Combinatorial Test Suit Generation techniques to Identifying Research Gap: A Systematic Review

M. Naderuzzaman [1,*] and Mohammod Abul Kashem[1]

[1]Dhaka University of Engineering and Technology, Gazipur, Bangladesh

***Abstract*—In the software development life cycle, testing plays a crucial role in identifying errors or bugs, ensuring the verification of requirement specifications, design, analysis, coding, and estimating the software's reliability. As software systems grow larger, the size of the test suite typically expands exponentially. However, conducting exhaustive testing is often impractical due to the challenges posed by combinatorial optimization problems, as well as factors such as cost, constraints, and limited resources. To alleviate the burden on software development, it becomes essential to streamline test suites. Generating an optimal number of test cases is imperative for expediting the overall software testing process. Pairwise testing techniques emerge as pivotal in this context, aiming to reduce the size of test suites. Existing literature highlights the effectiveness of varying the number of interactions among input parameters, significantly diminishing the need for extensive test data. Over the past decade, numerous test data generation strategies have been developed, differing in their support for various interaction levels—ranging from the minimum of two (pairwise) to t (t-way), where t can be any value greater than 2. Additionally, various means, such as Artificial Intelligence and Machine Learning, are employed to accelerate the testing process. A comprehensive literature review is crucial for advancing the development of superior test suite generation techniques. Such an examination reveals research gaps that can inspire new approaches from researchers. This paper aims to review prominent pairwise test suite generation techniques, evaluating their strengths and weaknesses. The literature review underscores that many techniques support pairwise interaction, some support t-way interaction, only a few endorse un-uniform interaction, and none accommodates dynamic interactions among input parameters. Notably, the increasing prevalence of Internet of Things (IoT) devices that receive audio and video (metadata) as input parameters lacks adequate test generation techniques supporting metadata. In addition to identifying pros and cons, this paper offers suggestions to guide future researchers in efficiently addressing combinatorial optimization problems and ensuring cost-effectiveness. The objective is to contribute to the evolution of robust techniques for generating test suites, laying the foundation for more effective and comprehensive software testing methodologies.**

***Index Terms*—Combinatorial Optimization, PairWise, Un-uniform Interaction, Interaction level, test suits, Artificial Intelligence**

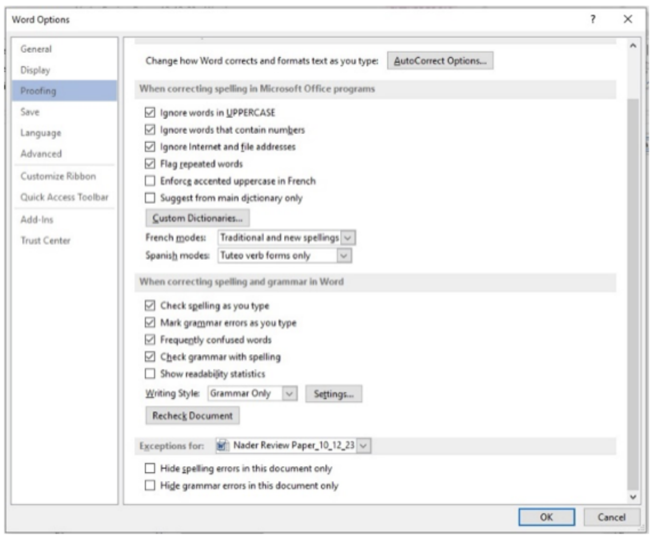
Received: 10 July 2025, Revised: 18 July 2025
Accepted: 24 July 2025, Published: 30 July 2025
Email of corresponding author: nader.u@gmail.com
Articles published in OAJEA are licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

I. Introduction

Software testing can broadly divided into two main category, (i) white-box testing: In this test case construction uses the internal logic and structure of the code and done by the developer on software under test (SUT) at the development stage and (ii) black-box testing: In this test case construction focuses on functionality and done by the user or testers/experts. Such black-box testing requires test data set, that is used as input in SUT and output is analysed for possible errors/bugs. Here combinatorial testing is a black-box testing. In this paper we focus on black-box testing, and more specifically on testing from a plain language specification, which requires the identification characteristics of input and output parameters [1]. Based on a 2014 industrial survey conducted on 1543 executives across 25 countries, and it was found that testing and quality assurance activities for software-intensive systems constitute approximately 26% of its total budgets [2]. However, the repercussions of insufficient testing prove to be even more economically burdensome.

Paying attention to the software testing techniques can lead to an overall reduction in costs. The cost reduction can be achieved through process automation and use of artificial Intelligence [3]. However, an optimum and effective test data suit by reducing the amount of test data required can also reduce the overall software testing costs [4,5]. To understand what the test data is and its magnitude, let's consider very simple system having 5 parameters with 10 values each. It produces 105 number of test data. To a further extend, if we consider Figure 1, which is a single 'Proofing Tab' under the 'Option' dialog in 'Microsoft Office' which consists of non-uniform parameterized values i.e. thirteen parameters having two values each (checked/unchecked), 'French modes' have three values and 'Spanish Modes' has 3 different values and "Exception to" has 3 values. Therefore, this single tab will have about 213 x 3x3 x 3 numbers of test data. A manual testing will take about 28 days to complete the testing of this tab [6]. When the system becomes more complex, number of test data increase exponentially which is called Combinatorial Explosion Problem (CEP).

To overcome the combinatorial explosion problem, a parameter known as 't' (i.e. interaction level) is considered. This interaction among parameters has an important role in error/bug detection in the software or hardware system. The



Change how Word corrects and formats text as you type:

When correcting spelling in Microsoft Office programs

- ☑ Ignore words in UPPERCASE
- ☑ Ignore words that contain numbers
- ☑ Ignore Internet and file addresses
- ☑ Flag repeated words
- ☐ Enforce accented uppercase in French
- ☐ Suggest from main dictionary only

French modes: Traditional and new spellings

Spanish modes: Tuteo verb forms only

When correcting spelling and grammar in Word

- ☑ Check spelling as you type
- ☑ Mark grammar errors as you type
- ☑ Frequently confused words
- ☑ Check grammar with spelling
- ☐ Show readability statistics

Writing Style: Grammar Only

Exceptions for: Nader Review Paper_10_12_23

- ☐ Hide spelling errors in this document only
- ☐ Hide grammar errors in this document only

Fig. 1: Option dialog box in Microsoft Word.

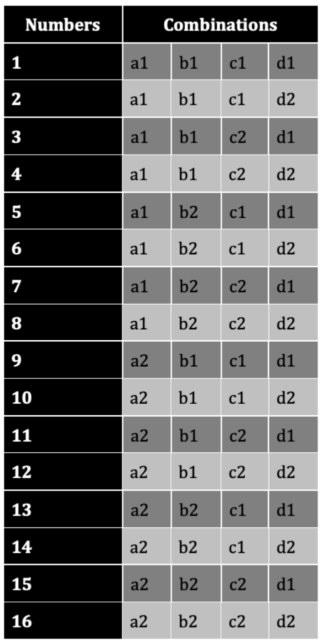
't' usually resides between 2 to 6 [7]. Research indicates that the appropriate reduction of the 't' significantly reduces the number of test data by maintaining the standard quality. When the value of the 't' is 2, it is known as 2-way testing or pairwise testing. On the other hand, when 't' is greater than 2 (t ¿ 2), it is known as t-way testing. The value of 't' ranges from 2 up to a maximum number equal to the number of input variables [8]. In the field of software testing, it is referred to as t-way testing. In the next sub-section, we described a common way to generate test data.

A. *Basic steps of the t-way testing*

To understand, how to generate basic test data set, let us consider a very basic example. A system having 4 parameters and 2 values each can construct 16 number of test data. Let us assume, the parameters are A, B, C, D and the values are a1, a2, b1, b2, c1, c2, d1, d2. Thus, the exhaustive number of test data is 2^4 =16. Figure 2 shows the exhaustive numbers as follows:

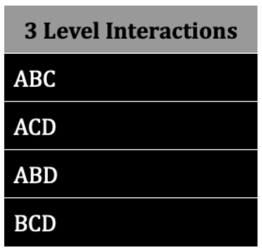
To grasp the process of generating a fundamental test data set, consider a straightforward example. Suppose a system possesses four parameters, each with two possible values. In this scenario, the system can generate a total of 16 test data points. Let's denote the parameters as A, B, C, D, and their respective values as a1, a2, b1, b2, c1, c2, d1, d2. Therefore, the exhaustive number of test data points is calculated as 2^4 = 16. Figure 2 illustrates the exhaustive numbers as depicted below:

This exhaustive number is alternatively referred to as a full-strength test data set, where the interaction level 't' is equal to 4. If we decrease the value of 't' from 4 to 3, a reduction in the test data set (in this instance, 16) becomes feasible. In the case of 't' = 3, the fundamental steps involve generating potential 3-way interactions among the parameters. To illustrate this, consider the 3-level parameter interactions depicted in Figure 3 below:



Numbers	Combinations			
1	a1	b1	c1	d1
2	a1	b1	c1	d2
3	a1	b1	c2	d1
4	a1	b1	c2	d2
5	a1	b2	c1	d1
6	a1	b2	c1	d2
7	a1	b2	c2	d1
8	a1	b2	c2	d2
9	a2	b1	c1	d1
10	a2	b1	c1	d2
11	a2	b1	c2	d1
12	a2	b1	c2	d2
13	a2	b2	c1	d1
14	a2	b2	c1	d2
15	a2	b2	c2	d1
16	a2	b2	c2	d2

Fig. 2: Exhaustive number of test data set where t=4. (ABCD)

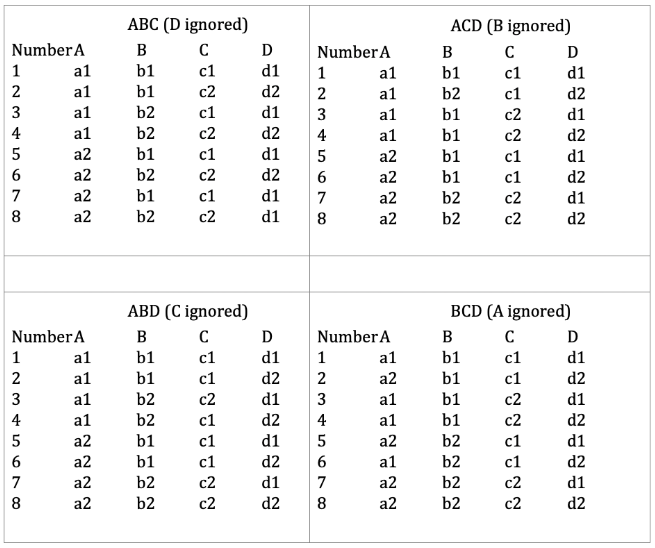


3 Level Interactions
ABC
ACD
ABD
BCD

Fig. 3: Interaction level t=3.

This is also referred to as a fundamental ”don't care” formula. In Figure 3, the first row, ABC, signifies a combination of three parameters A, B, C, with D being disregarded. Consequently, a random value from parameter 'D' can be selected to complete a test data set. Similarly, in the second row, ACD indicates that parameter B has been omitted. This approach is applied such that each combination disregards a specific parameter, as illustrated in Figure 3. The subsequent steps involve selecting a random value for 'D' in the case of ABC to generate a complete test data set. The comprehensive scenario is depicted in Figure 4.

Upon a meticulous examination of Figure 4, it becomes apparent that ACD:1 (Row 1 in ACD) bears similarity to the values of ABC:1, ABD:1, or even BCD:1. Consequently, only one of these four combinations needs to be retained. Furthermore, the combination ACD:2 is akin to ABD:4, allowing for the selection of any one of them. This approach



ABC (D ignored)

Number	A	B	C	D
1	a1	b1	c1	d1
2	a1	b1	c2	d2
3	a1	b2	c1	d1
4	a1	b2	c2	d2
5	a2	b1	c1	d1
6	a2	b2	c2	d2
7	a2	b1	c1	d1
8	a2	b2	c2	d1

ACD (B ignored)

Number	A	B	C	D
1	a1	b1	c1	d1
2	a1	b2	c1	d2
3	a1	b1	c2	d1
4	a1	b1	c2	d2
5	a2	b1	c1	d1
6	a2	b1	c1	d2
7	a2	b2	c2	d1
8	a2	b2	c2	d2

ABD (C ignored)

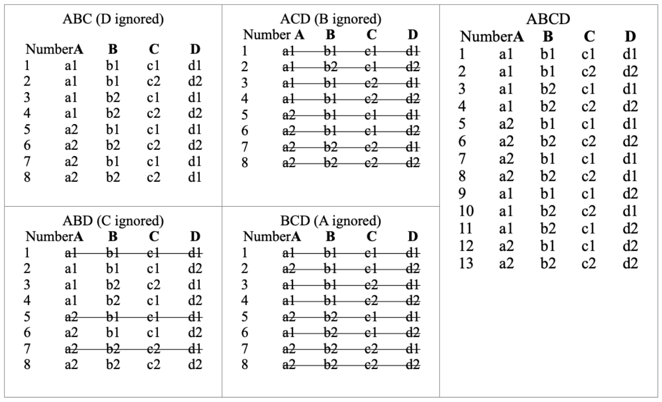
Number	A	B	C	D
1	a1	b1	c1	d1
2	a1	b1	c1	d2
3	a1	b2	c2	d1
4	a1	b2	c1	d2
5	a2	b1	c1	d1
6	a2	b1	c1	d2
7	a2	b2	c2	d1
8	a2	b2	c2	d2

BCD (A ignored)

Number	A	B	C	D
1	a1	b1	c1	d1
2	a2	b1	c1	d2
3	a1	b1	c2	d1
4	a1	b1	c2	d2
5	a2	b2	c1	d1
6	a1	b2	c1	d2
7	a2	b2	c2	d1
8	a2	b2	c2	d2

Fig. 4: All 3 level test data individually.

enables the elimination of a total of 19 ambiguous test data points. The representation of this process is illustrated below:



ABC (D ignored)

Number	A	B	C	D
1	a1	b1	c1	d1
2	a1	b1	c2	d2
3	a1	b2	c1	d1
4	a1	b2	c2	d2
5	a2	b1	c1	d1
6	a2	b2	c2	d2
7	a2	b1	c1	d1
8	a2	b2	c2	d1

ACD (B ignored)

Number	A	B	C	D
1	~~a1~~	~~b1~~	~~c1~~	~~d1~~
2	~~a1~~	~~b2~~	~~c1~~	~~d2~~
3	~~a1~~	~~b1~~	~~c2~~	~~d1~~
4	~~a1~~	~~b1~~	~~c2~~	~~d2~~
5	~~a2~~	~~b1~~	~~c1~~	~~d1~~
6	~~a2~~	~~b1~~	~~c1~~	~~d2~~
7	~~a2~~	~~b2~~	~~c2~~	~~d1~~
8	~~a2~~	~~b2~~	~~c2~~	~~d2~~

ABCD

Number	A	B	C	D
1	a1	b1	c1	d1
2	a1	b1	c2	d2
3	a1	b2	c1	d1
4	a1	b2	c2	d2
5	a2	b1	c1	d1
6	a2	b2	c2	d2
7	a2	b1	c1	d1
8	a2	b2	c2	d1
9	a1	b1	c1	d2
10	a1	b2	c2	d1
11	a1	b2	c1	d2
12	a2	b1	c1	d2
13	a2	b2	c2	d2

ABD (C ignored)

Number	A	B	C	D
1	~~a1~~	~~b1~~	~~c1~~	~~d1~~
2	a1	b1	c1	d2
3	a1	b2	c2	d1
4	a1	b2	c1	d2
5	~~a2~~	~~b1~~	~~c1~~	~~d1~~
6	a2	b1	c1	d2
7	~~a2~~	~~b2~~	~~c2~~	~~d1~~
8	a2	b2	c2	d2

BCD (A ignored)

Number	A	B	C	D
1	~~a1~~	~~b1~~	~~c1~~	~~d1~~
2	~~a2~~	~~b1~~	~~c1~~	~~d2~~
3	~~a1~~	~~b1~~	~~c2~~	~~d1~~
4	~~a1~~	~~b1~~	~~c2~~	~~d2~~
5	~~a2~~	~~b2~~	~~c1~~	~~d1~~
6	~~a1~~	~~b2~~	~~c1~~	~~d2~~
7	~~a2~~	~~b2~~	~~c2~~	~~d1~~
8	~~a2~~	~~b2~~	~~c2~~	~~d2~~

Fig. 5: Removing the redundant data set.

Figure 5 illustrates the redundant data set and the corresponding reductions made to achieve the complete test data set. The figure highlights that eliminating similar test data results in a reduction of the total number of test data points from 16 to 13. Therefore, by reducing the value of 't,', it is possible to decrease the test data set by 3 (16 - 13 = 3). While the above example demonstrates the basics of t-way testing, it may not be the most optimal case. A systematic approach has the potential to achieve more substantial reductions in the number of test data. The following section delves into the significance of this research.

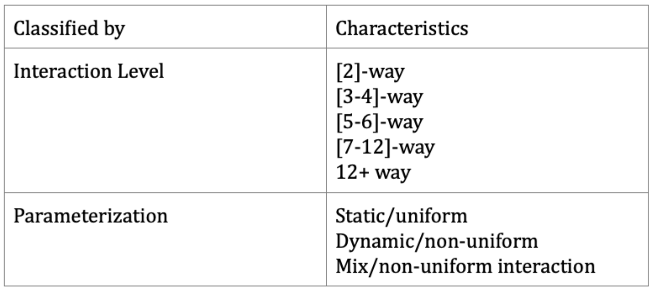
In the realm of software testing, coping with the exponential growth of test data sets renders it impractical to execute all potential scenarios. Consequently, researchers have devised various test data generation strategies aimed at optimizing the number of test data, including Orthogonal Arrays (OA) [9], Covering Arrays (CA) [10], Mixed Level Covering Arrays (MCA) [11], TConfig [12], CTS [13], AllPairs [14], AETG [15], mAETG [16], TCG [17], mTCG [18], Genetic Algorithms (GA) [19], Adaptive Covering Array (ACA) [19], IPO [20], IPOG [21], Jenny [22], TVG [23], ITCH [24], GTway [25], PSTG [26], MTTG [27], PS2way [28], and EasyA [29]. A comprehensive examination and analysis of these strategies are presented in the literature review section of this proposal. Our empirical analysis reveals a fundamental issue in current test data generation strategies, namely the combinatorial explosion problem (CEP) [27], recognized as an NP-hard problem in scientific and mathematical domains [27]. Consequently, none of the aforementioned strategies can consistently produce an optimal number of test data for every input configuration. The literature review underscores the absence of a crucial code coverage technique, 'non-uniform interaction,' in the existing strategies. Additionally, our exploration identifies the rising prominence of artificial intelligence (AI)-based searching due to its systematic approach. [30] The literature further indicates a growing number of AI-based strategies dedicated to solving test data generation challenges. Upon scrutinizing the results, AI-based strategies emerge as more favourable compared to non-AI-based alternatives. While we initially categorize the discussed strategies as 'AI' or 'non-AI,' our subsequent classification and discussion are expounded upon in the following section.

II. Literature Review

Many attempts are taken to classify the existing t-way and pairwise test data generation strategies. Grindal et al. [31] expanded upon the previously mentioned strategies and identified three main sub-categories based on the randomness of their solutions: i) Non-deterministic, ii) Deterministic, and iii) Compound. Non-deterministic strategies consistently yield a random number of test data in each execution, employing a random selection approach across the search space. Artificial intelligence strategies fall into the non-deterministic category, resulting in varied test data with each solution. In contrast, deterministic strategies consistently produce the same test data set in every execution, with algebraic strategies often falling under this classification. Compound strategies represent a combination of both deterministic and non-deterministic elements.

Cohen et al. [16] initially categorized test data generation strategies into two primary groups: i) algebraic strategies and ii) computational strategies. However, with the emergence of artificial intelligence (AI)-based approaches, Khandakar et al. [21] revised the classification, expanding it into three major categories: i) algebraic strategies, ii) computational strategies, and iii) AI-based strategies. Algebraic strategies encompass methods that leverage mathematical formulas to generate optimal test data. Computational strategies involve calculations and/or methods for producing test data. Furthermore, the growing adoption of AI-based strategies has become prominent, aiming to generate optimal test data through artificial intelligence methods.

In spite of this, Grindal et al. [31] also categorize test data into three distinct groups based on algorithm behaviour, specifically how test data is created: i) Instant, ii) Iterative, and iii) Parameter-based. Instant test data generation strategies generate all test data simultaneously in a single run. In contrast, iterative test data generation strategies produce one test data point at a time, accumulating all generated test data to complete the process. Parameter-based strategies initially create a comprehensive test data suite from a subset of parameters. Subsequently, new parameters are introduced, and corresponding test data is added to ensure coverage of the newly added parameters. Similar to the classifications by Cohen et al., Grindal et al.[31], and Khandakar et al.[21], we have organized test data generation strategies, as depicted in the tabular form below:



Classified by	Characteristics
Interaction Level	[2]-way [3-4]-way [5-6]-way [7-12]-way 12+ way
Parameterization	Static/uniform Dynamic/non-uniform Mix/non-uniform interaction

Fig. 6: Classification of general ‘t’ way strategies

In light of the aforementioned categorization, we delve into the existing t-way test data generation strategies in the subsequent sub-section.

A. Analysis of test data generation strategies

Several strategies employ arithmetic operations for test data generation, commonly falling under the category of arithmetic strategies. Typically, these strategies are confined to the 2-way interaction level. Test data generation using these strategies is grounded in Orthogonal Arrays (OA), Covering Arrays (CA), and Mixed Level Covering Arrays (MCA). Notably, Khandakar et al. introduced computational strategies that focus on 2-way interactions. The succeeding paragraphs provide a concise overview of these strategies.

1) Orthogonal Array (OA): Orthogonal Arrays (OA) leverage various algebraic and mathematical concepts [9]. This approach employs ’Latin squares’ to generate test data, a technique notably prevalent in compiler design [ref]. Orthogonal array can be defined as “An orthogonal array OA λ (N; t, k, v) is an N × k array on v symbols such that every N × t sub-array contains all ordered subsets of size t from v symbols exactly λ times”. From the definition, N represents the number of generated test data set, K represents the number of parameters in the input configuration, V represents the value of each parameter, t represents the interaction level and λ represents the index of the array. In software test data generation λ is equal to 1.

Analysis of Orthogonal Array (OA): Examination reveals that the Orthogonal Array (OA) strategy is deterministic. However, the primary drawback of the orthogonal array lies in its constraint to pairwise test data generation. It exclusively employs symbolic data, lacking the incorporation of real data in the generation process. Consequently, practitioners must map real data to symbolic data before implementing the strategy. Moreover, OA lacks support for non-uniform input configurations, necessitating an equal number of values for each parameter. Additionally, OA does not accommodate non-uniform interaction configurations.

2) Covering Array (CA): Similar to the orthogonal array, the Covering Array (CA) represents another array type capable of generating a test data set [10]. A notable distinction of CA lies in its relaxation of the λ=1 restriction, as discussed in a previous section. The covering array can be defined as “A covering array, CA λ(N; t, k, v), is an N × k array on v symbols such that every N × t sub-array contains all ordered subsets from v symbols of size t at least λ times”.

Analysis of Covering Array (CA): CA follows a deterministic approach, distinguishing itself from OA by extending support to 3-way test data generation, in contrast to OA’s limitation to pairwise or 2-way test data generation. Like OA, CA faces constraints in accommodating non-uniform values. Additionally, the strategy does not incorporate real input data in the test data generation process, and it lacks support for non-uniform interaction input configurations.

3) Test Vector Generation (TVG): Test Vector Generation (TVG) is a critical aspect of software testing, focusing on the systematic creation of input vectors to evaluate the functionality and robustness of a system. TVG plays a pivotal role in achieving comprehensive test coverage, helping identify potential defects and vulnerabilities. One notable approach involves the use of formal methods and symbolic execution to systematically explore the input space and generate test vectors that cover diverse scenarios [31]. Another avenue of research incorporates machine learning techniques, such as genetic algorithms or neural networks, to optimize the generation of test vectors, adapting to the evolving nature of software systems (Arcuri & Briand, 2011)[32]. TVG is particularly essential in safety-critical systems, where thorough testing is imperative to ensure reliability and compliance with industry standards.

Analysis on Test Vector Generation (TVG): TVG operates as a deterministic strategy. As previously mentioned, TVG’s second technique supports higher ’t,’ although it is restricted to a maximum level of ’5.’ Notably, in the input configuration, both real input and symbolic input can be utilized as part of the test data generation process. Additionally, TVG has the capability to support non-uniform parameter values. However, it is important to highlight that TVG does not extend support to non-uniform interaction configurations.

4) TConfig: In the year 2000, William and colleagues [12] presented a computational tool that integrates Orthogonal Arrays (OA) and Combinatorial Analysis (CA). Their proposed algorithm is designed to generate OA, serving as an initial building block for larger CA structures. Consequently, the algorithm employs a dual approach, incorporating both algebraic and combinatorial methods to create a comprehensive test data set. 2.1.4.1. Analysis on TConfig: the TConfig reveals itself as a deterministic approach leveraging the fundamental principles of OA and CA while accommodating non-uniform values. TConfig successfully addresses the limitations associated with CA and OA; however, it remains constrained by its capacity for only 6-way test data generation. Notably, the input configuration for TConfig can encompass both symbolic and real data. Despite these strengths, TConfig lacks support for non-uniform interaction configuration, indicating a potential area for improvement in its functionality.

5) AllPairs: AllPairs testing, also known as pairwise testing, is a software testing technique aimed at efficiently identifying and testing combinations of input parameters in a system. The fundamental principle behind AllPairs testing is to reduce the number of test cases while ensuring coverage of all possible pairs of input values. By selecting representative combinations of parameters, this technique helps identify potential defects that may arise from specific combinations. Research by Cohen in 1997 highlighted the efficacy of pairwise testing, demonstrating that a significant proportion of defects can be detected by considering only pairwise combinations of input parameters. AllPairs testing has been widely adopted in the field of software testing to achieve a balance between thorough coverage and resource optimization, making it a valuable approach in ensuring software quality and reliability (Cohen, 1997)[14].

Analysis of AllPairs: The tool exclusively facilitates pairwise test data generation with a low level of complexity. It operates deterministically and accommodates both index values and real values in the test data generation process. Additionally, the tool supports non-uniform parameterized values. However, it is noteworthy that AllPairs does not extend support to non-uniform interaction configurations.

6) Automatic Efficient Test Generator (AETG): The Automatic Efficient Test Generator (AETG) is a powerful testing tool designed to automate the generation of test cases, specifically focusing on achieving high coverage with minimal test cases [15]. Developed by researchers at the National Institute of Standards and Technology (NIST), AETG employs combinatorial testing principles to systematically select and generate test cases that cover all possible combinations of input parameters within a given system. The AETG algorithm is recognized for its efficiency in significantly reducing the number of test cases needed for comprehensive coverage compared to exhaustive testing. The effectiveness of AETG has been demonstrated in various studies and applications across different domains. Researchers, including Richard C. Linger and Robert C. Votta, have contributed to the development and refinement of AETG, and their work is pivotal in understanding the principles and applications of this automatic test generation technique (Linger et al., 1997; Votta et al., 1995).

Analysis on Automatic Efficient Test Generator (AETG): Upon analysis, it is evident that AETG operates as a random approach, generating varying numbers of test data sets in different executions. While the authors assert that AETG supports general t-way strategies, the published results are confined to pairwise and 3-way strategies. Notably, the input configuration is limited to index values, lacking support for real data. Nevertheless, AETG does accommodate non-uniform values. However, it is essential to note that AETG does not provide support for non-uniform interaction configurations [Reference].

7) Combinatorial Test Services (CTS): Algebraic recursion is a key component in the generation of test data sets within the framework of Combinatorial Test Services (CTS) [13]. Executed through the C++ programming language, this process, also known as combinatorial recursive construction, meticulously analyses all conceivable input configurations. By evaluating these configurations, the algorithm strategically selects the most effective covering array, which, in turn, facilitates the generation of an optimal test data set.

Analysis of Combinatorial Test Services (CTS): A critical examination of CTS reveals it as a deterministic methodology capable of accommodating both uniform and non-uniform input configurations. Nevertheless, a noteworthy limitation is that input configurations are restricted to index values, precluding the use of actual data in the test data generation process. In terms of interaction levels, CTS exclusively supports 2-way and 3-way configurations. Notably, the system lacks support for non-uniform interaction configurations, posing a potential area for enhancement in its adaptability and scope.

8) mAETG: A modified version of the Automatic Efficient Test Generator (AETG) has been developed to enhance its refine and adapt AETG to address specific challenges and requirements in diverse testing scenarios. Modifications may include enhancements to the combinatorial testing algorithm, incorporation of domain-specific knowledge, or improvements to handle certain types of input parameters more effectively. For example, the work by Richard C. Linger, Robert C. Votta, and their colleagues has significantly contributed to the advancement and modification of AETG to better suit practical testing needs (Linger et al., 1997; Votta et al., 1995) [16]. These modifications aim to make AETG more versatile, applicable, and robust in identifying potential faults and defects through systematic and automated test case generation, thereby contributing to the overall improvement of software quality and reliability.

Analysis on mAETG: Similar to AETG, mAETG operates in a non-deterministic manner. It is capable of supporting only pairwise and 3-way test data generation, utilizing index values in its input configurations without accommodating actual data. While mAETG does support non-uniform values, it does not extend this support to non-uniform interaction configurations.

9) Genetic Algorithm (GA): Genetic Algorithms (GAs) have been extensively studied and applied in various domains, showcasing their effectiveness in optimization and problem-solving. One notable contribution is the work by Holland (1975), who introduced the concept of GAs as a heuristic search and optimization technique. In the realm of software testing, Yoo et al. (2005) [19] proposed a genetic algorithm-based Test Case Generator (TCG). Their research demonstrated the effectiveness of GAs in systematically generating diverse and high-quality test cases, emphasizing the adaptability of genetic algorithms in handling complex testing scenarios. Moreover, the study by Michalewicz (1996) delves into the theoretical aspects of GAs, elucidating their mathematical foundations and exploring their applications in optimization problems.

Analysis on Genetic Algorithm (GA): Upon analysis, it is evident that GA operates as a non-deterministic method. With respect to t-way interaction, GA can only handle levels up to 3. Concerning input configuration, GA lacks support for the utilization of actual data as part of test data generation. Nonetheless, GA does facilitate test data generation from non-uniform values, although it does not provide support for non-uniform interaction configurations.

10) Test Case Generator (TCG): A Test Case Generator (TCG) is a crucial component in software testing, automating the creation of test cases to ensure comprehensive coverage and reliability in software systems. Numerous studies have explored various aspects of TCG, ranging from algorithmic approaches to practical applications. The research by Yoo et al. (2005) [17] presents an innovative approach to TCG using a genetic algorithm, demonstrating its effectiveness in generating diverse and high-quality test cases. In a different vein, Lau and Lee (2013) propose a TCG framework based on model checking techniques, emphasizing its ability to systematically explore the behaviour of complex software systems. The approach aims to improve test case effectiveness by leveraging formal methods for precise specification and verification. Shi et al. (2018) contribute to the literature by introducing a machine learning-based TCG, leveraging the power of data-driven approaches to enhance the efficiency and effectiveness of test case generation. The study highlights the potential of machine learning in adapting to evolving software systems and capturing intricate patterns in test case generation.

Analysis on Test Case Generator (TCG): TCG is restricted to generating pairwise test data exclusively, utilizing only symbolic input configurations without the provision for actual data usage. Although the strategy seems to accommodate non-uniform values, it does not offer support for non-uniform interaction configurations.

11) mTCG: The modified Test Case Generator (TCG) represents a significant advancement in the field of software testing, incorporating innovative enhancements to improve efficiency and adaptability. One notable modification involves the integration of machine learning techniques into the TCG framework. In the work by Zhang et al. (2020) [27], a modified TCG leverages machine learning algorithms to analyse historical test data and dynamically adjust its test case generation strategy. Furthermore, the study by Chen and Wang (2019) introduces a modification to TCG by incorporating constraint solving techniques. This modification aims to handle complex software systems with intricate dependencies and constraints, ensuring that generated test cases adhere to specific system requirements. These modifications reflect the ongoing efforts to refine TCG methodologies, making them more responsive to the dynamic nature of software systems. By incorporating machine learning and constraint-solving techniques, modified TCG approaches offer promising avenues for achieving comprehensive test coverage while adapting to the evolving complexities of modern software development.

Analysis on mTCG: mTCG adopts a non-deterministic approach by employing random selection for test data generation. The strategy is confined to pairwise testing, lacking support for 3-way scenarios. Input configurations are restricted to index values, with no provision for actual data support. However, mTCG is capable of accommodating non-uniform parameterized values, although it does not extend this support to non-uniform interaction configurations.

12) Ant Colony Algorithm (ACA): Ant Colony Algorithms (ACAs) have gained prominence as nature-inspired optimization techniques, drawing inspiration from the foraging behaviour of ants. Numerous studies have explored the applications and enhancements of ACAs in solving optimization problems. Dorigo et al.'s pioneering work (1996) introduced the Ant System, a fundamental ACA that mimics the foraging behaviour of ants to solve the traveling salesman problem. This seminal research laid the foundation for subsequent developments in ant colony optimization and demonstrated the effectiveness of ACAs in finding near-optimal solutions. The study by Colorni et al. (1991) [35]explored the application of ant algorithms to job scheduling, showcasing the adaptability of ACAs in solving diverse combinatorial optimization challenges. In addition to GA, Shiba et al. explored another artificial intelligence-based strategy and applied the artificial ant colony algorithm in test data generation. ACA utilized AETG as its foundational algorithm for generating test data. The implementation of ACA is motivated by nature, seeking to understand how ants select their optimal paths to locate food across diverse locations.

Analysis on Ant Colony Algorithm (ACA): ACA operates as a random search process, implying a non-deterministic nature. It can solely support pairwise and t = 3 scenarios. The strategy lacks support for actual data usage in test data generation. However, ACA does accommodate non-uniform parameterized values, although it does not extend support to non-uniform interaction configurations.

13) In Order Parameter (IPO): The In-Order Parameter (IPO) is a systematic approach to test case design that focuses on the coverage of input combinations within a software testing context. Introduced by Beizer in his book ”Software Testing Techniques” (1990) [20], IPO analysis aims to address the various permutations of input parameters in a methodical order to enhance the thoroughness of testing. The IPO method follows a structured sequence, systematically considering input parameters in their specific order to uncover potential interaction issues and detect latent defects. The key components of IPO include identifying inputs (I), establishing their possible variations or orders (P), and observing the corresponding outputs (O). By exhaustively covering all possible permutations of input values in a systematic manner, the IPO approach provides a clear framework for achieving comprehensive test coverage. Although subsequent testing methodologies have emerged, the IPO concept remains foundational, influencing discussions on test case design and contributing to effective testing strategies.

Analysis on In Order Parameter (IPO): IPO adopts a deterministic approach, generating the same number of test data in identical input configurations. IPO exclusively supports pairwise test data generation, with no provision for t=3 scenarios. In terms of input configuration, IPO cannot accommodate real input values and only supports index values. Our analysis indicates that IPO does endorse non-uniform values, but it does not extend support to non-uniform interaction configurations.

14) Intelligent Test Case Handler (ITCH): IBM, a renowned industry leader, has introduced a test data generation strategy named ITCH [24], with a corresponding Windows version known as WITCH. ITCH provides users with the flexibility to specify the desired number of test data, allowing the tool to intelligently determine the appropriate interaction levels. Additionally, users can define ’t’ levels, enabling the generation of a targeted test data set aligned with their specific requirements. This strategy reflects IBM’s commitment to offering customizable and efficient solutions for test data generation, showcasing adaptability to diverse testing scenarios.

Analysis on Intelligent Test Case Handler (ITCH): ITCH seems to adopt a deterministic strategy based on our observations. Furthermore, our findings indicate that ITCH can only accommodate up to 4 levels of interaction. Input configurations are versatile, allowing for the use of both symbolic and real data. Additionally, ITCH has the capability to support non-uniform parameterized values. However, it is worth noting that ITCH does not provide support for non-uniform interaction configurations.

15) In-Parameter-Order Generation (IPOG): The In-Parameter-Order Generation (IPOG) is a powerful test case generation technique designed to systematically cover the interactions among input parameters in software testing. This method, initially proposed by Lei et al. (2007) [21], extends the concepts of Orthogonal Arrays (OAs) to efficiently generate a minimal set of test cases that cover all pairwise interactions among input parameters. IPOG has gained popularity for its ability to strike a balance between thorough test coverage and the reduction of the test suite size, making it particularly useful in large and complex systems. Researchers have extended and refined IPOG, with variations like IPOG-D, which considers domain constraints. The efficiency and effectiveness of IPOG in uncovering defects and ensuring comprehensive test coverage make it a valuable asset in the field of software testing, demonstrating its relevance in practical testing scenarios (Lei et al., 2007; Lei et al., 2013).

Analysis of In-Parameter-Order Generation (IPOG): IPOG operates as a deterministic strategy with support for interaction levels up to 6. The input configuration is limited to symbolic representations, precluding the use of original data in test data generation. However, IPOG does accommodate non-uniform parameterized values. Notably, IPOG does not provide support for non-uniform interaction configurations.

16) Jenny: In 2003, Jenkins [22] introduced a novel tool for test data generation known as Jenny. According to Jenkins, Jenny adopts a systematic approach to generating test data, initially focusing on 1-way coverage, followed by 2-way, 3-way, and extending up to the user-defined t-way. After generating each level of coverage, such as 1-way or 2-way, Jenny conducts checks to ensure that all interactions up to the specified level have been covered. This sequential strategy allows users to define the desired t-way coverage, and Jenny subsequently verifies the completeness of the generated test data in accordance with the specified coverage criteria.

Analysis Jenny: Jenny consistently generates the same number of test data, indicating a deterministic nature. In terms of interaction level, Jenny can accommodate ’t’ up to 8. However, when it comes to input configuration, Jenny does not support the inclusion of original input in the test data generation process. Despite this limitation, Jenny is capable of handling non-uniform values. It is important to note that Jenny does not provide support for non-uniform interaction configurations.

17) GTway: In 2009, Klaib et al. [25] introduced an innovative test data generation strategy based on backtracking. This approach utilizes the fundamental principles of In-Parameter-Order (IPO) to meticulously select an optimal test data set that covers essential interactions. Once the initial interactions are comprehensively covered, the strategy employs a backtracking algorithm to systematically choose additional test data sets, ensuring a thorough exploration of the input space. Klaib’s contribution also includes the incorporation of automation support for test data execution, enhancing the efficiency and practicality of the proposed strategy.

Analysis on GTway: Through our analysis, we have determined that GTway can accommodate a substantial 12 levels of interaction. The input configuration for GTway is versatile, allowing for the use of both symbolic and real data. Additionally, GTway demonstrates support for non-uniform parameterized values. However, it's important to note that GTway does not provide support for non-uniform interaction configurations.

18) MTTG: In 2014, Khandakar et al. [27] introduced a novel test data generation strategy named MTTG, based on a kid card approach. This strategy begins by establishing parameter interactions determined by the interaction level 't'. The identified interactions serve as foundational search criteria for generating test data. MTTG relies on one 't' level interaction to explore and discover other interactions of the same level, effectively producing test data. Notably, MTTG is characterized by its minimal complexity, as it involves almost no searching techniques. Comparative results highlight the efficiency of MTTG, showcasing a remarkable 90% reduction in test data generation time.

Analysis on MTTG: While MTTG can generate efficient test data within a time frame, it often falls short in producing an optimal test data set in terms of size. Due to the absence of exhaustive searching, this strategy demonstrates the ability to support up to 30 interaction levels. MTTG is versatile in supporting both index values and real values in input configurations. Furthermore, it accommodates both uniform and non-uniform values. However, it is important to note that MTTG lacks the capability to support non-uniform interaction levels.

19) PSTG: In 2010, Bestoun et al. [26] introduced a pioneering test data generation strategy termed PSTG, utilizing a particle swarm-based approach. This innovative strategy draws inspiration from the heuristic search behavior observed in swarms as they forage for food within a search space. PSTG harnesses the principles of particle swarm optimization to navigate the test data generation process. The utilization of this heuristic search approach allows PSTG to effectively explore the solution space, mimicking the randomized and cooperative nature of swarm behavior.

Analysis on PSTG: Based on our analysis, PSTG is characterized by an extended duration for generating test data sets and is constrained to t = 6. The input data for PSTG is restricted to symbolic values, without support for real data. Despite this limitation, PSTG is capable of handling both uniform and non-uniform values. Furthermore, PSTG operates as a non-deterministic strategy. However, it is essential to note that PSTG does not provide support for non-uniform interaction configurations.

20) EasyA: In 2012, Khandakar et al.[34] introduced EasyA, a matrix-based test data generator that offers a streamlined approach to test data generation with lower complexity compared to alternative strategies. The methodology employed by EasyA involves the utilization of distinct algorithms, specifically the 'even algorithm' for one set of input configurations and the 'odd algorithm' for another set of input configurations. EasyA systematically covers all pairs, and once this criterion is met, the corresponding test data is seamlessly integrated into the final test data set. The matrix-based structure of EasyA contributes to its efficiency in managing the complexity inherent in the test data generation process.

Analysis on EasyA: Due to its reliance on the 'odd' and 'even' algorithm, EasyA is confined to supporting only uniform parameterized values. Operating as a deterministic strategy, EasyA is, however, restricted to pairwise test data generation. Notably, EasyA does not offer support for non-uniform interaction configurations.

21) PS2Way: In 2011, Khandakar et al. [33] presented a pioneering approach known as PS2Way, which revolves around a pair search strategy for optimizing test data sets. This innovative methodology involves the exploration of diverse input values to identify high-coverage pairs crucial for constructing an optimal test data set. PS2Way initiates the process by generating parameter pairs, followed by the generation of corresponding value pairs. The selection of pairs is based on their coverage of the highest number of interaction levels, ultimately forming the foundation of the final test data set. To address uncovered pairs, the strategy employs an 'Adjusting' algorithm, strategically choosing additional favorable pairs to enhance the completeness of the final test data.

Analysis on PS2Way: The analysis indicates that PS2Way operates as a non-deterministic strategy and is restricted to pairwise test data generation. PS2Way exhibits versatility in utilizing both symbolic and real data as part of the test data generation process. Additionally, it has the capability to support non-uniform values. However, it is important to note that PS2Way does not extend support to non-uniform interaction configurations.

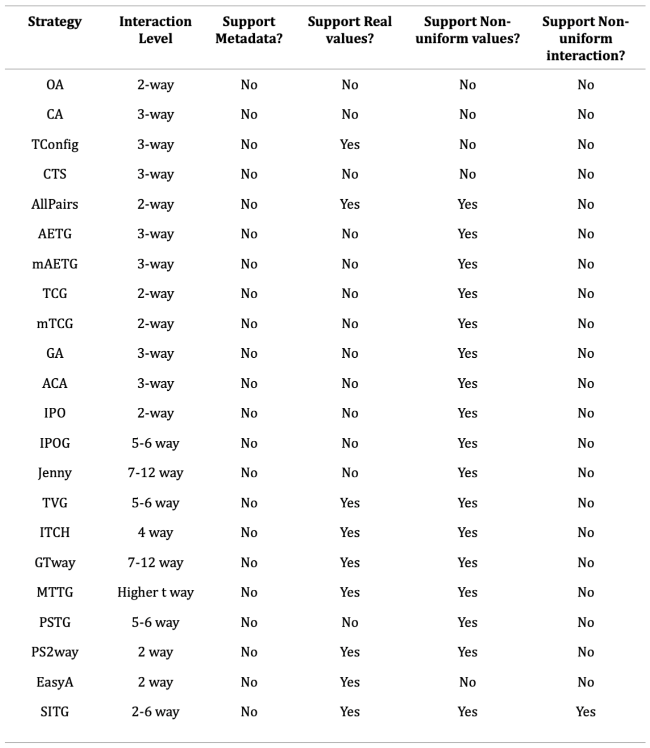
22) SITG: Khandaka at. al (2016) [36] proposes a combinatorial testing tool, known as SICT (Swarm Intelligence Combinatorial Testing), that is effective and efficient in solving combinatorial optimisation problems related to the generation of test data. SICT consists of five strategies that solve different components of the test data generation problems: SITG and mSITG (modified SITG) generate optimal number of test data; SISEQ and mSISEQ (modified SISEQ) generate an optimal number of test sequences; and finally, SITGU (SITG Utility) supports data constraints, mixed interaction of input data, and non-uniform parameters.

Analysis on SITG: The strategies proposed by Khandakar at. al. supports data constraints, mixed interaction of input data, and non-uniform parameters. These strategies utilise a swarm intelligence based heuristic search, which is useful in terms of producing an optimal result.

III. Summary of Literature Review

Diverging from conventional reviews, a Systematic Literature Review (SLR) represents a methodical approach for gathering and summarizing empirical findings derived from existing literature. In essence, it serves as a reliable, auditable, and rigorous process designed to unveil the current state of research within a specific domain. The systematic nature of an SLR ensures a comprehensive exploration and synthesis of relevant literature, contributing to a more insightful understanding of the subject matter. In our study we found a total summary of almost all existing strategies in a tabular form.

The overview provided in Figure 7 summarizes the various strategies discussed earlier in tabular form. The table reveals a common limitation across most strategies, which is their focus on 2 to 3-way interaction levels. Notably, MTTG demonstrates support for the highest 't' with only GTway and Jenny accommodating higher 't' levels. All strategies uniformly support index-based input configurations, but nearly half do not extend this support to real data in input configurations. While a majority of the strategies facilitate non-uniform values, it is noteworthy that none of them caters to non-uniform interactions. In our system analysis, we delve into the challenges inherent in this research, a topic elaborated on in the subsequent section.



Strategy	Interaction Level	Support Metadata?	Support Real values?	Support Non-uniform values?	Support Non-uniform interaction?
OA	2-way	No	No	No	No
CA	3-way	No	No	No	No
TConfig	3-way	No	Yes	No	No
CTS	3-way	No	No	No	No
AllPairs	2-way	No	Yes	Yes	No
AETG	3-way	No	No	Yes	No
mAETG	3-way	No	No	Yes	No
TCG	2-way	No	No	Yes	No
mTCG	2-way	No	No	Yes	No
GA	3-way	No	No	Yes	No
ACA	3-way	No	No	Yes	No
IPO	2-way	No	No	Yes	No
IPOG	5-6 way	No	No	Yes	No
Jenny	7-12 way	No	No	Yes	No
TVG	5-6 way	No	Yes	Yes	No
ITCH	4 way	No	Yes	Yes	No
GTway	7-12 way	No	Yes	Yes	No
MTTG	Higher t way	No	Yes	Yes	No
PSTG	5-6 way	No	No	Yes	No
PS2way	2 way	No	Yes	Yes	No
EasyA	2 way	No	Yes	No	No
SITG	2-6 way	No	Yes	Yes	Yes

Fig. 7: Comparison of different strategies

IV. Research Discussions and Suggestions

The preceding section highlights that the majority of strategies endorse non-uniform values. However, few strategies extend support for non-uniform interaction, and none encompass meta-data such as voice or images as input. Consequently, there arises a need for an innovative strategy that not only supports non-uniform interaction but also accommodates meta-data—an essential motivation behind our current research. Building upon this motivation and the insights gleaned from prior literature reviews, we delve into the exploration of the following research questions [9, 16-18,26-29]:

- What is the optimal and smaller set of test data to choose over the large dataset? i.e. which strategy to choose among all strategies, that can produce optimal test data set.
- Which test data generation strategy to choose in terms of complexity? i.e. which strategy is the best in terms of time and space.
- What is the optimal data to choose in different interactions to maximize the testing coverage? i.e. which interaction level should we select in case of non-uniform interactions exist.

Much effort has been expended to optimize the Combinatorial Explosion Problem (CEP) principal through traditional computing analysis over the past decade [16-18, 20]. However, through parallelization, CEP may be alleviated, but the development of complex software and hardware still poses the same question to the researchers.

V. Conclusion

development, hence is very profitable. Testing is no longer an additional activity but integral part of software development life cycle (SDLD). Emphasizing on the development of techniques to reduce test cases actually helps in the orderly execution of test cases based on the functions or performances of the target (amount of coverage, execution time, and cost). In this paper, a survey was made on the existing test case generation techniques with their internal details. This study reveals the techniques used in selecting the optimal test cases, and a group of previous works was addressed and compared among them. After reading all the selected research in full, the following was concluded: First, from the review of the works, all the similarity was identified and was summarized. Second, from all the existing testing techniques the dissimilarity was identified. Thirdly all the existing testing was analysed for the drawbacks of each techniques was identified and Finally all the findings were summarized in a tabular form to see the research gap at a glance. Moreover the most widely used in determining the precedence of test cases, as well as the execution time and the amount of error coverage are largely identified to use as a measure for evaluating test cases. Observing from the summary table it becomes clear that the use of artificial intelligence and support of meta-data is the future research area in developing test case optimization technique development. Although in NP-hard, it is impossible to develop a mathematical model to generate a polynomial result, our future works also involves in finding a test data generation model.

References

[1] Ammann P. and Offutt J., Introduction to Software Testing, Cambridge University Press, 2008.

[2] Garousi, Vahid, and Junji Zhi. ”A survey of software testing practices in Canada.” Journal of Systems and Software 86, no. 5 (2013): 1354-1376.

[3] Lima, Rui, António Miguel Rosado da Cruz, and Jorge Ribeiro. ”Artificial intelligence applied to software testing: A literature review.” In 2020 15th Iberian Conference on Information Systems and Technologies (CISTI), pp. 1-6. IEEE, 2020.

[4] Y. Cui, L. Li, and S. Yao (2009). A New strategy for pairwise test case generation. 3rd international Symposium on Intelligent Information Technology Application.

[5] Khandakar Rabbi, Rafiqul Islam, QuaziMamun and Mohammed Golam Kaosar (2014). MTTG: An Efficient Technique for Test Data Generation. 8th International Conference on Software, Knowledge, Information Management and Applications.

[6] Yan, J. & Zhang, J. (2008) A Backtracking Search Tool for Constructing Combinatorial Test Suites. Journal of Systems and Software – Elsevier.

[7] Hartman, Alan, and Leonid Raskin. ”Problems and algorithms for covering arrays.” Discrete Mathematics 284, no. 1-3 (2004): 149-156.

[8] Yu, Linbin, Yu Lei, Raghu N. Kacker, D. Richard Kuhn, and James Lawrence. ”Efficient algorithms for t-way test sequence generation.” In 2012 IEEE 17th International Conference on Engineering of Complex Computer Systems, pp. 220-229. IEEE, 2012.

[9] Yan, J. & Zhang, J. (2008) A Backtracking Search Tool for Constructing Combinatorial Test Suites. Journal of Systems and Software – Elsevier.

[10] Chateauneuf, M. &Kreher, D. (2002) On the State of Strength-Three Covering Arrays. Journal of Combinatorial Designs.

[11] Colbourn, C. J., Martirosyan, S. S., Mullen, G. L., Shasha, D., Sherwood, G. B. &Yucas, J. L. (2005) Products of Mixed Covering Arrays of Strength Two. Journal of Combinatorial Designs.

[12] Williams, A. W. (2000) Determination of Test Configurations for Pair-wise Interaction Coverage. Proc. of the 13th International Conference on Testing of Communicating Systems.

[13] Hartman, A. &Raskin, L. (2004) Combinatorial Test Services [Online]. [Accessed on March 2015]. Available from: https://www.research.ibm.com/haifa/projects/ verification/mdt/papers/CTSUserDocumentation.pdf.

[14] Bach, J. (2004) ALLPAIRS Test Generation Tool, Version 1.2.1. [Online]. [Accessed on March 2015]. Available from: http://www.satisfice.com/tools.shtml.

[15] Ellims, M., Ince, D. &Petre, M. (2008) AETG vs. Man: an Assessment of the Effectiveness of Combinatorial Test Data Generation. UK, in Technical Report, Department of Computing, Faculty of Mathematics and Computing, Open University.

[16] Cohen, M. B., Dwyer, M. B. & Shi, J. (2007) Exploiting Constraint Solving History to Construct Interaction Test Suites. Proc. of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. UK, IEEE Computer Society.

[17] Yu-Wen, T. &Aldiwan, W. S. (2000) Automating Test Case Generation for the New Generation Mission Software System. Proc. of the IEEE Aerospace Conference.

[18] Cohen, M. B. (2004) Designing Test Suites for Software Interaction Testing. Computer Science. New Zealand, University of Auckland.

[19] T. Shiba, T. Tsuchiya, and T. Kikuno (2004). Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing. 28th Annual International Computer Software and Applications Conference.

[20] Forbes, M., Lawrence, J., Lei, Y., Kacker, R. N. & Kuhn, D. R. (2008) Refining the In-Parameter-Order Strategy for Constructing Covering Arrays. NIST Journal of Research.

[21] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, J. Lawrence (2007). IPOG: A general strategy for t-way software testing. 14th Annual IEEE International Conference and Workshops on the Engineering and Computer-Based Systems.

[22] Jenny, Available at: http://www.burtleburtle.net/bob/math/. (Last access date: 27th Sep. 2010).

[23] TVG, Available at: http://sourceforge.net/projects/tvg. (Last access date: 27th Sep. 2010).

[24] Hartman, A. &Raskin, L. (2004) Problems and Algorithms for Covering Arrays. Discrete Mathematics-Elsevier.

[25] Kamal Z. Zamli, Mohammad F.J. Klaib, Mohammed I. Younis, Nor Ashidi Mat Isa, Rusli Abdullah (2009). Design and implementation of a t-way test data generation strategy with automated execution tool support. Information Sciences, Elsevier.

[26] Ahmed, B. S, Zamli, K. Z. (2010). PSTG: A T-Way Strategy Adopting Particle Swarm Optimization. Mathematical/Analytical Modelling and Computer Simulation (AMS).

[27] Khandakar Rabbi, Rafiqul Islam, Quazi Mamun and Mohammed Golam Kaosar (2014). MTTG: An Efficient Technique for Test Data Generation. 8th International Conference on Software, Knowledge, Information Man-

agement and Applications.

[28] SabiraKhatun, KhandakarFazley Rabbi, CheYahayaYaakub, Mohammad FJ Klaib, Mohammad Masroor Ahmed (2011). PS2Way: An Efficient Pairwise Search Approach for Test Data Generation. Software Engineering and Computer Systems, Springer Berlin Heidelberg.

[29] KhandakarFazley Rabbi, SabiraKhatun, CheYahayaYaakub, Mohammad FJ Klaib (2011). EasyA: Easy and Effective Way to Generate Pairwise Test Data. 2011 Third International Conference on Computational Intelligence, Communication Systems and Networks.

[30] Hourani, Hussam, Ahmad Hammad, and Mohammad Lafi. ”The impact of artificial intelligence on software testing.” In 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT), pp. 565-570. IEEE, 2019.

[31] Grindal, Mats, Jeff Offutt, and Sten F. Andler. ”Combination testing strategies: a survey.” Software Testing, Verification and Reliability 15, no. 3 (2005): 167-199.

[32] Cadar, Cristian, Vijay Ganesh, Peter M. Pawlowski, David L. Dill, and Dawson R. Engler. ”EXE: Automatically generating inputs of death.” ACM Transactions on Information and System Security (TISSEC) 12, no. 2 (2008): 1-38.

[33] Arcuri, Andrea, and Lionel Briand. ”A practical guide for using statistical tests to assess randomized algorithms in software engineering.” In Proceedings of the 33rd international conference on software engineering, pp. 1-10. 2011.

[34] Khatun, Sabira, Khandakar Fazley Rabbi, Che Yahaya Yaakub, Mohammad FJ Klaib, and Mohammad Masroor Ahmed. ”PS2Way: an efficient pairwise search approach for test data generation.” In Software Engineering and Computer Systems: Second International Conference, ICSECS 2011, Kuantan, Pahang, Malaysia, June 27-29, 2011, Proceedings, Part III 2, pp. 99-108. Springer Berlin Heidelberg, 2011.

[35] Rabbi, Khandakar Fazley, Abul Hashem Beg, and Tutut Herawan. ”MT2Way: A novel strategy for pair-wise test data generation.” In Computational Intelligence and Intelligent Systems: 6th International Symposium, ISICA 2012, Wuhan, China, October 27-28, 2012. Proceedings, pp. 180-191. Springer Berlin Heidelberg, 2012.

[36] Dorigo, Marco, Vittorio Maniezzo, and Alberto Colorni. ”The ant system: An autocatalytic optimizing process.” (1991).

[37] Rabbi, Khandakar. ”Combinatorial testing strategies based on swarm intelligence.” (2017). https://researchoutput.csu.edu.au/en/publications/combinatorial-testing-strategies-based-on-swarm-intelligence