IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Revue d'Intelligence Artificielle
Vol. 36, No. 2, April, 2022, pp. 319-326
Journal homepage: http://iieta.org/journals/ria

Combinatorial Test Case Generation Using Q-Value Based Particle Swarm Optimization

Subhash Tatale*, Vudatha Chandra Prakash

Department of CSE, Koneru Lakshmaiah Education Foundation, Vaddeswaram 520013, AP, India

Corresponding Author Email: subhashtatale@gmail.com

https://doi.org/10.18280/ria.360217

Received: 15 January 2022
Accepted: 4 April 2022

Keywords:
combinatorial testing, pairwise testing, particle swarm optimization, test case generation

ABSTRACT

Combinatorial testing is an effective method for generating test cases. Pairwise testing is a combinatorial approach that evaluates the interactions between the input test parameters while reducing test case size by selecting a broader search area. Most combinatorial testing research focuses on developing novel approaches for generating an optimal number of test cases that cover pairwise combinations of input test parameters. Using existing test case generation techniques, optimal or near-optimal combinatorial test cases are generated in polynomial time. The authors presented the Q-value-based Particle Swarm Optimization (Q-PSO) technique for efficiently and effectively generating an optimal number of test cases. The primary goals of the proposed technique are to generate test cases using a Q-value based PSO, which is easier to build and has fewer parameters to define than other meta-heuristic search methodologies and to put the proposed technique into practice and report on an empirical study that examines and verifies the significant impact factors in the proposed approach. Q-value is used to evaluate the particles (referred to as test cases) in the Q-PSO. The reward is totalled in the Q-value, which serves as the fitness function for PSO evolution. The Q-value of each particle determines its performance and indicates how quickly the particle can lead the system's state to the set of objective states. The authors used the Q-PSO technique to validate the efficiency and efficacy of the proposed approach. The Q-PSO technique's results are compared to existing metaheuristics and computation-based techniques. In most inputs based on the development environment, meta-heuristic search techniques take significantly longer than other greedy techniques. For some inputs, the proposed Q-PSO technique outperforms existing meta-heuristics techniques. Q-PSO results are also compared to IPOG, ITCH, Jenny, TConfig, TVG, and other well-known computational-based techniques. The goal of the comparison is to examine how the size of the test cases generated has grown over time.

1. INTRODUCTION

Unexpected interactions between software and system elements become increasingly prevalent as the complexity of a software project grows. These undesired encounters sometimes lead unwanted and difficult issues to detect. Regression testing, test case generation, test oracle problem, test coverage criterion design, and fault localization problem are the new research focus areas of software testing domain. The researchers presently recognized that the test case generation as a promising and attentive research area. The test case generation is considered as a promising and attentive research area by researchers nowadays. In this paper, the authors are giving more emphasis on generating combinatorial test cases. In the section, combinatorial testing and mathematical background of the proposed approach is summarized.

1.1 Overview of combinatorial testing

Combinatorial Testing (CT) is a specification-based test case generation technique and develops test cases depending on the artefacts being evaluated, which are based on functional requirements or complete development specifications. It is a helpful technique for discovering hardware or software systems issues based on the combinations of input or output system parameters. The input or output parameters of graphical user interfaces, entire software product and web forms have been tested using this technique.

Pairwise testing is one of the most extensively used combinatorial testing strategies. Because it examines all possible combinations of all input parameters, it is a valuable testing technique that considers all discrete possibilities of each pair of system input parameters. Exhaustive testing takes more time than pairwise testing, since the common of software problems are occurred by a single or two input parameters.

Let us consider an e-commerce configurable software system as a model to explain pairwise testing. In this case, there are four parameters in the system. The user can make payments through different smart phones using different payment modes made available by the system. Different web servers and database servers are the other system configurations of the e-commerce software system.

Table 1 shows the system configurations, which have four parameters combined with three values for each parameter. Although multiple testing methodologies for this software system may be helpful, unanticipated interactions between parameters are a general problem of failure of the software's. If the number of parameters and values rises, the software is more likely to fail. Manufacturers may need to analyze all

potential combinations of parameters to limit this danger and ensure the quality of such software. It is known as exhaustive testing, and it necessitates 81 test cases (i.e., 3x3x3x3). The pairs are nothing more than interactions between parameter values, and they are determined using the formula below.

The total number of pairs generated = (Par - 1 + Par – 2 +...+ Par - n) * (Val2), where Par denotes the number of parameters and Val denotes the number of values of those parameters. Total (3+2+1) * 32 = 54 pairs (two-way) are formed in the given case. However, as shown in Table 2, only nine test cases can be used to investigate all these pairwise interactions of system characteristics.

Table 1. Configurations of E-commerce software system

Configurations / Parameters	Smart Phone	Payment Mode	Web Server	Database Server
Values	IPhone	Paypal	Tomcat	SQL
	Android	Google pay	Glassfish	MongoDB
	Amazon Fire Phone	Phone pay	WebSphere	Access

Table 2. Pairwise test cases for the system configurations in Table 1

Test Case No.	Smart Phone	Payment Mode	Web Server	Database Server
1	Amazon Fire Phone	Phone pay	Tomcat	Access
2	Amazon Fire Phone	Google pay	WebSphere	MongoDB
3	Android	Phone pay	WebSphere	SQL
4	Android	Google pay	Glassfish	Access
5	IPhone	Phone pay	Glassfish	MongoDB
6	Android	Paypal	Tomcat	MongoDB
7	IPhone	Google pay	Tomcat	SQL
8	IPhone	Paypal	WebSphere	Access
9	Amazon Fire Phone	Paypal	Glassfish	SQL

This reduces the number of test instances (9 from 81) while covering all pairwise possibilities (54 pairs). The reduction in test case size from 81 to 9 in the example above may not appear to be substantial. It is, nevertheless, appealing for more complex inputs, such as ten parameters with ten values each. A total of 1010 test cases will be generated if exhaustive testing is performed. If pairwise testing is used, the proposed approach will only yield 172 test cases. Different mathematical notations have been employed to express such combinations as the importance of combinatorial testing has grown. In the next section, mathematical background in the view of combinatorial testing is discussed.

1.2 Mathematical background

Combinatorial test cases are generated using algebraic and computational mathematical methodologies. The extensions of mathematical functions are commonly used in algebraic procedures for generating Orthogonal Arrays (OA). The basis for describing combinatorial testing is an OA represented as OA (N; t; k; v) of size N, and strength t is N x k array in which there are t-interaction elements that appear exactly N/vt times for every N x t sub-array. The limitations of an orthogonal array are that it requires homogeneous parameters and values, and it is not available for all input parameter combinations. In general, the computations required by algebraic techniques are minor. However, algebraic methodologies regularly force limitations on the framework boundaries and qualities to which they can be applied [1].

The computational technique, in contrast to the algebraic approach, typically relies on producing all feasible combinations. It takes a lot of searching in the combinatorial space to find all the possible combinations. In the computational method, One Parameter at Time (OPAT) and One Test at Time (OTAT) techniques are used for building test cases. The OTAT technique creates a single test case and uses a variety of elements to assess its coverage. This approach iteratively analyses the needed interaction, producing a complete test case for each iteration. The technique greedily examines that the generated test case covers the most unexplored interactions and should be included in the final test suite at the end of each cycle. In the case of OPAT, a specific approach builds a final test case in a sequential way, one parameter at a time. This method builds the test case in stages, horizontally extending it till it is complete. If necessary, vertical extension is used to cover any remaining uncovered interactions [2]. The CA notation is a collection of standard mathematical notations for defining and constructing parameter and value combinations. A CA (N; t; k; v) depicts a N x k array containing v values, where each N x t sub-array contains all ordered subsets from v values of size t at least k times [3]. For example, the notation CA (9; 2, 33) represent a test suite of system with three parameters each with three values, to cover pairwise interactions. N = 9 indicates nine test cases are generated using pairwise testing. In some testing circumstances, parameters may have a variable number of values. In these cases, the Mixed Covering Array (MCA) is used. An MCA (N; t; k; (v1, v2, ..., vn)) is a N x k array in which each column i ($1 \leq i \leq k$) contains only levels from the set Vi and the rows of each N t sub-array at least once cover all t-sets from the t columns. For example, a test suite with the notation MCA (21; 2, $5^1 3^8 2^2$) represents N=21 (i.e., twenty-one test cases) for a system with eleven parameters where one parameter is having five values, eight parameters are having three values each and two parameters are having two values each to cover the two-way (pairwise) interactions.

2. SIGNIFICANCE OF THE WORK

Combinatorial explosion is a well-known issue of hardware as well as software systems because of the interactions of many systems configurations. This issue frequently has significant testing and quality assurance implications [4]. Due to time and budget restrictions, thorough testing is nearly impossible. Combinatorial test case generation is an NP-hard

problem, which means that as the parameter size grows, the computational time and degree of problem complexity expand exponentially [5]. As a result, determining the optimal number of test cases is challenging. To address this NP-hard problem, many artificial intelligence-based algorithms have been designed to find optimal solutions in polynomial time. To tackle these challenges, a sampling approach is used which selects a subset of inputs as test data from essentially infinite search space. Combinatorial test case generation is an NP-hard problem, which means that the computational time and degree of problem complexity grow exponentially as the parameter size grows. As a result, discovering the optimized number of test cases is a challenging task. According to current research, artificial intelligence-based testing strategies have shown success in achieving a solution that is close to ideal, because of which the size of test cases is smaller than existing strategies.

The authors of this paper suggested Q-value based PSO, a meta-heuristics-based algorithm for generating an optimal number of combinatorial test cases. The primary goals of this research work are listed below:

1. To generate test cases using a Q-value based PSO, which is simpler to build and has less parameter to define than other meta-heuristic search methodologies.

2. To put the proposed strategy into practice and report on an empirical study that examines the significant impact factors in the proposed approach and verifies its efficacy and efficiency.

3. REVIEW OF RELATED WORK

In this section, the authors discussed related work into two parts: meta-heuristics and computational based search techniques employed to develop combinatorial test cases.

3.1 Meta-heuristic-based techniques

The use of meta-heuristics as a computational approach for generating combinatorial test cases has recently gotten a lot of interest. Meta-heuristic-based algorithms appear to outperform other computational techniques, according to the literature. Meta-heuristic algorithms, in general, start with a random number of solutions. To improve these solutions, they are exposed to a series of adjustments. The appropriate candidates are chosen at each iteration until all the required combinations have been covered. Early attempts to produce combinatorial test cases using meta-heuristic algorithms included Ant Colony Optimization (ACO), Genetic Algorithms (GA), and Simulated Annealing (SA). Cohen et al. [6] employed the SA approach, which uses a vast random search space and transformation equations which are based on probability to generate combinatorial test cases. Shiba et al. [7] introduced a GA algorithm for generating combinatorial test cases. It all starts with chromosomes, which are randomly created test instances. On these chromosomes, mutation and crossover occur until and unless a termination condition is satisfied. The final test suite is generated by selecting best chromosomes at each cycle. ACOs, unlike GAs, search for food in the same way that ants do. The complexity of GA and ACO algorithms, as well as their potential for consuming computational resources, has been questioned. SA is prone to early convergence due to its sensitivity to its initial beginning point in the search space. As a result of these qualities, these algorithms are confined to low contact strengths. PSO algorithm is used to solve an extensive variety of combinatorial optimization applications. Ahmed et al. suggested many Particle Swarm Optimization (PSO) algorithms that simulate bird's swarm behavior [8-13]. Internally, PSO variant algorithms execute iterative global and local searches to discover the solution that will be added to the final suite until all the combination pairs are covered. Prakash et al. [14] completed a comprehensive review on combinatorial test case generation using different PSO algorithms. The performance of the various PSO variant algorithms was critically discussed. Bewoor et al. evaluated the performance of PSO with Tabu Search (TS) and GA [15, 16]. It is found that PSO outperforms the other meta-heuristics for random inputs. Bewoor et al. presented the Hybrid PSO solution for solving the combinatorial optimization problem of No Wait Flow Shop Scheduling, indicating that PSO outperforms other meta-heuristics algorithms [17]. Bangare et al. have worked in the software testing metrics and provided the research directions in their software quality work [18-22]. Tatale et al. [23-26] applied meta-heuristics techniques like SA, PSO to generate combinatorial test cases from UML sequence and activity diagrams.

3.2 Computational based techniques

A lot of contribution is made by many researchers into development of computational-based strategies for generating combinatorial test cases with a wide range of input configurations. Cohen et al. [27] introduced the Automatic Efficient Test Generator (AETG) tool based on the OTAT technique. After building all the required combinations, AETG builds a test case. For each iteration, AETG produces several test cases. The test case is carefully selected to cover the most unexplored combinations among these test cases. Lei and Tai [28] proposed the In-Parameter-Order (IPO) technique, which is based on the OPAT strategy. IPO creates a pairwise test set for the first two parameters, then expands it by creating a pair for the following three parameters, and so on, until all the system parameters are covered. If necessary, a vertical growth is made to cover the uncovered combinations. This tool performs poorly in terms of test size due to its unpredictable behavior. The developed Test Vector Generator (TVG) [29] is tool that produces test cases. The graphical user interface is designed to generate combinatorial test cases which are based on the input-output relationship of the system parameters. This strategy covers n-way combinations of the parameters. By creating test cases that cover the 1-way interaction, the OTAT technique [30] was applied. The test cases were then expanded to include pairwise combinations, and the process was repeated until all n-way interactions had been covered. the Eclipse Java plug-in tools Intelligent Test Case Handler (ITCH) [31] was created to produce test cases for n-way coverage. This strategy takes a long time and delivers unfavorable results due to the comprehensive search. Pande et al. [32-34] have presented a comprehensive review on capsule networks and applied it for IR which could be further applied in PSO.

4. PROPOSED WORK

The proposed Q-PSO algorithm is described in this section. The algorithm is separated into two sections, one for the Q-value and one for the PSO operation. Section 4.1 describes the

creation of particle Q-values, whereas section 4.2 describes the PSO operation and the Q-PSO flowchart.

4.1 Q-value generation

In the existing PSO algorithm, the test cases are considered as particles. Each particle consists of pairs of parameters and values of system configurations. Once the particle is generated, it calculates the number of pairs generated. These particles are made up of repeating and unique pairs that other particle in earlier iterations have covered because of repeating pairs in the particles, the size of the test suite increases. This is problem of fitness function of the existing PSO algorithm. To overcome this problem in the fitness function of the existing PSO algorithm, the authors of this research article proposed Q-value based PSO. The Q-value of each particle is calculated based on the number of pairs generated. When a particle is chosen, the Q-value of the chosen particle is calculated based on the reward and penalty of the pairs generated by the particle. The following is a description of the particle Q-value generating strategy.

These generated pairs are used to calculate the particle's Q-values. The penalty and reward are given to the pairs generated from the selected particles. Unique pairs receive rewards, but repetitive pairs that other particle in a prior iteration have already covered receive penalties. The Q-value of the particle is determined using these penalty and reward operators. The Q-value is generated based on the below formula.

$$q_{Value}(\mathrm{i}) = \sum_{j=1}^{m} \alpha * (UP) + \sum_{j=1}^{m} \beta * (DP) \tag{1}$$

α is the reward operator, which carries positive values, and β is the penalty operator, which carries negative values. UP is unique pairs generated by the particle, and DP is duplicate pairs generated by the particle. k is the number of parameters of the system configuration, m is maximum pairs generated by the selected particle, which is calculated using (k x (k-1))/2. If n particles are in the swarm, the Q-PSO algorithm takes n trials in one generation. An operator selects a particle based on its Q-value and operates on the system environment in each trial. The advantage of this Q-value generation is that it only considers unique pairs generated; because of this, the size of the test suite is minimum, and it gives maximum coverage.

4.2 Q-PSO operation

Initialization of the swarm and PSO evolution are two main steps in the PSO operation in Q-PSO. The Q-values of each particle in equation (1) create the fitness values for PSO evolution. The Q-value of each particle affects how well it manages the system. In the proposed Q-PSO, each particle's Q-value specifies how rapidly it may move the system's state to the set of objective states.

Poli et al. [34] proposed the PSO algorithm in 2007. It is a population-based algorithm that simulates the swarm behaviour of flocks of birds. PSO consists of sections for local and global searches that simultaneously manipulate a set number of potential solutions. The population is considered as swarm, and each proposed solution is considered as particle. Each particle keeps track of important information about its journey in the search space to find a better solution to the problem. This data is for the randomly selected i^{th} particle which contains the velocity (vi), position (xi), global best ($gbest_i$) and personal best ($pbest_i$).

The PSO method generates a random number of particles in random positions, then modifies their velocity rates to move them towards $pbest_i$ and $gbest_i$ respectively. Each particle moves through the search space by updating its position according to predefined rules. The velocity and position of the i^{th} particle for the d^{th} dimension search space are updated as follows:

Eq. (2) updates the velocity of the particle.

$$\begin{aligned} Vel_{i,d}(n) = {} & wVel_{i,d}(n\text{-}1) \\ & + c1rl(gbest_{i,d}(n\text{-}1) - Pos_{i,d}(n\text{-}1)) \\ & + c2r2(pbest_{i,d}(n\text{-}1) - Pos_{i,d}(n\text{-}1)) \end{aligned} \tag{2}$$

Eq. (3) updates the position of the particle.

$$Pos_{i,d} = X - Pos_{i,d}(n\text{-}1) + Vel_{i,d}(n) \tag{3}$$

where d is the dimension, i is particle index and n are the number of iterations, the inertia weight factor is w. r1, r2 are random numbers and c1, c2 are acceleration coefficients which are used to modify the weight distribution of the particles.

The following section outlines the flowchart and algorithm of Q-PSO. The flowchart of how each particle evolves by the end of each trial is shown in Figure 1.

The below is the pseudo code of the proposed Q-PSO method. The parameters and values of the system configuration are passed to the Q-PSO algorithm as the first two arguments.

1. Let Ps be set of all the combinations of parameters and values
2. Generate pairs Ps
3. Let Ts is a set of candidate tests
4. While particles do not cover all pairs
5. Randomly select particles from the search space
6. Evaluate particles for pairwise interactions with Ps
7. If particle covers unique pairs from Ps
8. Calculate rewards for unique pairs
9. Otherwise calculate penalty for duplicate pairs
10. Calculate Q-value of the particle based on reward and penalty
11. If Q-value is greater
12. Select particle into solution set Ts
13. End once all the pairs are covered

A combination list, Ps, containing pairwise interactions of parameters is constructed using combination generator logic (Line 1 and 2). After then, the programme generates a set of random parameter values for each particle (Line 5). A candidate particle (or a candidate test case) enters a check weight function while attempting to evaluate its coverage in the search space (Line 6). The check weight function returns the number of pairs covered after converting a given test case to its base pairs; for example, a weight of six indicates that the candidate test case can cover six pairs. Each candidate test case covers a maximum of (k x (k-1))/2 pairs, where k is the number of parameters. Indexing is required in this circumstance to speed up the process of discovering the covered pairings.

The candidate test case is made up of all unique pairs, duplicate pairs, or a combination of unique and duplicate pairs that a previously generated test case has already covered in an earlier iteration. The candidate test case's Q-value is calculated using the reward and penalty operator (Step 7 to 9). Equation (1) is used to calculate the Q-value (Line 10). The approach

modifies the particle's position in the next iteration using the update rule (Eqns. (2) and (3)), considering. After changing the position of the particles, the programme re-evaluates them and looks for a better Q-value. The algorithm adds particles to Ts of the output test cases if a better Q-value is found (Line 12). Until Ps is empty, the process is repeated.

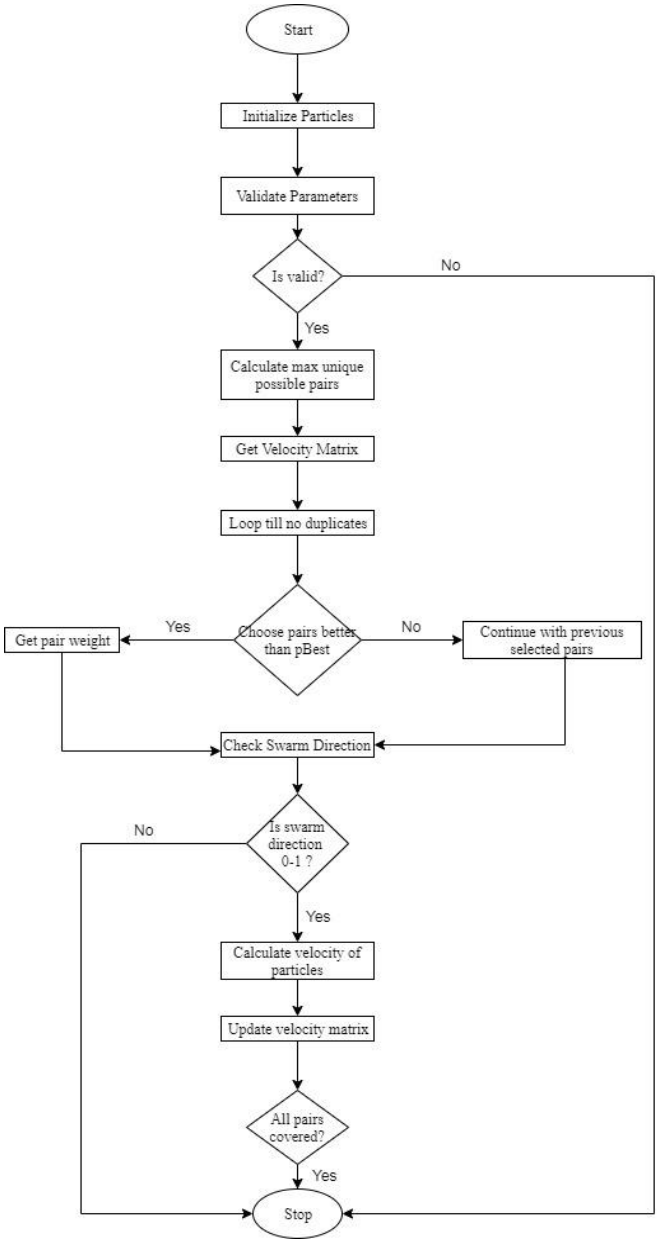


Figure 1. Flowchart of the proposed work

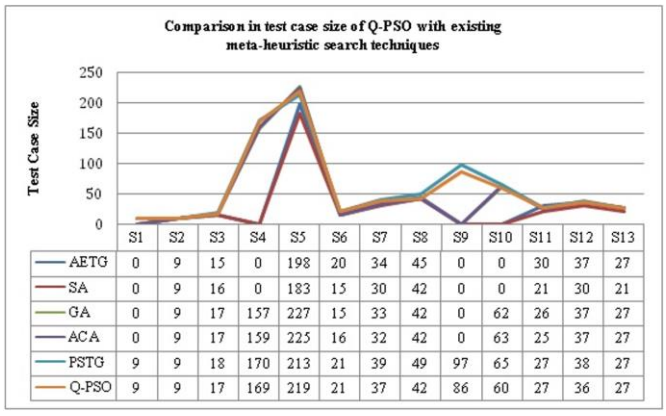
5. EXPERIMENTAL RESULTS

The results of the proposed Q-PSO algorithm are discussed in this section. The proposed Q-PSO algorithm is written in Python and operates on a Lenovo ThinkPad T400 laptop with the Windows 10 operating system. The proposed Q-PSO algorithm is tested using inputs that have been frequently used in the literature to compare the efficacy of various approaches. Some inputs have the same number of parameters as others, and others have a different number of parameters. The authors of this research article divided the experiments into two parts. The first section focuses on analyzing and comparing Q-PSO to previously reported meta-heuristic search results. The evaluation and comparison of Q-PSO with other computational-based techniques are discussed in the second part.

5.1 Result comparison of Q-PSO with existing meta-heuristic search techniques

The results of Q-PSO directly compared to the results of some meta-heuristics methods reported by Chen et al. [36]. Furthermore, because both GA and ACA outcomes are obtained from AETG, the AETG results are compared. The NA value in the cells shows that the results of those inputs are Not Available in the published literatures. In all inputs, SA always generated a minimum size of test cases. However, the SA strategy was time-consuming because of the large number of computational resources required. When the input was more complicated, the current PSO-based technique still produced promising outcomes. By comparing programming language, execution time and platform of various techniques, the efficiency and effectiveness of the PSO technique were proven. According to the authors, meta-heuristic search approaches take significantly longer than other greedy methods in most inputs based on this environment. The results generated in terms of test case size by each technique are shown in Table 3. There are usually two phases or stages to these meta-heuristics. Random test cases with fixed sizes are generated in the first stage. The second stage uses the meta-heuristic search techniques to select test cases which cover a wide range of conceivable combinations. When compared to previous meta-heuristics methods, the proposed Q-PSO algorithm outperforms them for some inputs. The proposed Q-PSO algorithm generates optimized number of test cases than existing PSTG algorithm for the mixed covering array input configuration i.e., System S6 to S13.

The Figure 2 represents the comparison of the results generated using Q-PSO algorithm and existing meta heuristic search techniques. The results show the number of test cases generated by each strategy.



	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13
AETG	0	9	15	0	198	20	34	45	0	0	30	37	27
SA	0	9	16	0	183	15	30	42	0	0	21	30	21
GA	0	9	17	157	227	15	33	42	0	62	26	37	27
ACA	0	9	17	159	225	16	32	42	0	63	25	37	27
PSTG	9	9	18	170	213	21	39	49	97	65	27	38	27
Q-PSO	9	9	17	169	219	21	37	42	86	60	27	36	27

Figure 2. Result comparison of Q-PSO with existing meta-heuristic search techniques

5.2 Result comparison of Q-PSO with existing computation-based search techniques

The findings of Q-PSO are compared to those of IPOG, ITCH, Jenny, TConfig, TVG, and other well-known computational-based techniques. The purpose of the comparison is to look at how the size of the test cases generated has grown over time. The number of parameters and

values can be fixed (i.e., a covering array) or variable (i.e., mixed covering array). Many strategies have been developed to create covering arrays, and each algorithm is designed to solve a specific set of issues.

IPO technique is based on the OPAT strategy which generates pairwise test set for the first two parameters, then expands it by creating a pair for the following parameters, and so on, until all the system parameters are covered. If necessary, a vertical growth is made to cover the uncovered combinations. This tool performs poorly in terms of test size due to its unpredictable behavior. TVG technique generates combinatorial test cases which are based on the input-output relationship of the system parameters. Jenny technique is dependent on OTAT strategy. ITCH technique takes a long time and delivers unfavorable results due to the comprehensive search.

In the majority of the inputs, Q-PSO beats other techniques, as seen in Table 4. Though the result Q-PSO algorithm is not the best for some input configurations, the number of test cases is within an acceptable range.

The Figure 3 represents the comparison of the results generated using Q-PSO algorithm and computation-based techniques. The results show the number of test cases generated by each strategy.

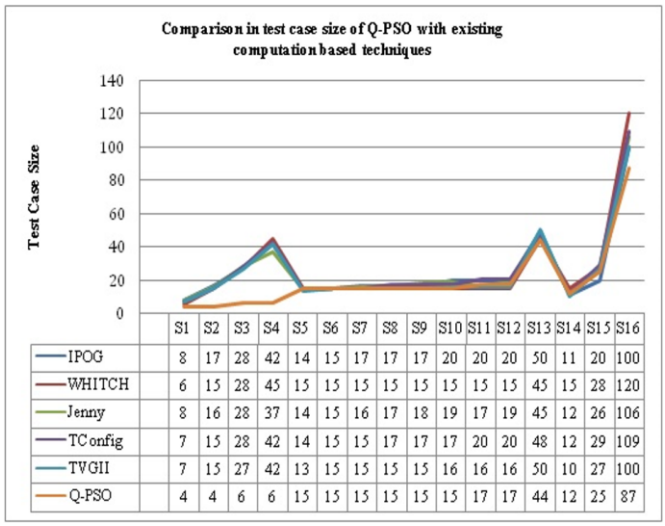


Figure 3. Result comparison of Q-PSO with existing computation-based techniques

Table 3. Result comparison in test case size of Q-PSO with existing meta-heuristic search techniques

System	System Configuration	AETG	SA	GA	ACA	PSTG	Q-PSO
S1	3^3	NA	NA	NA	NA	9	9
S2	3^4	9	9	9	9	9	9
S3	3^{13}	15	16	17	17	18	17
S4	10^{10}	NA	NA	157	159	170	169
S5	10^{20}	198	183	227	225	213	219
S6	$5^1 3^8 2^2$	20	15	15	16	21	21
S7	$6^1 5^1 4^6 3^8 2^3$	34	30	33	32	39	37
S8	$7^1 6^1 5^1 4^5 3^8 2^3$	45	42	42	42	49	42
S9	$10^1 9^1 8^1 7^1 6^1 5^1 4^1 3^1 2^1$	NA	NA	NA	NA	97	86
S10	$10^1 6^2 4^3 3^1$	NA	NA	12	13	65	60
S11	$5^1 4^4 3^{11} 2^5$	30	21	26	25	27	27
S12	$4^{15} 3^{17} 2^{29}$	37	30	37	37	38	36
S13	$4^1 3^{39} 2^{35}$	27	21	27	27	27	27

Table 4. Result comparison in test case size of Q-PSO with existing computation-based search techniques

System	System Configuration	IPO	ITCH	Jenny	TConfig	TVG	Q-PSO
S1	2^2	8	6	8	7	7	4
S2	2^3	17	15	16	15	15	4
S3	2^4	28	28	28	28	27	6
S4	2^5	42	45	37	42	42	6
S5	3^5	14	15	14	14	13	15
S6	3^6	15	15	15	15	15	15
S7	3^7	17	15	16	15	15	15
S8	3^8	17	15	17	17	15	15
S9	3^9	17	15	18	17	15	15
S10	3^{10}	20	15	19	17	16	15
S11	3^{11}	20	15	17	20	16	17
S12	3^{12}	20	15	19	20	16	17
S13	5^{10}	50	45	45	48	50	44
S14	$2^2 3^3$	11	15	12	12	10	12
S15	$2^{13} 4^5$	20	28	26	29	27	25
S16	$2^7 3^2 4^1 10^2$	100	120	106	109	100	87

6. CONCLUSIONS

By keeping the capacity to detect large errors, combinatorial testing can significantly lower the software testing cost. Generating optimal number of combinatorial test cases, on the other hand, is an NP-hard problem that has yet to be solved. The authors of this research work proposed and evaluated pairwise testing technique for software test case generation using Q-value based PSO. The authors were encouraged by the Q-PSO results, which showed that for most of the system input

sizes studied, they were able to get the optimal number of test cases. Q-PSO outperforms existing meta-heuristics and computational-based search techniques in the great majority of cases.

REFERENCES

[1] Cheng, C.S. (1980). Orthogonal arrays with variable numbers of symbols. The Annals of Statistics, 8(2): 447-453. http://dx.doi.org/10.1214/aos/1176344964

[2] Hartman, A., Raskin, L. (2004). Problems and algorithms for covering arrays. Discrete Mathematics, 284(1-3): 149-156. http://dx.doi.org/10.1016/j.disc.2003.11.029

[3] Ronneseth, A.H., Colbourn, C.J. (2009). Merging covering arrays and compressing multiple sequence alignments. Discrete Applied Mathematics, 157(9): 2177-2190.

[4] Tatale, S.B., Prakash, V.C. (2020): Enhancing acceptance test driven development model with combinatorial logic. International Journal of Advanced Computer Science and Applications (IJACSA), 11(10): 268-278. http://dx.doi.org/10.14569/IJACSA.2020.0111036

[5] Kuhn, R., Lei, Y., Kacker, R. (2008). Practical combinatorial testing: Beyond pairwise. IT Professional, 10(3): 19-23. http://dx.doi.org/10.1109/MITP.2008.54

[6] Cohen, M.B., Colbourn, C.J., Ling, A.C. (2003). Augmenting simulated annealing to build interaction test suites. In 14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003, Denver, CO, USA, pp. 394-405. http://dx.doi.org/10.1109/ISSRE.2003.1251061

[7] Shiba, T., Tsuchiya, T., Kikuno, T. (2004). Using artificial life techniques to generate test cases for combinatorial testing. In Proceedings of the 28th Annual International Computer Software and Applications Conference, Hong Kong, China, pp. 72-77. https://doi.org/10.1109/CMPSAC.2004.1342808

[8] Ahmed, B.S., Zamli, K.Z. (2010). PSTG: A t-way strategy adopting particle swarm optimization. In 2010 Fourth Asia International Conference on Mathematical/Analytical Modelling and Computer Simulation, Kota Kinabalu, Malaysia, pp. 1-5. http://dx.doi.org/10.1109/AMS.2010.14

[9] Ahmed, B.S., Zamli, K.Z. (2011). A variable strength interaction test suites generation strategy using particle swarm optimization. Journal of Systems and Software, 84(12): 2171-2185. http://dx.doi.org/10.1016/j.jss.2011.06.004

[10] Ahmed, B.S., Gambardella, L.M., Afzal, W., Zamli, K.Z. (2017). Handling constraints in combinatorial interaction testing in the presence of multi objective particle swarm and multithreading. Information and software Technology, 86: 20-36. http://dx.doi.org/10.1016/j.infsof.2017.02.004

[11] Ahmed, B.S., Sahib, M.A., Potrus, M.Y. (2014). Generating combinatorial test cases using Simplified Swarm Optimization (SSO) algorithm for automated GUI functional testing. Engineering Science and Technology, an International Journal, 17(4): 218-226. http://dx.doi.org/10.1016/j.jestch.2014.06.001

[12] Ahmed, B.S., Zamli, K.Z., Lim, C. (2011). The development of a particle swarm based optimization strategy for pairwise testing. Journal of Artificial Intelligence, 4(2): 156-165. http://dx.doi.org/10.3923/jai.2011.156.165

[13] Ahmed, B.S., Zamli, K.Z., Lim, C.P. (2012). Application of particle swarm optimization to uniform and variable strength covering array construction. Applied Soft Computing, 12(4): 1330-1347. https://doi.org/10.1016/j.asoc.2011.11.029

[14] Prakash, V., Tatale, S., Kondhalkar, V., Bewoor, L. (2018). A critical review on automated test case generation for conducting combinatorial testing using particle swarm optimization. International Journal of Engineering and Technology (UAE), 7(3.8): 22-28.

[15] Bewoor, L.A., Chandra Prakash, V., Sapkal, S.U. (2017). Evolutionary hybrid particle swarm optimization algorithm for solving NP-hard no-wait flow shop scheduling problems. Algorithms, 10(4): 121. http://dx.doi.org/10.3390/a10040121

[16] Bewoor, L.A., Chandraprakash, V., Sapkal, S. (2019). Evolutionary hybrid particle swarm optimization algorithm to minimize makespan to schedule a flow shop with no wait. Journal of Engineering Science and Technology, 14(2): 609-628. https://doi.org/10.3390/a10040121

[17] Bewoor, L.A., Prakash, V.C., Sapkal, S.U. (2018). Production scheduling optimization in foundry using hybrid Particle Swarm Optimization algorithm. Procedia Manufacturing, 22: 57-64. http://dx.doi.org/10.1016/j.promfg.2018.03.010

[18] Bangare, S.L., Bangare, P.S. (2012). Automated testing in development phase. International Journal of Engineering Science and Technology, 4(2): 677-680.

[19] Bangare, S.L., Bangare, P.S., Borse, S., Nandedkar, S. (2012). Automated API testing approach. International Journal of Engineering Science and Technology, 4(2): 673-676.

[20] Bangare, S.L., Khare, A.R., Bangare, P.S. (2010). Code parser for object oriented software modularization. International Journal of Engineering Science and Technology, 2(12): 7262-7265.

[21] Bangare, S.L., Khare, A.R., Bangare, P.S. (2011). Quality measurement of modularized object oriented software using metrics. ICWET '11: International Conference & Workshop on Emerging Trends in Technology Mumbai Maharashtra India, pp. 771-774 https://doi.org/10.1145/1980022.1980190

[22] Bangare, S.L., Khare, A.R., Bangare, P.S. (2011). Measuring the quality of object oriented software Modularization: Defining metrics and algorithm. International Journal on Computer Science and Engineering, 3(1): 445-450.

[23] Tatale, S.B., Prakash, V.C. (2021). A survey on test case generation using UML diagrams and feasibility study to generate combinatorial logic oriented test cases. International Journal of Next-Generation Computing, 12(2): 254-269. https://doi.org/10.47164/ijngc.v12i2.193

[24] Tatale, S., Prakash, V.C. (2021). Generation of combinatorial logic oriented test cases from UML sequence diagram. Journal of Theoretical and Applied Information Technology, 99(21): 5201-5216.

[25] Tatale, S., Chandra Prakash, V. (2022). Combinatorial test case generation from sequence diagram using optimization algorithms. International Journal of System

Assurance Engineering and Management, 13: 642-657. http://dx.doi.org/10.1007/s13198-021-01579-w

[26] Tatale, S., Prakash, V.C. (2022). Automatic generation and optimization of combinatorial test cases from UML activity diagram using particle swarm optimization. Ingénierie des Systèmes d'Information, 27(1): 49-59. https://doi.org/10.18280/isi.270106

[27] Cohen, D.M., Dalal, S.R., Fredman, M.L., Patton, G.C. (1997). The AETG system: An approach to testing based on combinatorial design. IEEE Transactions on Software Engineering, 23(7): 437-444. http://dx.doi.org/10.1109/32.605761

[28] Lei, Y., Tai, K.C. (1998). In-parameter-order: A test generation strategy for pairwise testing. In Proceedings Third IEEE International High-Assurance Systems Engineering Symposium (Cat. No. 98EX231), Washington, DC, USA, pp. 254-261. https://doi.org/10.1109/HASE.1998.731623

[29] Arshem J (2010). TVG. Available: http://sourceforge.net/projects/tvg.

[30] Hartman A, Klinger T, Raski L (2010). IBM Intelligent Test Case Handler [Online]. Available: http://www.alphaworks.ibm.com/tech/whitch.

[31] Jenkins B (2010). Jenny Test Tool [Online]. Available: http://www.burtleburtle.net/bob/math/jenny.html.

[32] Pande, S. D., Chetty, M.S.R. (2018). Analysis of capsule network (Capsnet) architectures and applications. J Adv Res Dynam Control Syst, 10(10): 2765-2771.

[33] Pande, S., Chetty, M. (2019). Bezier curve based medicinal leaf classification using capsule network. International Journal of Advanced Trends in Computer Science and Engineering, 8(6): 2735-2742 https://doi.org/10.30534/ijatcse/2019/09862019

[34] Pande, S.D., Chetty, M.S.R. (2021). Fast medicinal leaf retrieval using CAPSNET. In International Conference on Intelligent and Smart Computing in Data Analytics, pp. 149-155. https://doi.org/10.1007/978-981-33-6176-8_16

[35] Poli, R., Kennedy, J., Blackwell, T. (2007). Particle swarm optimization. Swarm Intelligence, 1(1): 33-57. http://dx.doi.org/10.1007/s11721-007-0002-0

[36] Chen, X., Gu, Q., Qi, J., Chen, D. (2010). Applying particle swarm optimization to pairwise testing. In 2010 IEEE 34th Annual Computer Software and Applications Conference, Seoul, Korea (South), pp. 107-116. http://dx.doi.org/10.1109/COMPSAC.2010.17