Comparison	Asymp. Sig. (2-tailed)	α Holm	Conclusion
CGWOA vs Jenny	0.000028	0.0100	Reject the null hypothesis
CGWOA vs PICT	0.000127	0.0125	Reject the null hypothesis
CGWOA vs IPOG-D	0.000826	0.0167	Reject the null hypothesis
CGWOA vs TConfig	0.002871	0.0250	Reject the null hypothesis
CGWOA vs IPOG	0.018355	0.0500	Reject the null hypothesis

All comparisons produced p-values below their corresponding Holm-adjusted significance thresholds, leading to the rejection of the null hypothesis in every case. This confirms that the performance differences observed between CGWOA, and the other methods are statistically significant.

This consistent statistical advantage reinforces the empirical findings and substantiates the effectiveness of CGWOA in producing more compact test suites. The results affirm that CGWOA significantly outperforms pure computation algorithms across diverse parameter and strength settings, making it a robust and efficient choice for combinatorial test suite generation.

V. CONCLUSION AND FUTURE WORK

Recent studies have demonstrated that combinatorial testing (CT) can be effectively enhanced using metaheuristic approaches, optimizing the test suite by significantly reducing its size. However, prior research indicates that certain essential test cases may still be omitted in the generated test suite, potentially affecting overall test coverage.

This study introduces the Combinatorial Grey Wolf Optimizer Algorithm (CGWOA) as an alternative metaheuristic approach for combinatorial testing. The results demonstrate that CGWOA effectively optimizes test suite generation while ensuring full t-way coverage, consistently outperforming conventional computation-based methods across a broad range of configurations, including low to high interaction strengths, parameter counts, and value cardinalities. Future research will focus on refining CGWOA, particularly in the context of t-way testing, to improve its applicability across diverse case studies.

Despite its effectiveness, this study's evaluation of CGWOA was limited to a specific software system and parameter set. Future research should expand validation to diverse applications, higher-order interactions, and investigate computational overhead for large-scale systems. Further comparative studies and analysis of parameter tuning will also enhance understanding of CGWOA's broader applicability and performance.

Additionally, further improvements may be explored through techniques such as adaptive parameter tuning, hybridization with other optimization algorithms, and applying multi-task optimization within combinatorial testing environments. By incorporating these enhancements, it is possible to develop a more effective test selection process while ensuring thorough system validation.

ACKNOWLEDGMENT

This research was supported by Ministry of Higher Education (MOHE) through Fundamental Research Grant Scheme (FRGS/1/2024/ICT03/UTHM/01/1).

REFERENCES

[1] D. Kuhn, R. Kacker, and Y. Lei, “Practical Combinatorial Testing,” Oct. 2010, doi: 10.6028/NIST.SP.800-142.

[2] L. Kampel, “Combinatorial Design Theory and Applications for Software Testing,” Thesis, Technische Universität Wien, 2025. doi: 10.34726/hss.2025.131527.

[3] H. Wu, L. Xu, X. Niu, and C. Nie, “Combinatorial testing of RESTful APIs,” in Proceedings of the 44th International Conference on Software Engineering, in ICSE ’22. New York, NY, USA: Association for Computing Machinery, Jul. 2022, pp. 426–437. doi: 10.1145/3510003.3510151.

[4] A. Fontes and G. Gay, “The integration of machine learning into automated test generation: A systematic mapping study,” Softw. Test. Verification Reliab., vol. 33, no. 4, p. e1845, 2023, doi: 10.1002/stvr.1845.

[5] C. M. Tiutin, M.-T. Trifan, and A. Vescan, “Defect Prediction-Based Test Case Prioritization,” Stud. Univ. Babeş-Bolyai Inform., vol. 65, p. 78, Dec. 2020, doi: 10.24193/subbi.2020.2.06.

[6] S. Mirjalili, S. M. Mirjalili, and A. Lewis, “Grey Wolf Optimizer,” Adv. Eng. Softw., vol. 69, pp. 46–61, 2014, doi: https://doi.org/10.1016/j.advengsoft.2013.12.007.

[7] A. A. Al-Sewari and K. Z. Zamli, “An Orchestrated Survey on T-Way Test Case Generation Strategies Based on Optimization Algorithms,” in

The 8th International Conference on Robotic, Vision, Signal Processing & Power Applications, Springer, Singapore, 2014, pp. 255–263. doi: 10.1007/978-981-4585-42-2_30.

[8] I. M. Chakravarti, "Orthogonal and partially balanced arrays and their application in design of experiments," Metrika, vol. 7, no. 1, pp. 231–243, Dec. 1963, doi: 10.1007/BF02613974.

[9] N. J. A. Sloane, "Covering arrays and intersecting codes," J. Comb. Des., vol. 1, no. 1, pp. 51–63, 1993, doi: 10.1002/jcd.3180010106.

[10] S. Ghosh, A. Singh, and S. Kumar, "HPB3C-3PG algorithm: A new hybrid global optimization algorithm and its application to plant classification," Ecol. Inform., vol. 81, p. 102581, Jul. 2024, doi: 10.1016/j.ecoinf.2024.102581.

[11] S. Gopi and P. Mohapatra, "Fast random opposition-based learning Aquila optimization algorithm," Heliyon, vol. 10, no. 4, p. e26187, Feb. 2024, doi: 10.1016/j.heliyon.2024.e26187.

[12] A. Petrowski and S. Ben Hamida, Evolutionary Algorithms. Hoboken, NJ, USA: John Wiley & Sons, 2016, doi: 10.1002/9781119136378

[13] S. Raji, A. Dehnamaki, B. Somee, and M. R. Mahdiani, "A new approach in well placement optimization using metaheuristic algorithms," J. Pet. Sci. Eng., vol. 215, p. 110640, May 2022, doi: 10.1016/j.petrol.2022.110640.

[14] A. Samad, H. B. Mahdin, R. Kazmi, R. Ibrahim, and Z. Baharum, "Multiobjective Test Case Prioritization Using Test Case Effectiveness: Multicriteria Scoring Method," Sci. Program., vol. 2021, no. 1, p. 9988987, 2021, doi: 10.1155/2021/9988987.

[15] M. A. Asyraf, M. Z. . Sahid, and N. . Zainal, "Comparative Analysis of Test Case Prioritization Using Ant Colony Optimization Algorithm and Genetic Algorithm", jscdm, vol. 4, no. 2, pp. 52–58, Oct. 2023, Accessed: Jul. 07, 2025. [Online]. Available: https://publisher.uthm.edu.my/ojs/index.php/jscdm/article/view/13585

[16] T. Dokeroglu, E. Sevinc, T. Kucukyilmaz, and A. Cosar, "A survey on new generation metaheuristic algorithms," Comput. Ind. Eng., vol. 137, p. 106040, Nov. 2019, doi: 10.1016/j.cie.2019.106040.

[17] V. Tomar, M. Bansal, and P. Singh, "Metaheuristic Algorithms for Optimization: A Brief Review," Eng. Proc., vol. 59, no. 1, Art. no. 1, 2024, doi: 10.3390/engproc2023059238.

[18] D. Madhavi, "A White Box Testing Technique in Software Testing : Basis Path Testing," Int. J. Sci. Res. Dev., vol. 2, no. 4, pp. 12–17, Jul. 2016. [Online]. Available: https://www.journal4research.org/Article.php?manuscript=J4RV2I4007

[19] Z. Mustaffa, M. H. Sulaiman, and Y. Yusof, "An Application of Grey Wolf Optimizer for Commodity Price Forecasting," Appl. Mech. Mater., vol. 785, pp. 473–478, 2015, doi: 10.4028/www.scientific.net/AMM.785.473.

[20] R. Ibrahim, "A Comparative Study of Metaheuristic Optimization Algorithms on Distinct Benchmark Functions," J. Soft Comput. Data Min., vol. 6, no. 1, Art. no. 1, Jun. 2025, doi: 10.1002/9781119136378.

[21] M. Camas, A. Coronado, C. Vargas-Salgado, J. Aguila-Leon, and D. Alfonso-Solar, "Optimizing Lithium-Ion Battery Modeling: A Comparative Analysis of PSO and GWO Algorithms," Energies, vol. 17, p. 822, Feb. 2024, doi: 10.3390/en17040822.

[22] F. Pace, A. Raftogianni, and A. Godio, "A Comparative Analysis of Three Computational-Intelligence Metaheuristic Methods for the Optimization of TDEM Data," Pure Appl. Geophys., vol. 179, no. 10, pp. 3727–3749, Oct. 2022, doi: 10.1007/s00024-022-03166-x.

[23] R. Mohakud and R. Dash, "Designing a grey wolf optimization based hyper-parameter optimized convolutional neural network classifier for skin cancer detection," J. King Saud Univ. - Comput. Inf. Sci., vol. 34, no. 8, Part B, pp. 6280–6291, Sep. 2022, doi: 10.1016/j.jksuci.2021.05.012.

[24] Z. Zhu, X. Zhou, D. Cao, and M. Li, "A shuffled cellular evolutionary grey wolf optimizer for flexible job shop scheduling problem with tree-structure job precedence constraints," Appl. Soft Comput., vol. 125, p. 109235, Aug. 2022, doi: 10.1016/j.asoc.2022.109235.

[25] Y. Qiu, X. Yang, and S. Chen, "An improved gray wolf optimization algorithm solving to functional optimization and engineering design problems," Sci. Rep., vol. 14, no. 1, p. 14190, Jun. 2024, doi: 10.1038/s41598-024-64526-2.

[26] X. Yu, N. Jiang, X. Wang, and M. Li, "A hybrid algorithm based on grey wolf optimizer and differential evolution for UAV path planning," Expert Syst. Appl., vol. 215, p. 119327, Apr. 2023, doi: 10.1016/j.eswa.2022.119327.

[27] M. Sam'an, Farikhin, and M. Munsarif, "An improved decision tree model through hyperparameter optimization using a modified gray wolf optimization for diabetes classification," Comput. Methods Biomech. Biomed. Engin., pp. 1–17, Feb. 2025, doi: 10.1080/10255842.2025.2460178.

[28] B. Wang, L. Liu, Y. Li, and M. Khishe, "Robust Grey Wolf Optimizer for Multimodal Optimizations: A Cross-Dimensional Coordination Approach," J. Sci. Comput., vol. 92, no. 3, Sep. 2022, doi: 10.1007/s10915-022-01955-z.

[29] S. Gupta and K. Deep, "A memory-based Grey Wolf Optimizer for global optimization tasks," Appl. Soft Comput., vol. 93, p. 106367, Aug. 2020, doi: 10.1016/j.asoc.2020.106367.

[30] I. Ahmad, F. Qayum, S. U. Rahman, and G. Srivastava, "Using Improved Hybrid Grey Wolf Algorithm Based on Artificial Bee Colony Algorithm Onlooker and Scout Bee Operators for Solving Optimization Problems," Int. J. Comput. Intell. Syst., vol. 17, no. 1, Dec. 2024, doi: 10.1007/S44196-024-00497-6.

[31] H. M. Fadhil, M. N. Abdullah, and M. I. Younis, "TWGH: A Tripartite Whale–Gray Wolf–Harmony Algorithm to Minimize Combinatorial Test Suite Problem," Electronics, vol. 11, no. 18, Art. no. 18, Jan. 2022, doi: 10.3390/electronics11182885.

[32] A. A. Muazu, A. S. Hashim, and A. Sarlan, "Review of Nature Inspired Metaheuristic Algorithm Selection for Combinatorial t-Way Testing," IEEE Access, vol. 10, pp. 27404–27431, 2022, doi: 10.1109/ACCESS.2022.3157400.

[33] H. M. Fadhil, M. N. Abdullah, and M. I. Younis, "Dynamic TWGH: Client-Server Optimization for Scalable Combinatorial Test Suite Generation," BIO Web Conf., vol. 97, p. 00115, 2024, doi: 10.1051/bioconf/20249700115.

[34] A. A. Muazu, A. S. Hashim, U. I. Audi, and U. D. Maiwada, "Refining a One-Parameter-at-a-Time Approach Using Harmony Search for Optimizing Test Suite Size in Combinatorial T-Way Testing," IEEE Access, vol. 12, pp. 137373–137398, 2024, doi: 10.1109/ACCESS.2024.3463953.

[35] D. Leiva, B. Ramos-Tapia, B. Crawford, R. Soto, and F. Cisternas-Caneo, "A Novel Approach to Combinatorial Problems: Binary Growth Optimizer Algorithm," Biomimetics, vol. 9, no. 5, Art. no. 5, May 2024, doi: 10.3390/biomimetics9050283.

[36] S. Li, Y. Song, and Y. Zhang, "Combinatorial Test Case Generation Based on ROBDD and Improved Particle Swarm Optimization Algorithm," Appl. Sci., vol. 14, no. 2, Art. no. 2, Jan. 2024, doi: 10.3390/app14020753.

[37] S. Esfandyari, V. Rafe, E. Pira, and liela Yousofvand, "Beautiful Mind: a meta-heuristic algorithm for generating minimal covering array," Feb. 14, 2024, Research Square. doi: 10.21203/rs.3.rs-3195308/v2.

[38] A. Malaysia et al., "Wingsuit Flying Search Optimization Algorithm Strategy for Combinatorial T-Way Test Suite Generation," International Journal of Advances in Soft Computing and its Applications, vol. 16, no. 3, pp. 272–293, Nov. 2024, doi: 10.15849/ijasca.241130.15.

[39] M. A. L. Silva, J. F. da Silva, S. R. de Souza, and M. J. F. Souza, "A scalability analysis of a multi-agent framework for solving combinatorial optimization via Metaheuristics," Eng. Appl. Artif. Intell., vol. 142, p. 109738, Feb. 2025, doi: 10.1016/j.engappai.2024.109738.

[40] A. A. Chaves, M. G. C. Resende, and R. M. A. Silva, "A random-key GRASP for combinatorial optimization," arXiv preprint, arXiv:2405.18681, May 2024. doi: 10.48550/arXiv.2405.18681

[41] A. A. Muazu, A. S. Hashim, U. D. Maiwada, U. A. Isma'ila, M. M. Yakubu, and M. A. Ibrahim, "Pairwise test case generation with harmony search, one-parameter-at-at-time, seeding, and constraint mechanism integration," Int. J. Electr. Comput. Eng. IJECE, vol. 14, no. 3, Art. no. 3, Jun. 2024, doi: 10.11591/ijece.v14i3.pp3137-3149.

[42] S. Qiao, "Analysis of Test Suite Optimization in Software Engineering Using Whale Algorithm," Jun. 2025, doi: 10.2139/ssrn.5295864.

[43] A. A. Muazu and A. S. Hashim, "Enhancing Harmony Search Metaheuristic Algorithm for Coverage Efficiency, Test Suite Reduction,

and Running Time in Combinatorial Interaction Testing," IEEE Access, vol. 13, pp. 110828–110852, 2025, doi: 10.1109/ACCESS.2025.3583176.

[44] Microsoft Corporation, "Page Setup Dialog Box," Microsoft 365. [Online].]. Available: https://support.microsoft.com/en-us/office, [Accessed: May 5, 2025].

[45] S. Bronnikov, "Jenny: A combinatorial testing tool," GitHub repository, Apr. 6, 2025. [Online]. Available: https://github.com/ligurio/jenny

[46] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG: A General Strategy for T-Way Software Testing," in 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07), Mar. 2007, pp. 549–556. doi: 10.1109/ECBS.2007.47.

[47] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing," Softw. Test. Verification Reliab., vol. 18, no. 3, pp. 125–148, 2008, doi: 10.1002/stvr.381.

[48] A. Williams, "TConfig Java Test Tool," [Online]. Available: http://www.site.uottawa.ca /~awilliam.

[49] Microsoft, "PICT: Pairwise independent combinatorial tool," GitHub repository, 2025. [Online]. Available: https://github.com/microsoft/pict

[50] R. Kuhn, "Practical applications of combinatorial testing," NIST ECUs presentation, 2012. [Online]. Available: https://csrc.nist.gov/CSRC/media/Presentations/Practical-Applications-of-Combinatorial-Testing-Pr/images-media/kuhn-ECU.pdf

[51] S. Holm, "A Simple Sequentially Rejective Multiple Test Procedure," Scand. J. Stat., vol. 6, no. 2, pp. 65–70, 1979, [Online]. Available: https://www.jstor.org/stable/4615733 [Accessed: May 02, 2025].