*Journal of Quality Measurement and Analysis* *JQMA* 21(3) 2025, 21-34
*e-ISSN:* 2600-8602 *http://www.ukm.my/jqma*
*https://doi.org/10.17576/jqma.2103.2025.02*

# INPUT-OUTPUT BASED RELATION COMBINATORIAL TESTING USING WHALE OPTIMIZATION ALGORITHM FOR GENERATING NEAR OPTIMUM NUMBER OF TEST SUITE

(*Pengujian Kombinatorial Perhubungan Berasaskan Input-Output Menggunakan Algoritma Pengoptimuman Paus untuk Menjana Nombor Hampir Optimum Suite Ujian*)

ANJILA J SUALI*, NURAMINAH RAMLI, ROZMIE RAZIF OTHMAN, HASNEEZA LIZA ZAKARIA, ISZAIDY ISMAIL, NOR SHAHIDA MOHD JAMAIL, RIMULJO HENDRADI & NUROL HUSNA CHE ROSE

*ABSTRACT*

Software testing is necessary to verify that the employed system is confident enough to be used in a specific task according to the required demand. Testing every possible interaction parameter to identify and resolve defects or minimize the number of faults might fall under exhaustive testing. Given the impracticality of testing every possible interaction due to time, budget, and resource constraints, combinatorial testing, which is t-way testing, is adopted to cover parameter interactions efficiently. This research focuses on the Input-Output Based Relations (IOR) testing strategy, which optimizes the test suite size by selecting critical parameters and employing "don't care" values for non-essential inputs. Combinatorial testing offers an alternative to overcome the problem. This study proposes a combinatorial testing method utilizing the Whale Optimization Algorithm (WOA). The study compares the performance of WOA with various existing strategies, such as Greedy, Density, TVG, Union, ParaOrder, ReqOrder, ITTDG, AURA, Java Algorithm (CTJ), TTSGA, and AFA. Experimental results indicate that WOA outperforms or matches the effectiveness of these strategies in generating smaller test suites, particularly for higher interaction strengths. The findings demonstrate WOA's competitive advantage in optimizing combinatorial test suites.

*Keywords*: Whale Optimization Algorithm, t-way testing, optimization Input-Output Relationship (IOR), artificial intelligence

*ABSTRAK*

Ujian perisian adalah penting untuk mengesahkan bahawa sistem yang digunakan cukup yakin untuk digunakan dalam tugasan tertentu mengikut permintaan yang diperlukan. Menguji setiap kebarangkalian interaksi parameter untuk mengenal pasti dan menyelesaikan kekurangan atau meminimumkan bilangan kerosakan juga dikenali sebagai ujian menyeluruh. Memandangkan ujian menyeluruh setiap interaksi tidak praktikal disebabkan oleh kekangan masa, belanjawan dan sumber terhad, ujian kombinatoial juga dikenali sebagai ujian *t-way* digunakan untuk mencakupi interaksi parameter dengan efektif. Penyelidikan ini memfokuskan pada strategi ujian Perhubungan Berasaskan Input-Output (IOR), yang mengoptimumkan saiz suite ujian dengan memilih parameter kritikal dan menggunakan nilai "tidak peduli" untuk input yang tidak penting. Ujian kombinatorial menawarkan alternatif untuk mengatasi masalah ujian menyeluruh. Kajian ini mencadangkan kaedah ujian gabungan menggunakan Algoritma Pengoptimuman Paus (WOA). Kajian ini membandingkan prestasi WOA dengan pelbagai strategi sedia ada, seperti Greedy, Density, TVG, Union, ParaOrder, ReqOrder, ITTDG, AURA, Java Algoritma (CTJ), TTSGA, dan AFA. Keputusan eksperimen menunjukkan bahawa WOA mengatasi atau sepadan dengan keberkesanan strategi ini dalam menjana suite ujian yang lebih kecil, terutamanya untuk kekuatan interaksi yang lebih tinggi. Penemuan menunjukkan kelebihan daya saing WOA dalam mengoptimumkan suite ujian gabungan.

*Kata kunci*: Algoritma Pengoptimuman Paus (WOA); ujian *t-way*; Perhubungan Berasaskan Input-Output (IOR); kecerdasan buatan (AI)

## 1. Introduction

Testing every possible interaction of parameters to identify faults in the System Under Test (SUT) is known as exhaustive testing. Testing every possible interaction ensures that the product being tested can be used confidently once all faults have been eliminated (Muazu *et al.* 2022). Unfortunately, due to time, budget, and resource limitations, implementing exhaustive testing is impossible (Leveson 2020; Younis *et al.* 2020). The *t*-way testing is one of the most effective ways to detect faults, other than equivalent class partitioning and boundary value analysis (Jahanbin & Zamani 2018; Ramli *et al.* 2017). Therefore, *t*-way testing is used to create a test suite based on interaction strength that covers parameter interactions at least once while maintaining a near-optimum number of tests.

The *t*-way testing, also known as combinatorial testing, focuses on identifying faults arising from the interactions of two or more input parameters (Othman & Zamli 2011; Zhang *et al.* 2014). The reason is due to the possibility that faults may not just occur within the parameters. Thus, it is crucial to test all possible parameter interactions. Over the past two decades, this technique has been developed to address such faults and produce a series of test cases to complete the test suite, covering interaction strengths such as uniform strength, variable strength, and input-output-based relations (IOR) for the SUT. Previous studies have provided empirical evidence of the effectiveness of metaheuristic search techniques such as Particle Swarm Optimization (PSO), Simulated Annealing (SA), General Variable Strength GVS, In-Parameter-Order-General (IPOG) and the Sine-Cosine algorithm (CSA) in producing smaller test suite sizes (Cohen *et al.* 2003; Lei *et al.* 2008; Mirjalili 2016; Othman *et al.* 2012). These techniques have been proven to support both uniform and variable strength.

Meanwhile, most of the computational algorithm strategies that support IOR compared to the metaheuristic strategies for generating a small number of test suites such as Greedy, Density, TVG, Union, ParaOrder, ReqOrder, Union, ITTDG, AURA, and Java Algorithm (CTJ). Only a few studies have been found about the metaheuristic strategies supporting IOR interactions, such as TTSGA and Adaptive Firefly (Ali *et al.* 2021; Ramli *et al.* 2016; Younis *et al.* 2020). This strategy succeeds in solving a specific set of problems, yet it does not guarantee solving all optimization problems with different types and natures. Thus, the No-Free Lunch (NFL) theorem allows researchers to propose new optimization strategies or improve the current ones for solving subsets of issues in various fields (Mirjalili 2016).

Motivated by the challenges, this paper presents an IOR strategy that utilizes the Whale optimization algorithm (WOA) to generate nearly optimal test suites. Moreover, the Whale Optimization Algorithm (WOA), introduced in its earlier development phase, remains a highly effective tool for t-way testing due to its simplicity, efficiency in generating near-optimal test suites, and its ability to address complex, high-dimensional optimization challenges (Mirjalili & Lewis 2016; Amiriebrahimabadi & Mansouri 2024; Nadimi-Shahraki *et al.* 2023). Thus, this strategy needs to be explored more in the field of testing SUT to optimize the test suite generation, especially for the strength of IOR interactions.

Between the interaction strengths, which are uniform, variable, and IOR interaction, IOR produces the smallest value of the test suite as the tester has pieces of knowledge of the SUT. So, the tester can select the crucial parameters to be tested. Furthermore, by choosing the best "don't care" value, the strategies can strive to generate a better number of test suite sizes (Ali

*et al*. 2021; Muazu *et al.* 2022). WOA strategy adopted as the optimization algorithm by representing the combinatorial optimization problem increases the capability of WOA in generating an optimized combinatorial test suite.

This paper is organized as follows: in section II, this paper discusses related work. Whale Optimization Algorithm is explained in detail in the next section. Section IV presents the results and discussion by comparing the WOA with the existing strategy that applies IOR in their strategies. Finally, section V is the conclusion of this paper.

## 2. Related Work

Over the past two decades, *t*-way testing has become an alternative to overcome exhaustive testing (Younis *et al.* 2020). T-way testing performs testing that focuses on the interaction between parameters to find the faults with high coverage of faults (Ahmed *et al.* 2015). Faults occur within a single parameter and interaction of two or more input parameters. Between the types of interaction strength, IOR interaction only focuses on crucial inputs that produce the particular output of SUT.

The origin of IOR, introduced by P.J. Schroeder, focuses on specific inputs based on particular outputs (Schroeder & Korel 2000). When performing the interaction parameters in the SUT, the IOR differs from the uniform and variable strength. For instance, to represent the IOR scenario as shown in Figure 1, illustrate one program, P, with four inputs, namely *A*, *B*, *C*, and *D*. The programme produces three outputs, *X*, *Y*, and *Z*. The function, *f*, output of *X* involves two input parameters, *A* and *C*, the *f* output of *Y* interacts with input parameters *B*, *C*, and *D*, while *f* output *Z* interacts with input parameters *A* and *D*.

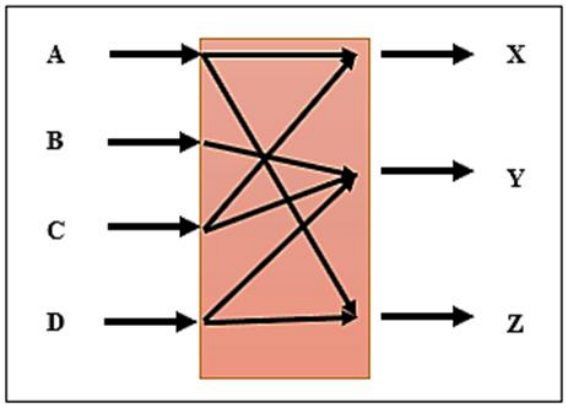


Figure 1: Input-output relationship

Mathematically, IOR can be expressed in covering array notation, IOR(*N*, $V_1^{p1}V_2^{p2}$ ... $V_n^{pn}$, *R*), where *N* is the number of test cases in the suite, $V_1^{p1}V_2^{p2}$ ... $V_n^{pn}$ is the parameters-values configuration for the SUT, and *R* is the number of interaction coverage requirements (Ali *et al.* 2021; Younis *et al.* 2020). In IOR testing, the production number of the test suite can be optimized as the testing ignores the unrelated input value parameters and the selection of the best don't care value (Ramli *et al.* 2016). In contrast, the Union strategy allocates unrelated parameters based on input and output information (Schroeder & Korel 2000). The input value parameters are combined with randomly generated parameter values to produce interactions from an output perspective. The combinations are then merged with duplicate test cases to fulfil the output expectation. Although it has the advantage of generating test suites quickly, the drawback is that the resulting size is massive.

In the meantime, the ParaOrder and RegOrder are variants of the IPOG strategy, which supports the Input-output relationship (IOR) to interaction relationship in t-way testing (Ziyuan *et al.* 2007). ParaOrder initializes the test suite based on a small number of factors, which are expanded by adding new factors in subsequent iterations (Zamli *et al.* 2011). The selection of factors for the next iteration uses the selectFactor function. Meanwhile, the ReqOrder generates the initial test suite based on the coverage requirement from the IOR interaction, arranging interactions between IOR factors based on the most significant combination of factors. If some parameters are irrelevant, they are ignored and left blank.

The Test Vector Generator (TVG) is a software application that utilizes a Graphical User Interface (GUI) to generate t-way tests with uniform strength and IOR (Arshem 2010; Tatale & Prakash 2022). The set comprises three algorithms: T-reduced, Plus-One, and Random set. According to studies, T-reduced performance is often the most ideal. The TVG algorithm can produce test suites that are easier to handle in highly complicated scenarios than other IOR-supported algorithms (Esfandyari & Rafe 2021).

Then, the Density strategy uses IOR interaction to generate optimum test suites for local and global searches (Bryce & Colbourn 2007). It selects contacts with the highest local density from each IOR relationship and then selects those with the highest global density, ensuring complete coverage. The Density program calculates global density for each possible combination.

Ong and Zamli introduced the AURA strategy, which supports uniform strength, variable strength, and IOR interaction (Ong & Zamli 2011). The strategy solves the problem of manually sorting the data test and not having the flexibility to generate test suites. AURA strategy performs the post-processing mapping to convert the symbolic value to actual data automatically.

The ITTDG approach selects a final test case from numerous candidates (Othman *et al.* 2014). The best candidate covers the most uncovered tuples produced by every interaction or IOR the user provides (Othman & Zamli 2011). It uses random and iterative techniques for the selection of test cases. The strategy produces potential candidates only when many values can be covered by the test cases with the most uncovered tuples.

The Jaya algorithm (CTJ) supports an input-output-based relationship (IOR) (Younis *et al.* 2020). The CTJ applied in combinatorial testing employs five phases: data analysis, data mapping, generation of combined input values, generation of test cases, and creation of the final test suite. Data analysis guarantees the accuracy of information input, while data mapping minimizes both time and errors. Input values are combined according to the intensity of their interaction or the link between the input and output. Test cases are developed by randomly selecting input values from each parameter and adjusting them according to the best and worst test scenarios.

The TTSGA is a variant of Ant colony optimization (ACO), an algorithmic technique employed to address optimization issues, which draws inspiration from the foraging behaviour exhibited by ant species (Dorigo *et al.* 2006). The TTSGA supports uniform strength, variable strength, and IOR (Shiba *et al.* 2004; Ramli *et al.* 2016; Chen *et al.* 2009). Ants work by using pheromones to indicate a favourable route for other colony members, establishing the most direct route from the nest to the food source and transporting it back. The TTSGA algorithm uses the IOR technique to enhance the efficiency of producing optimal test suites by incorporating constraints to prevent unnecessary interactions (Ramli *et al.* 2016). ACO is adding random proportional rules to the t-way test suite generator.

In 2021, another strategy, namely The Adaptive Firefly (AFA), also supports IOR, where the strategy selects the best test case with better weightage from the overall process (Ali *et al.* 2021). However, adjustments are needed to adjust the algorithm to suit the IOR interaction. The

Adaptive Firefly strategy can be employed even if the strength is the same for each function output. The key priority is to support limitations and consider the interaction among input parameter values as it impacts program output. The proposed strategy overlooks unrelated values and focuses on supporting the limitations of the IOR process.

## 3. Methodology

The Whale optimization algorithm (WOA) has been introduced by Mirjalili and Lewis (Mirjalili & Lewis 2016). This algorithm basically solves the mathematical optimization problem, specifically benchmark functions. The selected algorithm replicates the hunting movements of humpback whales. In general, the whales belong to the Artiodactyla lineage, including the world's largest mammals. They use a variety of sounds to navigate, communicate, and hunt for food (Norris 2023). These sounds can travel thousands of miles. There are various species of whales, such as Bryde's, humpback, sperm, killer, and many more. Whales can be as big as the school bus for the humpback whale. The humpback whale is easily recognizable by its unique body shape, which includes long pectoral fins and tubercles on the head. These whales are well-known for their hunting food method, the 'bubble-net attacking.' The whales hunt small fish, such as krill, at the water's surface by spinning around and shrinking the circle by creating bubbles along a 9-shaped path to gather closer to the prey. This approach is called the spiral bubble-net feeding approach.

The WOA takes its inspiration from the ingenious hunting technique of humpback whales, known as 'bubble-net attacking,' to trap the prey. The algorithm has two phases: exploitation and exploration. At first, the whale will perform exploitation. In exploitation, several mechanisms exist, such as encircling prey, shrinking mechanism, and spiral updating positions. The whales or agents search for prey by encircling the prey by using Eq. (1). Then, when the target prey, known as the best current candidate, is identified in the encircling process, the other agent will update their position and move towards the best agent that holds the best current solution.

$$\vec{X}(t+1) = \vec{X} * (t) - \vec{A} \,.\, \vec{D} \tag{1}$$

$$\vec{A} = 2\vec{a} \,.\, \vec{r} - \vec{a} \tag{2}$$

$$\vec{C} = 2 \,.\, \vec{r} \tag{3}$$

$$\vec{D} = \left| \vec{C} . \vec{X} * (t) - \vec{X}(t) \right| \tag{4}$$

where *t* represents the current iteration, *X** represents the best solution obtained so far, and *X* is the current solution. Meanwhile, | | is an absolute value, $\vec{A}$ and $\vec{D}$ is the coefficient vector. To calculate the value of $\vec{A}$, $\vec{C}$ and, $\vec{D}$ using Eq. (2), (3), and (4). Value $\vec{a}$ is linearly decreased from 2 to 0 during the iterations, and $\vec{r}$ is a random vector in [1,0].

Meanwhile, the shrinking mechanism applied when $\vec{a}$ value in Eq. (2) decreases, making coefficient vector *A* is reduced significantly. Hence, the search agent's new position can be located within the range of the agent's initial position and the position of the current best agent.

Next is the spiral updating position using Eq. (5) to calculate the distance between agents and the best solution. The equation is trying to mimic the helix-shaped movement of humpback whales. It should be noticed that, in Eq. (5), $\vec{D}'$ is the distance between the current solution and

the best solution, $b$ is the constant value (defining the shape of the logarithmic spiral), and $l$ is a random number in [-1, 1].

$$X\,(t\ +\ 1) =\ D`.ebl.cos\,(2\pi l) +\ -X*(t) \tag{5}$$

WOA can switch between encircling movement and the spiral shape, depending on the probability value, $p$. If the value is less than 0.5, the agent will go to the shrinking encircle mechanism. Meanwhile, if the $p$-value is greater than or equal to 0.5, it will go to the spiral updating position. It can be portrayed as follows, where the value of p [0,1].

$$X(t+1)\begin{cases} Calculation\ of\ 2,\ if\ p < 0.5 \\ Calculation\ of\ 6,\ if\ p\ \geq 0.5 \end{cases} \tag{6}$$

The second phase is the exploration, which is the global search. Unlike in the exploitation phase, while the prey is searching, the update position will be based on the randomly generated solution (i.e. $\overrightarrow{Xrand}$) instead of the best-found solution. The update position equation in searching for the prey phase is given by Eq. (7), where the vector $\vec{D}$ can be found using Eq. (8). In addition, the exploration phase will only be executed when $\vec{A}$ is greater than 1 or less than -1 to force the searching process to move far away from the current solution.

$$\vec{X}(\mathrm{t}\ +\ 1) =\ \overrightarrow{Xrand} - \vec{A}.\vec{D} \tag{7}$$

$$\vec{D} = \left|\ \vec{C}\ .\overrightarrow{Xrand} -\ \vec{X}\ \right| \tag{8}$$

The WOA flowchart is shown in Figure 2. Initially, the algorithm performs random searches and calculates the fitness function for each agent. It then updates each agent's parameters (a, A, C, l, p) in a loop that runs until t < MaxT. For each agent, if p < 0.5 and $|\overrightarrow{A}| < 1$, the agent's position encircles the prey; otherwise, a random agent's position is updated for exploration. Then, if $p > 0.5$, a spiral pattern is used to update the position.

Positions are corrected if they go beyond the search space, and fitness is recalculated. The best agent is updated if a better candidate appears. This process repeats until convergence and the optimal solution is found.

Three main components of the WOA generate the combinatorial test suite generation: the T-Way Tuple List Generator, Interaction Search, and Test Case Generator.

### 3.1. *Implementing WOA for Test Suite Generator*

After describing the WOA algorithm, this section will explain the process of generating test suite size. Three main components of the WOA generate the combinatorial test suite generation: the T-Way Tuple List Generator, interaction search, and test case generator.

#### *3.1.1 T-way tuple lists generator*

This component generates a tuple list table according to the parameter interactions. Generating the test suite is completed when the empty tuple list table is used. There are a few steps to generate the tuple list. Firstly, the algorithm receives the values of parameters, input values, and strengths. Then, the tuple lists are generated according to the interaction strength, $t$. For instance, a SUT has four parameters (i.e., $W$, $X$, $Y$, $Z$). The two parameters hold two input values and four input values (i.e., w1, w2, w3, w4, x1, x2, y1, y2, y3, y4, z1, z2) and the $t = 2$. When $t = 2$, the interaction of parameters for the SUT are $WX$, $WY$, $WZ$, $XY$, $XZ$, and $YZ$. The tuple list

generation according to the interaction strength. The input parameters that are not covered assign a "don't care" (DC) value to complete the interaction. DC means that any value can be assigned. Thus, the total value of the tuple list that needs to be covered in the SUT is 48 according to the involvement of parameter and interaction strength, $t = 2$. The interaction parameter will be stored in the tuple lists table. The value of the tuple list is removed once covered by the test case. The iterations continued until all of the tuple lists were covered. The process is described in Figure 3.

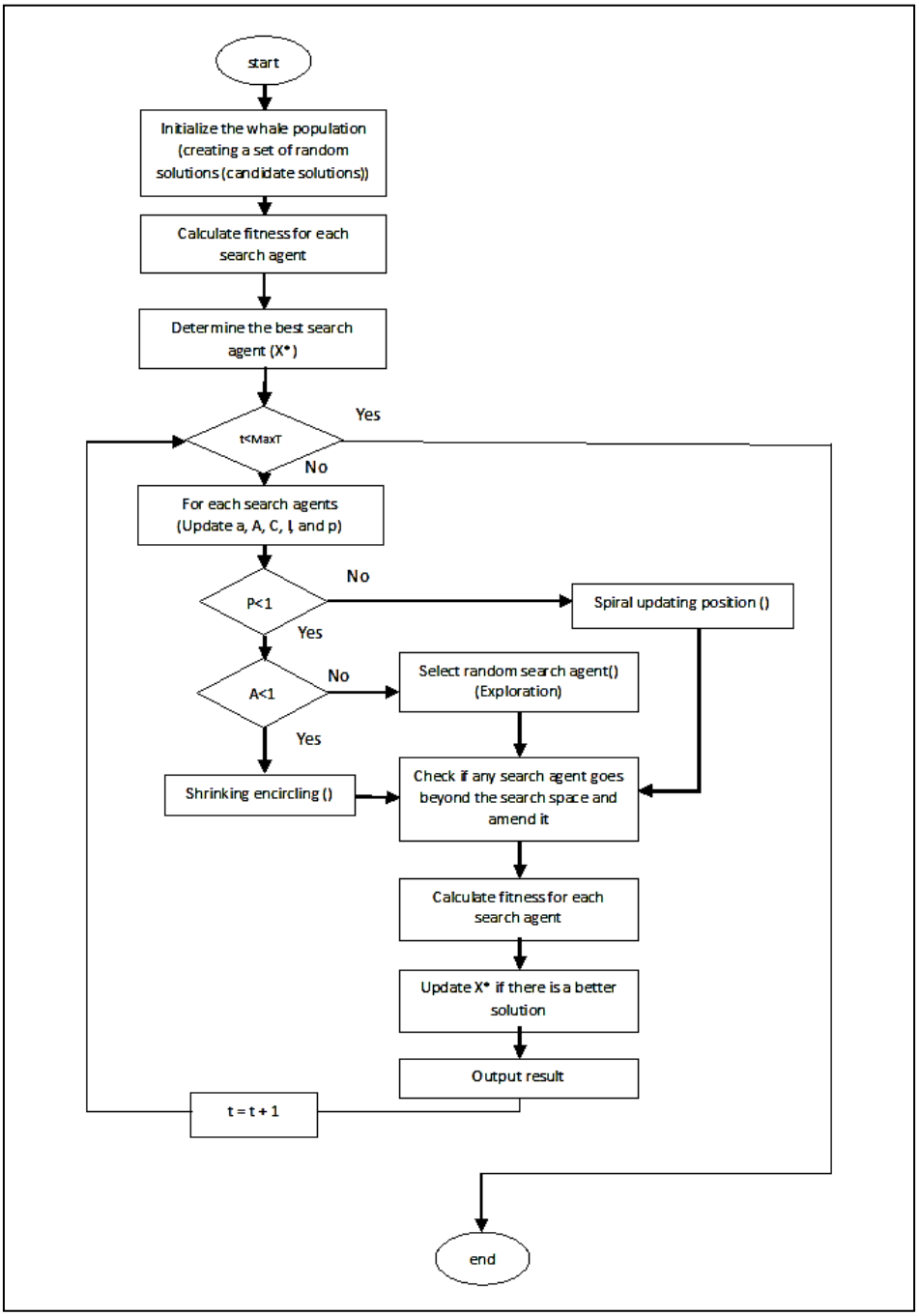


Figure 2: Flowchart of WOA

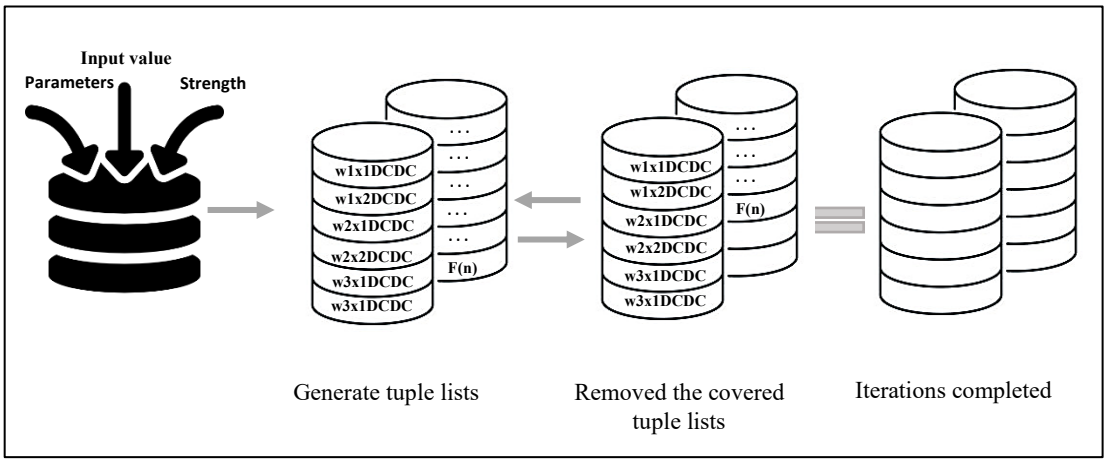


Figure 3: T-way Tuple List Generator process

### *3.2.2. Interaction search*

In this phase, the WOA generates the initial population and calculates the fitness function of the best agent, which is called the test case that carries the most weight (number of tuple lists). The tuple lists taken by the best agent will be compared with the table of tuple lists and removed from the lists. Figure 4 illustrates the position vectors of the encircling prey mechanism searching for potential prey. Once the best agent is found, the other agents update their position and move towards the best agent. While the current agent attempts to find the best solution, it will try to find another potential solution along the way.

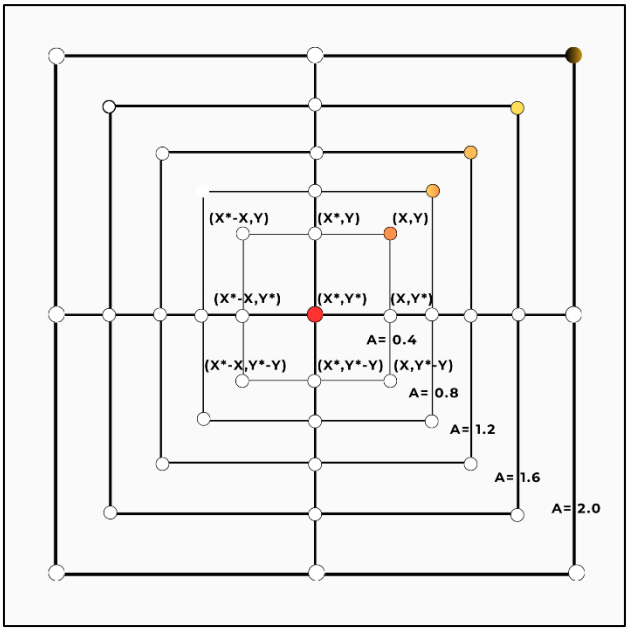


Figure 4: Positions vectors of encircling prey mechanism

The interaction search is based on the probability, *p,* and A variables in the algorithm. The value of *p* has been defined when the value is less than 0.5, and the agent will either go to shrinking encircling or random search (Mirjalili & Lewis 2016). After that, the algorithm will check on the A value, if it is less than 1, it will go to the shrinking encircling, or else it will go to the random search to find another potential solution. The A value will decrease, which will be affected by the A value from 2 to 0 during the iterations. Value A can be defined in [-1,1]. The new position of search agent can be pointed around the original position (*X, Y*) and the current best agent (*X*, Y**).

Moreover, the algorithm will check if any of the agents goes beyond the search space and amend it. In every iteration, the best agent will be updated by calculating the fitness function of each search agent. If any search agent carries the most weight, it will replace the position of the best agent, and the number of tuple lists will decrease until all the tuples are empty in the table.

### *3.2.3 Test case generator*

The last component is the test case generator. It begins by explaining that each test case can be represented as a pointer, with half of the input parameters defined by the *x*-coordinate and the other half by the *y*-coordinate. For example, in the test case w1x2y1z1, parameters *W* and *X* are mapped to the *x*-coordinate, while parameters *Y* and *Z* are mapped to the *y*-coordinate. The division of the test case is illustrated in Figure 5.

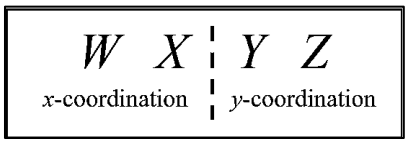


Figure 5: Division of the test case

The state of the pointer is illustrated in Figure 6, showing the agent in the WOA carrying multiple tuple lists. The agent with the highest number of tuple lists will be converted into a test case for the final test suite. Simultaneously, deductions will be made from the table containing the tuple lists held by the best solution.

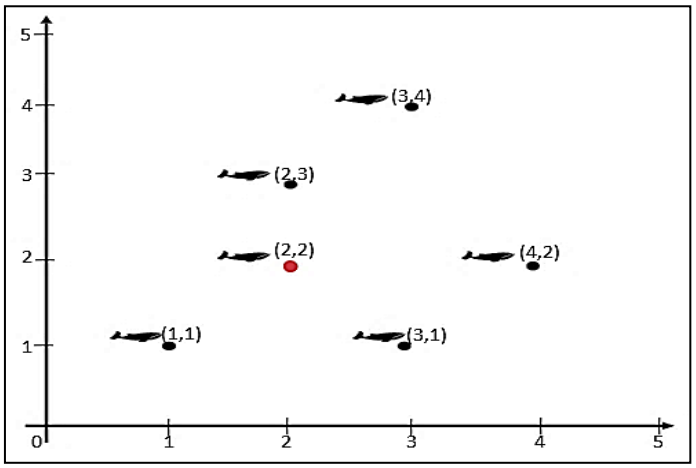


Figure 6: Agents in WOA carry tuple lists

## 4. Result and Discussion

Two experiments are conducted based on different configuration settings to verify the effectiveness of WOA based on the benchmarking defined (Ali *et al.* 2021; Othman & Zamli 2011). The experiment was conducted on Windows 10 Pro with a 64-bit CPU. The processor is a 2.20GHz Core Intel i3 with 4GB of RAM. The software used to run this experiment is Eclipse IDE for Java Developers (includes Incubating components) version 2022-09 (4.25.0). Several existing strategies for performing the IOR have been discussed in the related work section to be compared with the WOA strategy. The strategies involved, such as Greedy, Union, ParaOrder, ReqOrder, TVG, Density, AURA, ITTDG, CTJ, AFA, and TTSGA are compared with the WOA.

The first configuration setting involves 10 parameters of 3 values each, IOR ($N$, $3^{10}$, $R$). The second configuration setting consists of 10 non-uniform systems with 3 parameters of 2 value, 3 parameters of 3 value, 3 parameters of 4 value, and 1 parameter of 5 value, IOR ($N$,$2^3 3^3 4^3 5^1$, $R$). There are 100 input-output relationships, $|R|$, used for both experiments, and the test begins with 10 requested interactions and then adds another 10 interactions in the next test until all the $|R|$ is finished. The $|R|$ values have been determined, as shown in Table 1.

Table 1: 100 Input-output Relationship, $|R|$

| $\|R\|$ | Value |
|---|---|
| 10 | {1, 2, 7, 8}, {0, 1, 2, 9}, {4, 5, 7, 8}, {0, 1, 3, 9}, {0, 3, 8}, {6, 7, 8}, {4, 9}, {1, 3, 4}, {0, 2, 6, 7}, {4, 6} |
| 20 | $\|R = 10\|$ + {2, 3, 4, 8}, {2, 3, 5}, {5, 6}, {0, 6, 8}, {8, 9}, {0, 5}, {1, 3, 5, 9}, {1, 6, 7, 9}, {0, 4}, {0, 2, 3} |
| 30 | $\|R = 20\|$ +{1, 3, 6, 9}, {2, 4, 7, 8}, {0, 2, 6, 9}, {0, 1, 7, 8}, {0, 3, 7, 9}, {3, 4, 7, 8}, {1, 5, 7, 9}, {1, 3, 6, 8}, {1, 2, 5}, {3, 4, 5, 7} |
| 40 | $\|R = 30\|$ +{0, 2, 7, 9}, {1, 2, 3}, {1, 2, 6}, {2, 5, 9}, {3, 6, 7}, {1, 2, 4, 7}, {2, 5,8}, {0, 1, 6, 7}, {3, 5, 8}, {0, 1, 2, 8} |
| 50 | $\|R = 40\|$ +{2, 3, 9}, {1, 5, 8}, {1, 3, 5, 7}, {0, 1, 2, 7}, {2, 4, 5, 7}, {1, 4, 5}, {0, 1, 7, 9}, {0, 1, 3, 6}, {1, 4, 8}, {3, 5, 7, 9} |
| 60 | $\|R = 50\|$ +{0, 6, 7, 9}, {2, 6, 7, 9}, {2, 6, 8}, {2, 3, 6}, {1, 3, 7, 9}, {2, 3, 7}, {0, 2, 7, 8}, {0,1, 6, 9}, {1, 3, 7, 8}, {0, 1, 3, 7} |
| 70 | $\|R = 60\|$ +{1,4}, {0,9,3}, {3,7,9}, {0,6,8,4}, {3,5}, {1,2,8,9}, {0,6}, {0,3,7}, {2,4}, {7,8,9} |
| 80 | $\|R = 70\|$ +{3,7,6}, {3,8,9}, {2,5,6,9}, {4,7,9}, {5,8}, {4,6,7,9}), {6, 9}, {6,7}, {3,4,7}, {4,8} |
| 90 | $\|R = 80\|$ +{0,9}, {0,2,6}, {1,4,8,9}, {7,8}, {5,8,9}, {3,6,7,9}, {4,8,9}, {2,4,6,9}, {4,8,9}, {3,5,9} |
| 100 | $\|R = 90\|$ +{0,4,9}, {0,6,8,9}, {4,5,8}, {2,5}, {3,5,6,8}, {2,4,7}, {4,5,6,7}, {5,7,9}, {3,5,8,9}, {2,9} |

The results of both experiments are shown in Tables 2 and 3. Previous strategies mostly performed experiments with a value of up to 60, except for the ITTDG and AFA. This WOA also performs the experiment until $|R|$ = 100.

Table 2: Generated test suite size for IOR ($N$, $3^{10}$, $R$)

| | Computational | | | | | | | | | Metaheuristic | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\|R\|$ | Greedy | Union | ParaOrder | ReqOrder | TVG | Density | AURA | ITTDG | CTJ | AFA | TTSGA | **WOA** |
| 10 | 104 | 503 | 105 | 153 | 86 | 86 | 89 | **81** | 88 | 83 | 86 | **81** |
| 20 | 110 | 858 | 103 | 148 | 105 | 95 | 99 | 94 | 100 | **90** | 94 | **90** |
| 30 | 122 | 1599 | 117 | 151 | 125 | 116 | 132 | 114 | 118 | 106 | 111 | **105** |
| 40 | 134 | 2057 | 120 | 160 | 135 | 126 | 139 | 122 | 128 | 114 | 119 | **113** |
| 50 | 138 | 2635 | 148 | 169 | 139 | 135 | 147 | 131 | 134 | 122 | 129 | **121** |
| 60 | 143 | 3257 | 142 | 176 | 150 | 144 | 158 | 141 | 145 | **131** | 138 | **131** |
| 70 | NA | NA | NA | NA | NA | NA | NA | 139 | NA | **133** | NA | 134 |
| 80 | NA | NA | NA | NA | NA | NA | NA | 140 | NA | 137 | NA | **136** |
| 90 | NA | NA | NA | NA | NA | NA | NA | 110 | NA | **83** | NA | 138 |
| 100 | NA | NA | NA | NA | NA | NA | NA | 101 | NA | **90** | NA | 140 |

Table 3: Generated test suite size for IOR ($N$, $2^3$ $3^3$ $4^3$ $5^1$, $R$)

| | Computational | | | | | | | | Metaheuristic | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\|R\|$ | Greedy | Union | ParaOrder | ReqOrder | TVG | Density | AURA | ITTDG | CTJ | AFA | TTSGA | **WOA** |
| 10 | **137** | 505 | 144 | 154 | 144 | 144 | 144 | 144 | 144 | 144 | 144 | 144 |
| 20 | **158** | 929 | 161 | 187 | 161 | 160 | 182 | 160 | 165 | 160 | 160 | 160 |
| 30 | 181 | 1861 | 179 | 207 | 179 | 165 | 200 | 169 | 170 | **160** | 164 | 161 |
| 40 | 183 | 2244 | 183 | 203 | 181 | 165 | 207 | 173 | 173 | **161** | 166 | 162 |
| 50 | 198 | 2820 | 200 | 251 | 194 | 183 | 222 | 183 | 191 | **180** | 183 | 181 |
| 60 | 207 | 3587 | 204 | 250 | 209 | 197 | 230 | 199 | 209 | **187** | 197 | **187** |
| 70 | NA | NA | NA | NA | NA | NA | NA | 190 | NA | **187** | NA | 188 |
| 80 | NA | NA | NA | NA | NA | NA | NA | 249 | NA | 242 | NA | **241** |
| 90 | NA | NA | NA | NA | NA | NA | NA | 268 | NA | 264 | NA | **250** |
| 100 | NA | NA | NA | NA | NA | NA | NA | 260 | NA | **254** | NA | 255 |

The first experiment's configuration setting result ($N$, $3^{10}$, $R$) in Table 2 reveals that the WOA method outperforms the other strategies in producing several optimal test set sizes for $|R|$ = 30, 40, 50, and 80. Furthermore, the optimal tests specified for$|R|$ = 20 and $|R|$ = 60 are shared by WOA and AFA. For $|R|$ = 10, WOA and ITTDG have the same optimal test set at 81 cases. AFA and ITTDG are the best options in terms of test set sizes, with the WOA providing the most overall. The optimal test set for $|R|$ = 70, 90, and 100 is produced by the AFA. In the meantime, the Union strategy is the most incompetent, producing massive tests as the number of parameters increases. The WOA stands out due to its ability to consistently perform in generating relatively small test suites as compared to the traditional computational strategies, especially as the number of parameters $|R|$ Increases.

In the second experiment using configuration setting ($N$, $2^3$ $3^3$ $4^3$ $5^1$, $R$) in Table 3, Greedy produced the optimal test set for $|R|$ = 10 and $|R|$ = 20. Then, AFA produces the most optimal number of tests set for $|R|$ = 30, 40, 50, 70 and 100. Meanwhile, the WOA only produces the optimal test set at $|R|$ = 80 and $|R|$ = 90. AFA and WOA share the optimal test set for $|R|$ = 60. At this configuration setting, the WOA shows competitive results as it falls second place after AFA, producing the most optimal number of test sets. Then, the Density strategy also shows competitive results but cannot produce the optimal number of tests. The Union strategy is the worst since it generates excessive tests. The results show that the WOA strategy could generate fewer test suites.

After obtaining the result from the experiment, the next step is to perform a Wilcoxon Signed-Rank test to investigate the significant difference between the strategies further. The confidence level of the significant difference between the strategies is above 95 per cent. The null hypothesis ($H_0$) indicates no significant difference between the strategies when the p-value > 0.5. Meanwhile, the alternative hypothesis ($H_1$) assumes significant differences between the strategies. The selected strategies according to the availability data where data not available "NA" was excluded from the analysis. The analysis uses the Wilcoxon Rank test to pair the WOA with other strategies. Thus, the analysis strategy includes ITTDG, AFA, and WOA.

Table 4: Wilcoxon Signed-Rank test against ITTDG and AFA for Table 2 and Table 3

| Pairs | WOA > | WOA < | WOA = | Asymptotic Sig. (2-sided test) | Decision |
|---|---|---|---|---|---|
| ITTDG - WOA | 0 | 19 | 1 | 0.039 | Reject the $H_0$ |
| AFA - WOA | 8 | 7 | 5 | 0.766 | Retain the $H_0$ |

According to the statistical analysis in Table 4, $H_0$ is rejected for ITTDG, indicating a significant difference in test suite generation, making the WOA a better strategy than the ITTDG strategy. The majority of the positive data was taken by the WOA, with the number of ties only one. Therefore, WOA outperforms the ITTDG strategy. On the other hand, for the AFA strategy, the $H_0$ is retained, which means there is no significant difference between the strategies when the $p = 0.766$, which is much greater than the significant threshold of 0.05. However, the slight difference makes it a competitive alternative for specific testing scenarios, although not significantly better. Thus, further investigation with different parameters, configuration settings, or larger datasets may help clarify that WOA's advantages could be better performed under different conditions.

## 5. Conclusion

Exhaustive testing can be minimized by employing a testing IOR when the tester has a comprehensive understanding of the SUT and is aware of the expected system output, reducing the number of test cases compared to testing all possible interaction parameters. Therefore, this study utilizes WOA for testing IOR and demonstrates competitive results compared to other existing strategies such as Greedy, Density, TVG, Union, ParaOrder, ReqOrder, Union, ITTDG, AURA, Java Algorithm (CTJ), ITTDG, and AFA. While the concept of WOA is relatively simple, its effectiveness in searching for the best solution is unparalleled, especially in input-output based relations. It can be seen from the results that the WOA outperformed all the strategies expected of the AFA strategy. Most of the test suites generate substantially smaller amounts than the other strategies. WOA's power lies in its ability to simulate the natural hunting behaviour of humpback whales, enabling it to explore the search space efficiently. Moreover, the algorithm has a straightforward mechanism where searching is balancing either searching for a solution in a broad area or refining the solution. However, some improvement needs to be applied to the whale optimization algorithm, especially regarding slow convergence speed and local optima stagnation. In future work, we plan to enhance the drawbacks of WOA by integrating it with other algorithms to overcome the drawbacks and produce an optimal number of test suites.

## Acknowledgment

The author would like to acknowledge the support of the Fundamental Research Grant Scheme (FRGS) under grant number FRGS/1/2020/ICT01/UNIMAP/02/1 from the Ministry of Higher Education Malaysia.

### References

Ahmed B.S., Abdulsamad T.S. & Potrus M.Y. 2015. Achievement of minimized combinatorial test suite for configuration-aware software functional testing using the Cuckoo Search algorithm. *Information and Software Technology* **66**: 13–29.

Ali A.S.M., Othman R.R., Yacob Y.M. & Abdelmula H.S.A.B. 2021. An efficient combinatorial input output-based using adaptive firefly algorithm with elitism relations testing. *Advances in Science, Technology and Engineering Systems Journal* **6**(4): 223–232.

Amiriebrahimabadi M. & Mansouri N. 2024. A comprehensive survey of feature selection techniques based on whale optimization algorithm. *Multimedia Tools and Applications* **83**(16): 47775-47846.

Arshem J. 2010. Test Vector Generator (TVG). https://sourceforge.net/projects/tvg/ (30 March 2025).

Bryce R.C. & Colbourn C.J. 2007. The density algorithm for pairwise interaction testing. *Software Testing, Verification and Reliability* **17**(3): 159-182.

Chen X., Gu Q., Li A. & Chen D. 2009. Variable strength interaction testing with an ant colony system approach.

*2009 16th Asia-Pacific Software Engineering Conference*, pp. 160–167.
Cohen M.B., Colbourn C. J. & Ling A.C.H. 2003. Augmenting simulated annealing to build interaction test suites. *14th International Symposium on Software Reliability Engineering,* pp. 394–405.
Dorigo M., Birattari M. & Stutzle T. 2006. Ant colony optimization. *IEEE Computational Intelligence Magazine* **1**(14): 28–39.
Esfandyari S. & Rafe V. 2021. GALP: A hybrid artificial intelligence algorithm for generating covering array. *Soft Computing* **25**(11): 7673–7689.
Jahanbin S. & Zamani B. 2018. Test model generation using equivalence partitioning. *2018 8th International Conference on Computer and Knowledge Engineering, ICCKE 2018*, pp. 98–103.
Lei Y., Kacker R., Khun D.R., Okun V. & Lawrence J. 2008. IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing. *Softw. Test. Verif. Reliab* **18**(3): 125–148.
Leveson N. 2020. Are you sure your software will not kill anyone?. *Communications of the ACM* **63**(2): 25–28.
Mirjalili S. 2016. SCA: A Sine Cosine Algorithm for solving optimization problems. *Knowledge-Based Systems* **96**: 120–133.
Mirjalili S. & Lewis A. 2016. The whale optimization algorithm. *Advances in Engineering Software* **95**: 51–67.
Muazu A.A., Hashim A.S. & Sarlan A. 2022. Application and adjustment of "don't care" values in t-way testing techniques for generating an optimal test suite. *Journal of Advances in Information Technology* **13**(4): 347–357.
Nadimi-Shahraki M.H., Zamani H., Asghari Varzaneh Z. & Mirjalili S. 2023. A systematic review of the whale optimization algorithm: Theoretical foundation, improvements, and hybridizations. *Archives of Computational Methods in Engineering* **30**(7): 4113-4159.
Norris K.S. 2023. *Whales, Dolphins, and Porpoises*. Berkeley and Los Angeles, CA: Univ of California Press.
Ong H.Y. & Zamli K.Z. 2011. Development of interaction test suite generation strategy with input-output mapping supports. *Scientific Research and Essays* **6**(16): 3418–3430.
Othman R.R., Khamis N. & Zamli K.Z. 2014. Variable strength t-way test suite generator with constraints support. *Malaysian Journal of Computer Science* **27**(3): 204–217.
Othman R.R. & Zamli K.Z. 2011. ITTDG: Integrated T-way test data generation strategy for interaction testing. *Scientific Research and Essays* **6**(17): 3638–3648.
Othman R.R., Zamli K.Z. & Nugroho L.E. 2012. General variable strength t-way strategy supporting flexible interactions. *Maejo International Journal of Science and Technology* **6**(3): 415–429.
Ramli N., Othman R.R., Abdul Khalib Z.I. & Jusoh M. 2017. A review on recent T-way combinatorial testing strategy. *MATEC Web of Conferences* **140**: 01016.
Ramli N., Othman R.R. & Ali M.S.A.R. 2016. Optimizing combinatorial input-output based relations testing using Ant Colony algorithm. *2016 3rd International Conference on Electronic Design (ICED)*, pp. 586–590.
Schroeder P.J. & Korel B. 2000. Black-box test reduction using input-output analysis. *ACM SIGSOFT Software Engineering Notes* **25**(5): 173–177.
Shiba T., Tsuchiya T. & Kikuno T. 2004. Using artificial life techniques to generate test cases for combinatorial testing. *Proceedings of the 28th Annual International Computer Software and Applications Conference, COMPSAC 2004*, pp. 72–77.
Tatale S. & Prakash V.C. 2022. Combinatorial test case generation using q-value based particle swarm optimization. *Revue d'Intelligence Artificielle* **36**(2): 319–326.
Younis M.I., Alsewari A.R.A., Khang N.Y. & Zamli K.Z. 2020. CTJ: Input-output based relation combinatorial testing strategy using jaya algorithm. *Baghdad Science Journal* **17**(3): 1002–1009.
Zamli K.Z., Othman R.R., Younis M.I. & Mohamed Zabil M.H. 2011. Practical adoptions of T-way strategies for interaction testing. *Software Engineering and Computer Systems: Second International Conference, ICSECS 2011, Kuantan, Pahang, Malaysia, June 27-29, 2011, Proceedings, Part III 2*, pp. 1–14.
Zhang J., Zhang Z. & Ma F. 2014. Introduction to combinatorial testing. *Automatic Generation of Combinatorial Test Data*: 1-16. Berlin, Heidelberg: Springer.
Ziyuan W., Changhai N. & Baowen X. 2007. Generating combinatorial test suite for interaction relationship. *SOQUA'07: Fourth International Workshop on Software Quality Assurance - In Conjunction with the 6th ESEC/FSE Joint Meeting*, pp. 55–61

*Faculty of Electronic Engineering Technology (FTKEN)*
*Universiti Malaysia Perlis,*
*02600 Arau, Perlis, MALAYSIA*
*E-mail: anjilajsuali91@gmail.com*[*]*, husnarose@unimap.edu.my*

*Advanced Computing, Centre of Excellence (CoE),*
*Universiti Malaysia Perlis*
*02600 Arau, Perlis, MALAYSIA*
*E-mail: nuraminah@unimap.edu.my, rozmie@unimap.edu.my, hasneeza@unimap.edu.my, iszaidy@unimap.edu.my*

*Faculty of Computer and Information Sciences,*
*Prince Sultan University, Riyadh*
*P.O.Box No. 66833 Rafha Street,*
*Riyadh, 11586, SAUDI ARABIA*
*E-mail: Njamail@psu.edu.sa*

*Faculty of Sciences and Technology,*
*Universitas Airlangga,*
*60115, Surabaya Jawa Timur, INDONESIA*
*E-mail: rimuljohendradi@fst.unair.ac.id*

Received: 18 December 2024
Accepted: 2 May 2025

[*]Corresponding author