An orchestrated survey of available algorithms and tools for Combinatorial Testing

Sunint Kaur Khalsa and Yvan Labiche
Department of Systems and Computer Engineering
Carleton University
Ottawa, Canada
sunintkhalsa@cmail.carleton.ca, labiche@sce.carleton.ca

***Abstract*— For functional testing based on the input domain of a functionality, parameters and their values are identified and a test suite is generated using a criterion exercising combinations of those parameters and values. Since software systems are large, resulting in large numbers of parameters and values, a technique based on combinatorics called Combinatorial Testing (CT) is used to automate the process of creating those combinations. CT is typically performed with the help of combinatorial objects called Covering Arrays. The goal of the present work is to determine available algorithms/tools for generating a combinatorial test suite. We tried to be as complete as possible by using a precise protocol for selecting papers describing those algorithms/tools. The 75 algorithms/tools we identified are then categorized on the basis of different comparison criteria, including: the test suite generation technique, the support for selection (combination) criteria, mixed covering array, the strength of coverage, and the support for constraints between parameters. Results can be of interest to researchers or software companies who are looking for a CT algorithm/tool suitable for their needs.**

Keywords— Combinatorial testing, Covering arrays, strength of testing, algorithms, Category-partition

I. INTRODUCTION

Software testing is the process of ensuring a software under test (SUT) performs as intended. Software testing techniques can be broadly categorized as black box, when test case construction focuses on functionality, or white box, when test case construction uses the internal logic and structure of the code. In this paper we focus on black-box testing, and more specifically on testing from a plain language specifications, which requires the identification characteristics of input and output parameters[1] . Various input parameter modeling and selection techniques have been suggested in literature (e.g.,[2-6]), and we primarily consider the Category Partition method [6].

Category partitioning begins by identifying the parameters and environment variables of functional subsystems. Parameters are the inputs to the functional subsystems either by the user or some other functional unit. Environment variables are factors that may impact behaviour. These parameters and environment variables are then envisioned into categories, which are characteristics of the parameter (or environment variable) that are deemed important from a testing point of view. Each characteristic leads to the definition of so-called choices, which are equivalent classes, possibly using boundary value analysis, splitting the domain of values (implicitly) defined by the characteristic. Constraints can then be used to specify for instance that some choices from two different categories should always be used together, can never be used together, or can be used together under certain condition. The choices (at most one choice per category) are then combined to form test frames according to a selection criterion, while satisfying constraints. The test frames are then provided actual values to produce test cases: one test frame typically becomes one test case, although it is possible (though costly) to identify several sets of test inputs and therefore several test cases for a single test frame.

Four main selection criteria to combine choices have been defined [1]. With the Each Choice criterion, each choice in each category must appear in at least one test frame. With the Pair-wise criterion, an adequate test suite exercises each possible (according to constraints) pair of choices from different categories at least once. With the Base Choice criterion, a base choice is selected for each category. This is the most "important" choice for the category that is to be tested more often than other choices. A first test frame is created by using all the base choices, i.e., the base choice for each category. Other test frames are created by holding all but one base choice constant and using each non-base choice once for the one non-constant choice, while satisfying constraints. The All combinations (or N-wise) criterion ensures that all the possible (according to constraints) combinations of choices are exercised by the set of test frames.

In order to use these selection criteria when one has a large number of parameters and choices, a technique is required which can effectively and efficiently make combinations. A widely used technique for this task is Combinational Testing (CT), rooted in the mathematical concept of combinatorics and which leads to the construction of .combinatorial objects called Orthogonal Arrays (OA) or Covering Arrays (CA). CT can be broadly applied at two levels [7]: At the configuration level, system configurations are considered as parameters for testing e.g. operating systems, browsers, network protocols; At the input parameter level, the actual inputs to the system or subsystem are considered either in terms of actual values or in terms of partition of the input space as defined by equivalence partitioning.

The reader will notice we need two different terminologies. With Category partition, parameters are characterized by categories, which are split into choices, and

choices need to be combined (one choice per category) to form test frames / test cases. In the CT domain, parameters have values and one combines those values (one value per parameter) to form test cases. We can establish a mapping between the two terminologies: categories and choices (Category partition) map to parameters and values (CT). Unless otherwise specified, we will use the category/choice terms when the discussion is on category partition, and we will use the parameter/value terms when the discussion is on CT. We may need to mix terms, though without loss of clarity when one remembers the mapping.

A CA or an OA is a matrix in which columns represent the parameters and rows corresponds to the test frames. There are few differences in the features of OA and CA but for application in category partition, a CA is typically preferred for a number of reasons, including: an OA assumes that all the categories have the same number of choices, which is rarely the case in practice [8].

Specifically, in the context of category partition, a test engineer would be looking for a CA generation solution that could support one or more of the following: different categories typically have different number of choices; choices are typically associated with constraints to enforce or prevent some combinations, or to ensure that a choice only appears once in the set of test frames; different selection criteria (see previous discussion) can be considered to generate combinations of choices. Contrary to other studies that compare the effectiveness of CA generation technologies at producing the least number of test cases, we are interested in functionalities of such technologies. We believe this comparison will help researchers and practitioners to analyse the tools and algorithms befitting to their needs.

As further discussed below in section II, our search for such information was not successful. We therefore decided to systematically identify and review existing CA generation technologies and compare them according to the above-mentioned objectives (among other things).

The rest of the paper is structured as follows. Section II discusses related work. Section III discusses the protocol we followed to identify algorithms/tools generating covering arrays, while section IV discusses the comparison criteria we are interested in. Sections V and VI present results. Section VII discusses threats to the validity of our study. We conclude in section VIII.

II. Related Work

Various types of CAs have been defined [9]. A (standard) Covering Array is typically defined as an array of N rows and p columns, N being the number of test cases and p the number of parameters, each one having v possible values, such that for every selection of t columns (t being called the strength of the array) all possible t-tuples of v values appear within the rows of the array [10]. A Mixed Covering Array (MCA) allows parameters to have various numbers of values [10]. A Variable Strength Covering Array (VSCA) ensures several strength values are achieved for different set of parameters [10]. A Constraint Covering Array (CCA) accounts for forbidden combinations of values, a.k.a., forbidden tuples [11]. A sequence covering array [12] accounts for sequence in which parameter values must be provided to the system under test, which is especially relevant when testing GUI-based software. For obvious reasons, these are the covering arrays we are mostly interested in, for use with category partition.

Other covering arrays include Error Locating Arrays [13], Test Case Aware Covering arrays [14], Cost aware covering arrays [9, 15], Incremental Covering arrays [16] and l-Biased Covering array [17].

We found papers [18, 19] which surveyed methods for generating covering arrays. These papers surveyed covering arrays generation techniques on the basis of size and time of generation of covering arrays. They however did not discuss extensively the tools or algorithms which supported a specific technique, the coverage strength or selection criteria. Other survey (e.g., [8, 20-23]) focussed on the techniques but did not discuss all the tools and algorithms supporting those techniques in detail. For instance they do not discuss support for constraints or higher coverage strength which is essential in our category partition context. The nearest work to our survey is by Rahman et al. [22], who discussed various techniques, their strengths and weaknesses along with the coverage strengths they support. They also mentioned if a specific technique supports constraints. They did not extensively mention the tools or algorithms supporting a specific technique, the constraint handling and representation technique adopted by a tool/algorithm, the selection criteria supported by a tool. In other words we intend to provide a more complete picture than what can be found in the literature to date. There is only one research work [11], to the best of our knowledge, which discusses the constraint handling support in tools/algorithms for nine tools whereas we discuss 32 such tools.

To summarize, none of the research work, to the best of our knowledge, surveys the tools and algorithms as extensively as what we report in this paper or compare them on the basis of comparison criteria which we outlined earlier.

III. Selection Protocol

We did not strictly follow established guidelines [24] to conduct a systematic mapping study [25], mostly deviating from those guidelines in the way we identified relevant publications since we did not rely on online databases such as IEEE eXplore. We nevertheless followed the SMS principles by considering research questions (section III.A), establishing a precise procedure to identify relevant publications (section III.B), clearly stating publication inclusion and exclusion criteria (sections III.C and III.D), and by defining a publication comparison framework (section IV). This way, we intend to be as systematic and reproducible as possible and we therefore dedicate a fair amount of space in this paper to those details.

A. *Research questions*

Recall that we are interested in using a CA generation technology to produce test frames in the context of category partition testing technique. We therefore identified the following research questions:

RQ1: What are the available tools/algorithms for generating combinatorial tests?

RQ2: Which techniques, e.g., based on mathematical construct or based on a meta-heuristic search, are used for generating covering arrays for combinatorial testing?

RQ3: Which selection criteria (to generate test frames, i.e., combinations of choices) does each tool/algorithm support?

RQ4: What is the maximum coverage strength supported by each tool/algorithm?

RQ5: Which tools support constraints and how do they represent and handle them?

RQ6: Which tools support mixed covering arrays?

B. Selection procedure

The selection procedure we followed started from survey papers [8, 18-23], which gave a fairly good idea regarding the techniques used for generating combinational tests (Covering Arrays). But since our objective was to search for available tools/algorithms which support each specific technique, we first looked at the tools/algorithms mentioned in those surveys. We searched and studied literature on these tools/algorithms one by one. We extensively reviewed the related work and result sections of these papers, searching for new tools/algorithms being compared to the first list of tools/algorithms. We repeated this process multiple times, recursively, until no new tool/algorithm was identified.

Further, to ensure that our list was as complete as possible, we also searched for tools/algorithms in the papers where the survey papers were cited. We further reviewed the thesis of various researchers [17, 26-28], technical reports [29], books [7], websites (e.g., www.pairwise.org) and feature documents of various tools (e.g., ACTS, PICT).

C. Excluded papers

During the selection procedure we identified many studies which proposed an improvement over another existing algorithms, such as lowering the bound of CAs, but these papers did not have an implementation or much experimental results of comparison with other algorithms, and were not changing the essence of the algorithm to such an extent that our classification of the new algorithm would differ from that of the original. Hence these studies were excluded. Tools/algorithms based on Orthogonal Arrays (e.g., OATS, rdExpert, reducearray2, reducearray3) were excluded because of their limitations mentioned in the Introduction. Papers on other input parameter modeling technique e.g. classification trees (CTE_XL), Combinatorial testing for Software Product lines, Grammar based combinatorial testing, testing of compilers were also excluded. Algorithms/tools supporting prioritization of the values or parameters were also excluded. We have focused on literature only in English.

D. Included papers

We have included tools and algorithms which generate combinatorial test suite. We included tools/algorithms which support input/output relationships, distance based techniques for the selection of parameters and values. We have made an exception here regarding the selection of an algorithm named Distance Based Technique [30]. This work does not perform comparison with other tools but we have included it in our survey because it supports three selection criteria, coverage strength of 5 and is the only distance based technique which supports constraints. The basis of this inclusion is the variety in results.

The AETG's Web service [31] is based on the algorithm proposed by Cohen et al. [32] which was further improved by Cohen in [26]. For our review we will be considering the commercial tool AETG Web Service which is available online. ACTS [33] implements several combinatorial test generations algorithms like IPOG and IPOD [34], IPOF [35], IPOF2 [35], and IPOG-C [36] which uses constraints, all being rooted in the In parameter Order (IPO) algorithm [37]. We decided to consider ACTS itself rather than all these improvements separately.

IV. Comparison framework

The comparison framework consists of various comparison criteria, derived from our research questions, we will use for comparing the tools and algorithms we selected (section III).

A. Techniques for the generation of covering array

Various techniques for generating covering arrays for CT have been proposed in literature. The construction of CAs are usually performed in two steps [7]. In the first step a set containing all the possible t-wise combinations is generated. In the second step the test suite is generated to cover all the combinations obtained in the first step. Both steps collectively are called the technique for test suite generation. Researchers have suggested various paradigms for the characterization of the test suite generation techniques. Grindal et al. [8] characterize techniques on the basis of the determinism of the generated output. They broadly categorized techniques as *deterministic* and *non-deterministic* and further into *heuristic*, *artificial life based*, *iterative* (test suite generated in iterative steps) and *instant* (test suite generated in one step) depending on the type of algorithm being used and how the test suite is generated. In this classification the categories however are not disjoint. For instance they have classified covering arrays as deterministic although an algorithm generating a CA does not necessarily produce deterministic results when they are generated using Simulated Annealing [38].

Nie and Leung [20] performed an extensive survey and provided another classification scheme which classified the covering array generation techniques into greedy algorithms, heuristic search, mathematical methods and random method. It is interesting to note that, as reported by Nie and Leung, a technique may fall into more than one category: e.g., the hybrid techniques of Bryce and Charles [39] combines a heuristic search and a greedy algorithm to benefit from both techniques. We extended this taxonomy in this paper.

B. Test generation strategy

The algorithms for combinatorial test suite generation can be broadly categorized into Test based generation and Parameter based generation. An algorithm uses either a test case or a parameter as the building block for the generation of

the test suite. In test-based generation, one test is build at a time such that the test covers as many t-way combinations as possible and hence spans over all the parameters. Automatic Test Case Generator (AETG) [32] falls in this strategy. Parameter based generation, begins with t parameters, makes a test suite for t-wise interaction and then adds more parameters to it. While adding new parameters new test (rows) are also added, often greedily, so that each addition leads to maximum t-way interactions in the extended set of parameters. This is the strategy of In Parameter Order [40]. The Algebraic Techniques which follow a recursive approach also use a parameter based strategy. The building block in a recursive algebraic technique is a smaller covering array, which is a group of parameters, and the larger arrays are obtained from smaller arrays [41].

C. Selection Criteria

In a typical situation, exercising all the possible combinations of parameter values, i.e., t-way coverage for a problem with t parameters, is simply not practical or feasible because of the large set of parameters and values. Hence, it is important to select the parameters and values strategically so that their combinations can lead to a manageable set of test cases. In software testing, this is typically achieved thanks to test selection criteria, four of which have already been mentioned in the context of category partition: Each Choice, Pair-Wise, Base Choice and All Combinations.

Some authors have suggested other criteria [1, 8] in the CT domain:

- Uniform Strength Interaction or t-wise: This criterion requires that any combination of values belonging to t parameters should be combined at least once in the test suite. Here all the parameters are supposed to be uniformly integrated with a constant value t [42]. The Pair wise criterion previously mentioned corresponds to a uniform strength of 2.
- Variable strength Interaction or mixed strength interaction: This is an extension to the t-wise criterion that requires t-wise interaction among a subset of parameters and q-wise interaction among the remaining parameters [43].
- Input output based interaction: Instead of exercising interactions of the complete set of parameters, this criterion is to split the set of parameters into (possibly overlapping) subsets that each contain the parameters that impact the value of one output parameter [44]; a selection criterion like the ones previously discussed (the authors use the all combinations criterion but another criterion could be used) can then be used on each subset of parameters and results need to be combined to obtain complete test cases.
- Distance based criterion: the goal of the criterion is to select combinations of parameter values, i.e., test cases, that are as diverse as possible, diversity being measured as the distance between those test cases, for instance using the Hamming distance [3].
- Random input criterion: This criterion selects a randomly chosen number of test cases and each test case is a randomly selection of parameter values.

Although reporting on experimental work involving those criteria is not the purpose of this paper, we nevertheless would like to mention a few results. Grindal et al [45] observed that the *Each Choice* criterion supplied unpredictable results so much so that they were not very useful to the testers. Because of their nature, *base choice* and *input output based interaction* were able to detect different types of faults as compared to other criteria. *Base Choice* was observed to give better fault detection results when there were a limited number of choices per category that could be considered base choices; when this was not the case *Pair-Wise* gave better results. Othman et al. [46] showed that Input output based parameter interaction gave better results in terms of cost (i.e., number of test cases) and ability to find faults than uniform and variable strength interaction. Others have found that the presence of constraints between choices heavily impacts the use of those criteria, including Base choice which, in some cases, does not exercise every single choice [47]. All these authors unanimously argue that it is difficult to generalize those results as they largely depend on the system under test.

D. Coverage Strength Support

Strength support is an important characteristic for the comparison of tools or algorithms producing CAs. The intuition, confirmed experimentally, is that the faults are detected when parameters interact. Studies [7] have shown that 100% fault detection can be achieved by a maximum of 4-wise to 6-wise interaction. Other studies found that pair-wise performs better than higher strength values are is therefore more cost-effective. Identifying the strength of a tool/algorithm is therefore important.

E. Constraint Support

Constraints are limiting the construction of a CA by forbidding some combinations. One can distinguish between environment constraints and system constraints [7]. Environment constraints are related to the configurations of the system, e.g., Linux OS can never be combined with Internet Explorer. System constraints on the other hand are constraints on the system, e.g., one user cannot select a value less than 10. System constraints can also be used to test the robustness of the system, e.g., behaviour of the system if the user selects invalid options.

Constraints can be represented either as forbidden tuples, allowed tuples or formally specified with the help of prepositional formulas or logical expressions using Boolean, relational or arithmetic operators [36] [48]. A forbidden tuple is a combination of parameter-values which can not appear in the final test suite. A single constraint can give rise to any number of forbidden tuples [11]. In a typical situation, constraints are formally specified, not necessarily explicitly, by the test engineer. Tools/algorithms which accept formally specified constraints are therefore more usable than those that do not since in the latter case, the test engineer needs to remodel the constraint input to transform the formal specification into a list of forbidden or allowed tuples [11].

We recognized four mechanisms for handling constraints by the tools/algorithms. The first mechanism is handling constraint before executing a specific test generation

algorithm. This mechanism can be adopted when only allowed tuples are given as input and can prevent changing the test generation algorithm. The second mechanism is to replace the invalid test cases with valid ones once the test suite has been generated using a specific technique [49]. The third mechanism is to integrate constraint handling into the CA generation algorithm with an ad-hoc procedure. The last mechanism is to integrate the selection of valid tuples, according to constraints, to the algorithm generating combinatorial the test suite by integrating a SAT solver.

F. Support for Mixed Covering Arrays

A typical software system will have a large number of parameters and each parameter will not necessarily have the same number of values. So a tool/algorithm should be able to support mixed covering arrays to cater the need of such a software system.

V. RESULTS

In this section we will answer the research questions individually (sections V.A to V.F). Some of those results are combined in section VI. The complete list of 75 algorithms/tools we selected by following our selection protocol, along with their raw classification using our comparison framework can be found in Appendix Section IX.

A. RQ1: What are the available tools/algorithms for generating combinatorial tests?

The objective of this research question is to know the available tools in the realm of combinatorial testing using our search protocol and categorizing them and show their year of introduction in Fig. 1 over four years intervals from 1991 (the earliest year we found) to 2014. The total number of tools/algorithms obtained using our search protocol is 75 (TABLE III. and TABLE IV.). Among the first tools for generating tests were T-Gen [50], introduced in 1991, and based on the category partition method [6]. In the next four years no tool was proposed and then from 1999 onwards there has been a constant rise in the number of tools/algorithms for generating covering arrays for combinatorial suiting. This clearly marks the importance of functional testing and a need to have an optimal test suite. 69% of the tools/algorithms have been proposed in the last eight years.

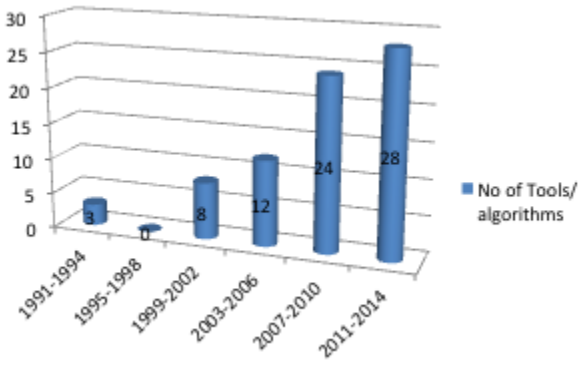

Fig. 1. Number of tools/algorithms identified over years, presented over four year intervals

While searching for specific tools/algorithms we observed that authors suggested improvements to their own technology over the years while proving experimentally that their new algorithm was doing better than before. In our study an algorithm with multiple references corresponds to the improvements the algorithm went through, along the way, and we considered the results of the latest upgrade.

B. RQ2: Which techniques are used for generating covering arrays for combinatorial testing?

1) Greedy vs. Meta-heuristic vs …

We identified five different types of techniques used for the generation of combinatorial test suites: Greedy Techniques, Meta-Heuristic Techniques, Adaptive random / Adhoc techniques, Hybrid Techniques and Algebraic techniques. We are going to classify all the algorithms and tools, obtained from our search, into these five categories.

A Greedy Technique generates tests by uncovering a locally optimal solution and ensures that each new test uses the maximum possible uncovered combinations. They are usually faster than the Meta Heuristic techniques but do not always produce the smallest test suites. These algorithms return a local optimum rather than a global optimum. Tools/algorithms based on backtracking algorithms, branch and bound techniques, exhaustive search, AETG type algorithm [32], IPO based algorithms [40] etc all fall in this category. A Greedy algorithm is typically used when a problem does not have a known best polynomial time algorithm [18].

The generation of a combinatorial test suite is an optimization problem and Meta-heuristic techniques are also used to solve it. These techniques can be evolutionary algorithms e.g. Genetic Algorithms or naturally inspired algorithms e.g. Particle Swarm Optimization or any other standard known optimization algorithms. Such an algorithm searches the neighbourhood of a solution and finds the best fit. The algorithm starts from a pre existing test called a seed and, after performing a series of transformations, achieves a test suite that has a minimum number of tests and maximum uncovered tuples. A heuristic search such as Simulated Annealing produces smaller sets than a greedy algorithm but takes more time to execute [51]. The various Meta-heuristic techniques we identified and are used for generating covering arrays are Hill Climbing, Simulated Annealing, Tabu Search, Genetic Algorithm, Ant Colony Optimization, Partial Swarm Algorithm, Harmony Search, Extremal optimization and Great Flood.

The category of Adaptive random or ad-hoc techniques contains two types of techniques. Adaptive random uses an algorithm that relies on a measure of distance between the parameter values, e.g., using the Hamming distance, to generate the test suite that are maximally apart from one another (e.g., [52] [30]). The set of ad-hoc techniques contains those which are not using any of the other techniques. An ad-hoc approach typically selects the test cases randomly or on the basis of some input distribution (e.g., [53] [54]).

Hybrid approaches were also proposed by researchers to achieve better and optimal results. The objective behind

combining techniques is to reduce the size and generation time of the CA and increase the coverage and hence the fault detection. For instance, Bryce et al. [39] combine a greedy algorithm with a heuristic search, Cohen et al. [55] combined a mathematical approach with Simulated Annealing.

Algebraic Techniques create covering arrays by either directly computing a mathematical function or by using defined rules. Some algebraic constructions also use recursion to obtaining larger covering arrays from smaller building blocks [41]. The applicability of algebraic approach is limited because they impose restrictions on the system configurations which they can accept. This approach is usually an extension to the algorithms of orthogonal array [7].

Fig. 2 shows that, out of these 75 algorithms we found, 40 (53%) used a greedy approach for the generation of the combinatorial test suite, 13 (17%) tools/algorithms used Meta heuristic techniques, 5 (6%) tools/algorithms belonged to the category of Adaptive random and adhoc. We found 6 (8%) tools/algorithms which used a hybrid approach, either combining a greedy technique and a metaheuristic technique, or a greedy technique and an algebraic technique. There were 4 (5%) tools/algorithms which supported algebraic techniques. We attribute this small number to the fact that algebraic techniques are not as versatile as other techniques (e.g., to support many different strength values, to support constraints). Most of the research in algebraic methods is focusing on generating smaller covering arrays, which can then be used as a seed for other techniques. In our search we also found 7 (9%) tools which were not accompanied by a detailed technical documentation which could help us classify them according to this criterion.

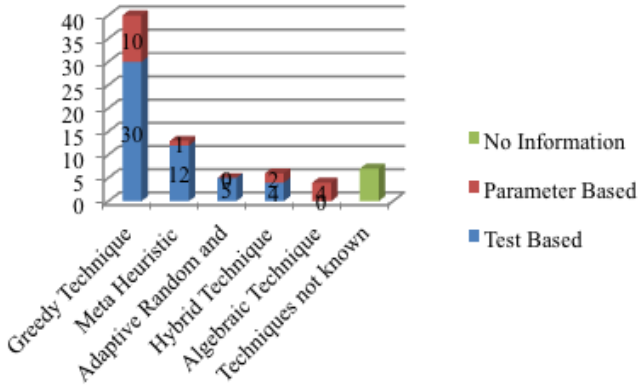

Fig. 2. Categorization of the tools/algorithms on the basis of techniques and generation strategy.

The advantage of greedy and meta-heuristic technique is they can be applied to any size of system configurations, i.e. there is no restriction on the number of parameters or the number of values each parameter can take. The downside is that they take more time to create a CA [7]. On the other hand, Algebraic Techniques are extremely fast and lightweight but only on a subset of system configurations. They cannot, as well, deal efficiently with constraints [20].

2) Test based vs parameter based

Further tools/algorithms can be classified according to their generation strategy (test based vs. parameter based). We observed that out of the 68 tools/algorithms, whose technical details are known to us, 75% of the tools (51 out of 68) followed a test based generation strategy and 25% (17 out of 68) followed a parameter based generation strategy. Out of the 40 tools/algorithms which generated test suites using a greedy approach 30 followed a test based generation and 10 followed a parameter based generation. Fig. 2 summarizes those results.

3) Meta-heuristic techniques

Fig. 3 shows the number of tools using a specific metaheuristic technique for the generation of covering arrays. The tool/algorithms which use meta-heuristic techniques either belong to the category of meta-heuristic or to the category of hybrid (Fig. 2). Three meta-heuristic techniques namely Particle Swarm, Genetic algorithm and Simulated Annealing are more widely used than others.

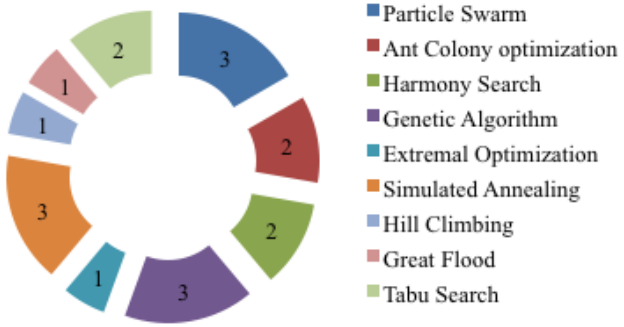

Fig. 3. Number of tools/algorithms using different Meta-Heuristic algorithms for the generation of Covering Arrays

C. *RQ3: Which selection criteria does each tool/algorithm support?*

We identified seven different selection criteria supported by the 75 tools/algorithms: base choice, each choice, input/output, distance, uniform strength, variable strength and random.

The base choice criterion requires the identification of a based choice for each category, that is a choice that is considered the most important of the choices of a category. Since identifying the base choice of a category can be implicitly done by assigning weights to the category's choices and selecting the best (max or min, depending) weighted choice as base choice, we classified all the tools/algorithms which support the assignment of weights to parameter values in the base choice criterion category.

We also made a difference between the each choice criterion and the uniform strength criterion. Uniform strength typically means a strength t of at least two (t>=2) where as each choice corresponds to uniform strength of strength one (t=1). A tool supporting uniform strength of at least two would support uniform strength of one. In our analysis we put the tools/algorithms which explicitly mention their support for the each choice criterion (uniform strength one) in a separate each choice criterion category. During our research we also found a few tools that support variable strength CAs without specifically mentioning support for uniform strength. However since the former implies the latter, we classified those tools as variable strength and include the variable

strength category into the uniform strength category graphically (Fig. 4).

Obtained results are shown in TABLE V. and are summarized graphically in Fig. 4: 72 (96%) tools support uniform strength, 24 (32%) tools support variable strength (and therefore uniform strength). Nine tools support three criteria: input output, variable strength and uniform strength. Another three tools ACTS [33], PICT [56] and IBM Focus [48] support uniform strength, variable strength and base choice criteria. Out of these tools IBM focus and PICT support assigning weights to values. Tcases [57] supports four criteria, which is the maximum we found: uniform strength, variable strength, random and each choice. Two tools support distance based: [52] and [30]; the latter also supports random and uniform strength whereas the former support uniform strength, in addition to distance based.

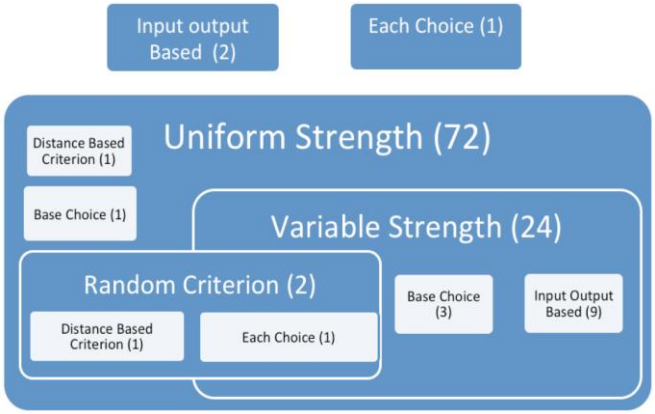

Fig. 4. Support of the tools for the selection criteria

D. *RQ4: What is the maximum coverage strength supported by each tool/algorithm?*

Maximum strength has been studied collectively for uniform strength and variable strength tools/algorithms (TABLE VI. . For a tool/algorithm which only supports uniform strength, the highest strength is obtained for a specific test configuration. In case results were available for more than one test configuration, we chose that configuration, from variable strength or uniform strength, which supports the highest strength value and mixed covering array with maximum number of parameter values.

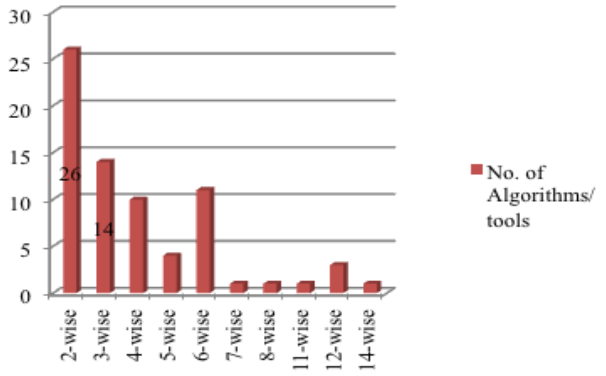

Fig. 5. Maximum coverage strength supported by tool/algorithm

The results of the research question are shown in Fig. 5. This is the result of 72 tools as three tools do not support uniform strength (Fig. 4). Out of 72 tools 26 (36%) support a maximum strength of two, 14 (19.5%) support the strength of three, 11 (15%) support the strength of six, three (4%) support the strength of 12 and we found one tool (Harmony Search Strategy [10]) which supports the coverage strength of 14.

For the selection of research papers to answer this question we followed the following approach. The strength is obtained from two types of sources; the strength experimented in researcher's own work and/or any other research work in which a comparison has been made with that specific tool. We have taken the higher strength of the two and included it in our analysis. There were certain tools/algorithms for which the results were not shown or detailed information was not available but the authors claimed that their algorithm supported a certain strength. We used values reported by authors but flagged the papers in TABLE VI. Further, ATD [58] did not have experimental results but the authors claim to support t-wise coverage, for this tool we assumed the most common value of t=2.

Even though AETG Web Service [31] is based on Cohen et al. technique [32], which supports uniform strength, random inputs and all combinations as the selection criteria, the AETG Web Service only explicitly supports the first criterion of those three. On the basis of our paper selection criteria (section III.D), we only consider the AETG Web Service and only consider its support for uniform strength.

E. *RQ5: Which tools support constraints and how do they represent and handle them?*

Out of the total tools/algorithms in our research work we found 32 (44%) tools/algorithms which support constraints (TABLE VII.). The two important aspects of constraint support we are focusing on are representation and handling mechanism.

Fig. 6 summarizes the results. Tools supporting constraints require an input under the form of forbidden tuples (12, i.e., 37.5%), allowed tuples (2 i.e., 6%) or formal specification (13, i.e., 40%). For five tools, available documentation indicates support for constraints but fails to provide further details so we can classify.

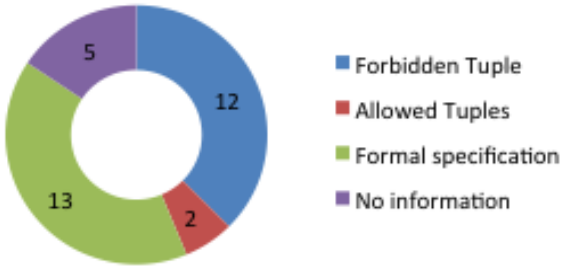

Fig. 6. No of tools/algorithms supporting a specific constraint representation

Fig. 7 summarizes the mechanisms to handle constraints during CA construction. 59% (13+6) of the tools have a mechanism embedded in the CA construction algorithm to handle constraints: 40% use an ad-hoc algorithm, 19% use a SAT solver. None of the tools have been found to use the

mechanism of replacing invalid test cases once the test suite has been generated. Certain tools did not have enough information in the technical documents, in order to help us make decisions.

In our study we found three tools which supported robustness testing, i.e., test for the invalid values, without requiring that specific out-of-bound choices be specified as input: PICT [56], PictMaster[59] and IBM Focus [48].

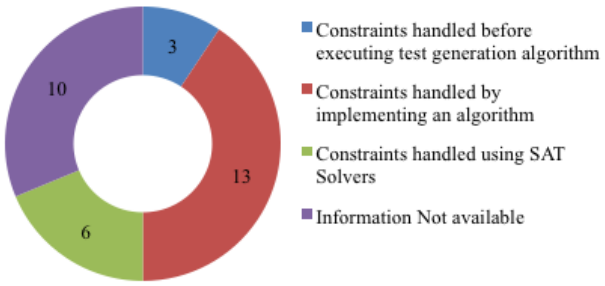

Fig. 7. No of tools/algorithms supporting a specific constraint handling mechanism

F. RQ6: Which tools support mixed covering arrays?

Fig. 8 shows the tools/algorithms which support mixed covering arrays as compared to the tools/algorithms which supported only traditional covering arrays, the detail can be seen in TABLE VIII. We found that 76% of the tools support mixed covering arrays i.e. the parameters given as input can have varying numbers of values whereas 5% of the tools/algorithms assume all parameters have the same number of values. The figure also shows that we were not able to collect that information for 14 tools.

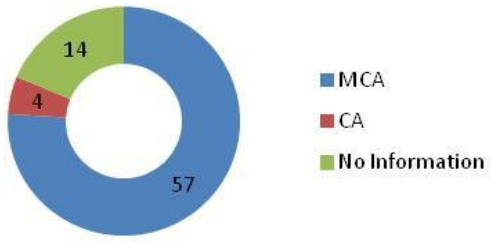

Fig. 8. Support for mixed covering arrays

VI. DISCUSSION

In this section we combine research questions to get a more complete picture of algorithms/tools' capabilities.

A. Combining RQ1, RQ2 and RQ3

The objective of this combination is to analyse the technique and the selection criteria that technique mainly supports: TABLE I. We observe that base choice is only supported by greedy algorithms. Similarly, the input output based criterion is only supported by greedy algorithms (11 algorithms). 66% of the algorithms (16 out of 24) which support variable strength are using a greedy technique. On the other hand only 25% (6 of 24) of the tools which support variable strength used heuristic techniques. A similar trend is observed in case of algorithms which support the uniform strength criterion. Recall that 38 algorithms out of 72 (52%) used greedy technique, 13 algorithms out of 72 (18%) used heuristic techniques, 6 out of 72 (8%) used Hybrid techniques where as 4 (5%) and 5 (7%) algorithms respectively used algebraic and adaptive random techniques. It is important to note here that algebraic techniques contributes to the generation of covering arrays using only the uniform selection criterion and do not support any other criterion. Similarly, hybrid techniques only support uniform strength. The support for distance based and random criteria is only provided by adaptive random and adhoc techniques. None of the tool supports all combinations.

From this analysis it can be concluded that greedy techniques largely support the generation of covering arrays using multiple selection criteria, i.e., base choice, variable strength, uniform strength and I/O based, where as heuristic techniques support only variable strength and uniform strength.

TABLE I. NUMBER OF TOOLS/ALGORITHMS ON THE BASIS OF TECHNIQUES AND SELECTION CRITERIA

	Greedy (Parameter)	Greedy (Test)	Meta-Heuristic (Parameter)	Meta-Heuristic (Test)	Algebraic (Parameter)	Adaptive Random and adhoc (Test)	Hybrid (Parameter)	Hybrid (Test)	Don't Know
Each Choice									2
Base Choice	1	3							
Variable Strength	3	13		6					2
Uniform Strength	10	28	1	12	4	5	2	4	6
I/O Based criteria	1	10							
Distance Based criteria						2			
Random criteria						1			1

A. Combining RQ1, RQ2, RQ4

The objective of this combination is to identify which technique supports which coverage strengths. TABLE II. shows the results of this combination. We observe that Greedy techniques support a range of strengths varying from 2 to 12. The higher strengths in greedy techniques are supported by test based generation: GTWay [60], GVS [61] and ITTDG [62]. The test configuration for these algorithms used 12 parameters, with a maximum number of values of 10 for two parameters. It is important to mention here that two of these algorithms, i.e., GVS and ITTDG, support three selection criteria (variable strength, uniform and input output based criteria), which clearly shows that greedy techniques have outperformed other techniques on the basis of support of selection criteria and higher strength values. The highest strength in our survey was however supported by an algorithm named Harmony Search Strategy (HSS) [10] with a strength of 14. HSS uses a meta-heuristic technique and test based generation for generating covering arrays. The HSS algorithm supports variable strength and uniform strength and the strength of 14 is obtained with a test configuration of 14 parameters each having three different possible values.

We also observe from TABLE II. that Algebraic techniques support a maximum strength of four whereas IPOD [34], which is a hybrid of algebraic technique and greedy parameter based technique, supports a strength of 6. Similarly, the hybrid of meta-heuristic with greedy techniques has also elevated the strength support of meta-heuristic techniques to 4 with an exception of Tabu search. Selecting one technique over another should also consider other factors such as the size of the CA generated or the time it takes to generate it. This is beyond the scope of the present work.

TABLE II. NO. OF TOOLS/ALGORITHMS ON THE BASIS OF TECHNIQUES AND COVERAGE STRENGTH

Selection criteria	Greedy (Parameter)	Greedy(Test)	Meta-Heuristic (Parameter)	Meta-Heuristic (Test)	Algebraic (Parameter)	Adaptive Random and adhoc (Test)	Hybrid (Parameter)	Hybrid(Test)	Don't Know
2-wise	2	10	1	4	2	3			4
3-wise	1	7		4			1		1
4-wise	1	2			2	1		4	
5-wise	3					1			
6-wise	2	5		2			1		1
7-wise				1					
8-wise		1							
11-wise	1								
12-wise		3							
14-wise				1					

B. *Combining RQ1, RQ2, RQ5*

The objective of this combination is to know which technique supports constraints: Fig. 9. 53% of the algorithms/tools (17 of 32) which support constraints use a Greedy Technique. These algorithms/tools either implement the constraint handling algorithm or use a SAT solver for handling the constraints. This is followed by 13% of the algorithms which use Meta heuristic techniques (e.g., simulated annealing). A meagre number of tools based on algebraic, adaptive random and hybrid techniques support constraints. It can be concluded from this observation that a greedy technique is more flexible to the implementation of constraints as compared to other techniques.

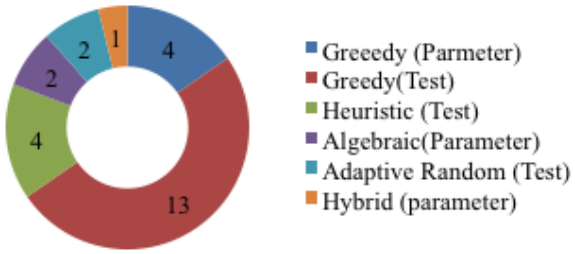

Fig. 9. No. of tools algorithms on the basis of techniques and constraint support

VII. THREATS TO VALIDITY

We believe that the list of tools/algorithms we have identified (see complete list in TABLE III. and TABLE IV.) is the most extensive one to date, and definitely more extensive than the literature we surveyed. We cannot however ignore the possibility of missing a tool or algorithm. One threat which we foresee in our work is that if a specific tool/algorithm is not compared, referred to or mentioned in a surveyed work, thesis or website there are chances that we have missed it. We however believe the risk is small since we captured publications by the main actors in the field.

While assigning a suitable category to a technique used by the algorithm for generating a combinatorial test suite, we encountered situations when the algorithms were not mentioned in detail or not mentioned at all. For the tools we could not find the algorithms we have categorized them as "information not available", and for some research papers which lacked proper explanations we made the nearest possible guess for the type of algorithm. Data may therefore not be entirely accurate. We however show there are a very few number of those occurrence and therefore the threat to our general observations and conclusions is small.

While looking for the maximum strength a tool supports we have considered two types of research work; work in which that tool/algorithm is proposed and the work in which that specific tool is used for comparison. Whichever strength is greater has been included in our analysis. We are aware of the fact that even while making an extensive search we might have missed some research work which would have given us a yet higher strength for a specific algorithm/tool. That can be a threat to the validity of our work. In addition to that the tools algorithms proposed after March 2014 have not been included.

Last, we detailed our measurement framework and we believe our characterizations are robust enough to be reliable, thus leading to trustworthy results.

VIII. CONCLUSION

Functional testing from a plain English specification, for instance following the category partition method, requires that one identifies parameters, categories, choices and then combine those choices according to some selection criteria, while accounting for constraints on choices, to eventually generate test cases. Covering arrays have been used for a long time to generate such combinations. Covering arrays come in various forms and have various capabilities and it is difficult to identify which covering array generation technology is the most suitable to the problem of generating test cases for the category partition method. When faced with this problem we searched for a solution and did not find enough data to make an enlightened one. We therefore decided to perform a systematic survey of technologies supporting covering array generation. We report in this paper on the procedure we followed in this systematic survey and on the procedure we followed to characterize the covering array technologies we have found.

We eventually identified 75 covering array generation technologies. Our comparison framework allowed us to make a number of observations.

We observe that different covering array construction technologies support different sets and numbers of selection criteria in different amounts: 43% of the greedy technique support up to three criteria; 46% of the meta-heuristic techniques support two criteria; 40% of the algorithms based on adaptive random techniques support up to three criteria.

We believe these differences are not intrinsic to the construction technologies: for instance, there is no reason to believe that meta-heuristic techniques (or hybrid ones) could not support the complete list of criteria we have listed previously in the paper, or higher strength values (at the expense perhaps of longer execution times); we conjecture greedy algorithms have been so far popular due to their simplicity. Some technologies support very high strength values (up to 14), and 70% of the tools do not support a strength greater than four. The cost-benefit of such values is, as far as we know, yet to be confirmed experimentally. We found that only 44% of the 75 tools support constraints, and that constraints are provided mostly either as forbidden tuples of formal specifications. Constraints are mostly handled by greedy construction techniques; however, again, there is no reason to believe other techniques could not equally handle constraints.

We observe that although metaheuristic, adaptive random/adhoc and algebraic techniques form a smaller part of the tool supporting covering array construction, they are equally focused on advance features for creating covering arrays as greedy based technologies. On the other hand tools/algorithms based on Greedy techniques are plenty in number which can be attributed to the fact that they are flexible to implement. They support large system configurations including constraints, selection criteria, mixed covering arrays and higher strengths, which is essentially a requirement for software testing.

Going back to our problem on identifying CA construction technology to support the category-partition testing method, whereby one needs that technology to handle constraints, variable numbers of choices per category (i.e., values of parameters), and selection criteria including at least pair-wise, we can conclude the following: a greedy algorithm is likely the best choice to date as this kind of technology supports selection criteria, various strength and constraints; in case there are few or simple constraints, a user may be able to spell out forbidden tuples and use a greedy algorithm that accepts such input (e.g., [63-65]); in case of complex or numerous constraints, manually constructing forbidden tuples may not be practical so a greedy algorithm that uses an adhoc algorithm for constraints (e.g., [48, 56]) or that incorporates a SAT solver (e.g., [33, 66, 67]) may be the ideal choice.

REFERENCES

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

[2] Grochtmann M. and Grimm K., "Classification trees for Partition Testing," JSTVR, 3 (2), pp. 63-82, 1993.

[3] Malaiya Y. K., "Antirandom Testing: Getting the most out of black box testing," Proc. ISSRE'95, pp. 86-95, 1995.

[4] Chen T. Y., Tang S.-F., Poon P.-L. and Tse T., "Identification of Categories and Choices in Activity Diagrams.," Proc. QSIC, pp. 55-63, 2005.

[5] Myers G. J., *The Art of Software Testing*, John Wiley & Sons, 1979.

[6] Ostrand T. J. and Balcer M. J., "The category-partition method for specifying and generating fuctional tests," 31(6), pp. 676-686, 1988.

[7] Kuhn D. R., Lei Y. and Kacker R. N., *Introduction to Combinatorial testing*, CRC Press, 2013.

[8] Grindal M., Offutt J. and Andler S. F., "Combination testing strategies: A survey," JSTVR, 15, pp. 167-199, 2005.

[9] Yilmaz C., Fouché S., Cohen M. B., Porter A., Demiroz G. and Koc U., "Moving Forward with Combinatorial Interaction Testing," 47(2), pp. 37-45, 2014.

[10] Alsewari A. R. A. and Zamli K. Z., "Design and implementation of a harmony-search-based variable-strength t-way testing strategy with constraints support," IST, 54 (6), pp. 553-568, 2012.

[11] Cohen M. B., Dwyer M. B. and Shi J., "Interaction Testing of Highly-Configurable Systems in the Presence of Constraints," ISSTA, pp. 129-139, 2007.

[12] Kuhn D. R., James M. H., James F. L., Raghu N. K. and Yu L., "Combinatorial Methods for Event Sequence Testing," Proc. ICST pp. 601-609, 2012.

[13] Martinez C., Moura L., Panario D. and Stevens B., "Locating errors using ELAs, Covering arrays and adaptive testing algorithms," JDS, 23 (4), pp. 1776-1799, 2009.

[14] Yilmaz C., "Test Case-Aware Combinatorial Interaction Testing," TSE, 39 (5), pp. 684-706, 2013.

[15] Demiroz G. and Yilmaz C., "Cost aware combinatorial interaction testing," Proc. VALID, pp. 9-16, 2012.

[16] Sandro F., Myra B. C. and Adam P., "Incremental covering array failure characterization in large configuration spaces," Proc. ISSTA, pp. 177-188, 2009.

[17] Turban R. C., Algorithms for covering arrays, Thesis, Arizona State University, 2006

[18] Kuliamin V. V. and Petukhov A. A., "A survey of methods for constructing Covering Arrays," PCS, 37 (3), pp. 121-146, 2011.

[19] Kuliamin V. and Petukhov A., "Covering array Generation Method Survey," Proc. ISoLA, 6416, pp. 382-396, 2010.

[20] Nie C. and Leung H., "A survey of combinatorial testing," ACM, 43 (2), pp. 1-29, 2011.

[21] Ahmed B. S. and Zamli K. Z., "A Review of Covering arrays and their applications to Software Testing," JCS, 7 (9), pp. 1375-1385, 2011.

[22] Rahman A., Al-Sewari A. and Zamli K. Z., "An Orchestrated Survey on T-Way Test Case Generation Strategies Based on Optimization Algorithms," Proc. ROVISP, 291, pp. 255-263, 2014.

[23] Anand S., Burke E. K., Chen T. Y., Clark J., Cohen M. B., Grieskamp W., Harman M., Harrold M. J. and Mcminn P., "An orchestrated survey of methodologies for Automated Software Test Case Generation," JSS, 86 (8), pp. 1978-2001, 2013.

[24] Kitchenham B. and Charters S., "Guidelines for performing systematic literature reviews in software engineering. ," Keele University, 2007.

[25] Arksey H. and O'Malley L., "Scoping Studies: Towards a methodological framework," IJSRM, 8 (2), pp. 19-32, 2005.

[26] Cohen M. B., Designing Test Suites for Software Interaction Testing, Thesis, New Zealand, Computer Science, 2004

[27] Al-Khiro M. I. Y., MIPOG: A Parallel T-Way Minimization strategy for combinatorial testing, Thesis, School of Electrical and Electronic Engineering, 2010

[28] Yu L., Advanced Combinatorial testing algorithms and applications, Thesis, The University of Texas at Arlington, 2013

[29] Kuhn D. R., Kacker R. N. and Lei Y., "Practical Combinatorial Testing," National Institute of Standards and Technology, NIST Special Publication 800-142, 2010.

[30] Bryce R. C., Colbourn C. J. and Kuhn D. R., "Finding Interaction Faults using distance based strategies," Proc. ECBS, pp. 4-13, 2011.

[31] Sciences A. C., "The AETG Web Service ", http://aetgweb.argreenhouse.com/ (Last accessed May, 2014)

[32] Cohen D. M., Dalal S. R., Fredman M. L. and Patton G. C., "The AETG System: An approach to testing based on combinatorial design," TSE, 23 (7), pp. 437-444, 1997.

[33] Yu L., Lei Y., Kacker R. N. and Kuhn D. R., "ACTS: A Combinatorial Test Generation Tool," Proc. ICST, pp. 370-375, 2013.

[34] Lei Y., Kacker R., Kuhn D. R., Okun V. and Lawrence J., "IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing," JSTVR, 18 (3), pp. 125-148, 2008.

[35] Forbes M., Lawrence J., Lei Y., Kacker R. N. and Kuhn D. R., "Refining the In-Parameter-Order Strategy for Constructing Covering Arrays," JRNIST, 113, pp. 287-297, 2008.

[36] Yu L., Lei Y., Nourozborazjany M., Kacker R. N. and Kuhn D. R., "An Efficient Algorithm for Constraint Handling in Combinatorial Test Generation," Proc. ICST, pp. 242-251, 2013.

[37] Tai K.-C. and Lei Y., "A test generation strategy for pairwise testing," TSE, 28 (1), pp. 109-111, 2002.

[38] Ahmed B. S. and Zamli K. Z., "A variable strength interaction test suites generation strategy using Particle Swarm Optimization," JSS, 84 (12), pp. 2171-2185, 2011.

[39] Bryce R. C. and Colbourn C. J., "One-test-at-a-time heuristic search for interaction test suites," Proc. GECCO pp. 1082-1089, 2007.

[40] Lei Y. and Tai K. C., "In-parameter-order: A test generation strategy for pairwise testing," Proc. HASE, pp. 254-261, 1998.

[41] Williams A. W., "Determination of Test Configurations for Pair-Wise Interaction Coverage," Proc. TestCom, 2000.

[42] Williams A. W. and Probert R. L., "A measure for component interaction test coverage.," Proc. AICCSA 2001, 2001.

[43] Cohen M. B., Gibbons P. B., W.B M. and Colburn C. J., "Constructing test cases for interaction testing," Proc. ICSE'03, pp. 38-48, 2003.

[44] Schroeder P. J. and Korel B., "Black-Box Test Reduction Using Input-Output Analysis," Proc. ISSTA, pp. 173-177, 2000.

[45] Grindal M., Lindström B., Offutt J. and Andler S. F., "An Evaluation of Combination Strategies for Test Case Selection," JESE, 11 (4), pp. 583-611, 2006.

[46] Othman R. R. and Zamli K. Z., "T-Way Strategies and Its Applications for Combinatorial Testing " IJNCAA, 1 (2), pp. 459-473, 2011.

[47] Labiche Y. and Sadeghi F. R., "Experimenting with Category Partition's 1-way and 2-way test selection criteria," Proc. ICST, pp. 301-310, 2013.

[48] Segall I., Tzoref-Brill R. and Farchi E., "Using Binary Decision Diagrams for Combinatorial Test Design," Proc. ISSTA'11, pp. 254-264, 2011.

[49] Grindal M., Offutt J. and Mellin J., "Handling Constraints in the Input Space when Using Combination Strategies for Software Testing," School of Humanities and Informatics, University of Sk¨ovde, 2006.

[50] Toczki J., Kocsis F., Gyimothy T., Danyi G. and Kokoi G., "SYS/3- A software Development tool," LNCS, 477, pp. 193-207, 1991.

[51] Garvin B. J., Cohen M. B. and Dwyer M. B., "Evaluating improvements to a meta-heuristic search for constrained interaction testing," JESE, 16 (1), pp. 61-102, 2011.

[52] Huang R., Xie X., Chen T. Y. and Lu Y., "Adaptive Random Test Case Generation for Combinatorial Testing," Proc. COMPSAC, pp. 52-61, 2012.

[53] Khatun S., Rabbi K. F., Yaakub C. Y. and Klaib M. F. J., "A Random Search Based Effective Algorithm for Pairwise Test Data Generation," Proc. INECCE, pp. 293-297, 2011.

[54] Bach J., "Allpairs Test Case Generation Tool," 1.2.1, http://www.satisfice.com/tools.shtml (Last accessed Feb, 2014)

[55] Cohen M. B., Colbourn C. J. and Ling A. C. H., "Augmenting Simulated Annealing to Build Interaction Test Suites," Proc. ISSRE, pp. 394, 2003.

[56] Czerwonka J., "Pairwise testing in real world," Proc. PNSQC, pp. 419-430, 2006.

[57] Project M., "Tcases- A model driven test case generator," 1.1.0, https://code.google.com/p/tcases/ (Last accessed March, 2014)

[58] Atyoursideconsullting, "ATD," http://www.atyoursideconsulting.com/products/atd/atd_funcfeatures_tcg.html (Last accessed Feb,2014)

[59] Ltd I. S. a. S. C., "PictMaster," 5.7.3, http://en.sourceforge.jp/projects/pictmaster/ (Last accessed)

[60] Zamli K. Z., Klaib M. F. J., Younis M. I., Isa N. A. M. and Abdullah R., "Design and implementation of a t-way test data generation strategy with automated execution tool support," JIS, 181 (9), pp. 1741-1758, 2011.

[61] Othman R. R., Zamli K. Z. and Nugroho L. E., "General variable strength t-way strategy supporting flexible interactions," MIJST, 6 (3), pp. 415-429, 2012.

[62] Othman R. R. and Zamli K. Z., "ITTDG: Integrated T-way test data generation strategy for interaction testing " SRE, 6 (17), pp. 3638-3648, 2011.

[63] Jenkins B., "Jenny," http://www.burtleburtle.net/bob/math/jenny.html (Last accessed April, 2014)

[64] Calvagna A. and Gargantini A., "T-wise combinatorial interaction test suites construction based on coverage inheritance," JSTVR, 22 (7), pp. 507-526, 2012.

[65] Li L., Cui Y. and Yang Y., "Combinatorial Test Cases with Constraints in Software Systems " Proc. CSCWD, pp. 195-199, 2012.

[66] Calvagna A. and Gargantini A., "A Logic-based approach to combinatorial testing with constraints," LNCS, 4966, pp. 66-83, 2008.

[67] Zhao Y., Zhang Z., Yan J. and Zhang J., "Cascade: A Test Generation Tool for Combinatorial Testing," Proc. ICST, pp. 267-270, 2013.

[68] Tung Y.-W. and Aldiwan W. S., "Automating Test Case Generation for the New Generation Mission Software System," Proc. AeroConf, 1, pp. 431-437, 2000.

[69] Cohen M. B., Dwyer M. B. and Shi J., "Construction Interaction Test Suites for Highly Configurable Systems in the Presence of Constraints: A greedy Approach," TSE, 34 (5), pp. 633-650, 2008.

[70] Ltd I. S. a. S. C., "PICTMaster," 5.7.3, http://en.sourceforge.jp/projects/pictmaster/ (Last accessed March, 2014)

[71] Hartman A., "IBM Intelligent test case handler," 1.0, http://www.alphaworks.ibm.com/tech/whitch (Last accessed Dec 2013)

[72] Schroeder P. J., Arshem J., Kim A. E. and Bolaki P., "Combining Behavior and Data Modeling in Automated Test Case Generation," Proc. QSIC, pp. 247-254, 2003.

[73] Arshem, "TVG," http://sourceforge.net/projects/tvg (Last accessed April 2014)

[74] Schroeder P. J., Faherty P. and Korel B., "Generating Expected Results for Automated Black-Box Testing," Proc. ASE, pp. 139-148, 2002.

[75] Wang Z., Xu B. and Nie C., "Greedy Heuristic Algorithms to Generate Variable Strength Combinatorial Test Suite " Proc. QSIC, pp. 155-160, 2008.

[76] Wang Z., Nie C. and Xu B., "Generating Combinatorial Test Suite for Interaction Relationship " Proc. SOQUA, pp. 55-61, 2007.

[77] Sherwood G. B., "Effective Testing of Factor Combinations," Proc. STAR, 1994.

[78] Colbourn C., Cohen M. and Turban R. C., "A Deterministic Density Algorithm for Pairwise Interaction Coverage," Proc. ICSE, pp. 245-252, 2004.

[79] Bryce R. C. and Colbourn C. J., "A Density-Based Greedy Algorithm for Higher Strength Covering Arrays," JSTVR, 19 (1), pp. 37-53, 2009.

[80] Wang Z. and He H., "Generating Variable Strength Covering Array for Combinatorial Software Testing with Greedy Strategy " JS, 8 (12), pp. 3173-3181, 2013.

[81] Ong H. Y. and Zamli K. Z., "Development of interaction test suite generation strategy with input-output mapping supports " SRE, 6 (16), pp. 3418-3430, 2011.

[82] Kuhn D. R., Kacker R. N. and Lei Y., "Practical combinatorial testing," 2010.

[83] McCaffrey J., "Pairwise Testing with QICT," 24(9), pp., 2009.

[84] Zhao Y., Zhang Z., Yan J. and Zhang J., "Cascade: A Test Generation Tool for Combinatorial testing," Proc. Cascade: A Test Generation Tool for Combinatorial testing, pp. 267-270, 2013.

[85] Raaphorst S., Variable strength Covering arrays, Thesis, University of Ottawa, 2013

[86] Klaib M. F. J., Zamli K. Z., Isa N. A. M., Younis M. I. and Abdullah R., "G2Way - A Backtracking Strategy for Pairwise Test Data Generation " Proc. APSEC, pp. 463-470, 2008.

[87] Rabbi K. F., Beg A. H. and Herawan T., "MT2Way: A Novel Strategy for Pair-Wise Test Data Generation," ISICA 2012, CCIS, pp. 180-191, 2012.

[88] Rabbi K. F., Khatun S., Yaakub C. Y. and Klaib M. F. J., "EPS2Way: An Efficient Pairwise Test Data Generation Strategy " IJNCAA, 1 (4), pp. 1099-1109, 2011.

[89] Chen X., Gu Q., Qi J. and Chen D., "Applying Particle Swarm Optimization to Pairwise Testing," Proc. COMPSAC, pp. 107-116, 2010.

[90] Yuan J., Jiang C. and Jiang Z., "Improved Extremal Optimization for Constrained Pairwise Testing," Proc. ICRCCS, pp. 108-111, 2009.

[91] Garvin B. J., Cohen M. B. and Dwyer M. B., "An Improved Meta-heuristic Search for Constrained Interaction Testing," Proc. SSBSE, pp. 13-22, 2009.

[92] McCaffrey J. D., "Generation of Pairwise Test Sets Using a Genetic Algorithm," Proc. COMPSAC, pp. 626-631, 2009.

[93] Shiba T., Tsuchiya T. and Kikuno T., "Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing," Proc. COMPSAC'04, pp. 72-77, 2004.

[94] Stardom J., Metaheuristics and the Search for Covering and Packing Arrays., PhD Thesis Thesis, Simon Fraser University, 2001

[95] Flores P. and Cheon Y., "PWiseGen: Generating Test Cases for Pairwise Testing Using Genetic Algorithms " Proc. CSAE, pp. 747-752, 2011.

[96] Chen X., Gu Q., Li A. and Chen D., "Variable Strength Interaction Testing with an Ant Colony System Approach," Proc. APSEC, pp. 160-167, 2009.

[97] LI J., Xing D. and Zhao Y., "Combinatorial Test Suite Generation of Variable Strength Based on Harmony Search " JNIS, 4 (2), pp. 177-188, 2013.

[98] Gonzalez-Hernandez L., Rangel-Valdez N. and Torres-Jimenez J., "Construction of Mixed Covering Arrays of Variable Strength Using a Tabu Search Approach," Proc. COCOA, 6508, pp. 51-64, 2010.

[99] Younis M. I., Zamli K. Z. and Isa N. A. M., "IRPS – An Efficient Test Data Generation Strategy for Pairwise Testing," Proc. KES, pp. 493-500, 2008.

[100] Younis M. I. and Zamli K. Z., "MIPOG - An Efficient t-Way Minimization Strategy for Combinatorial Testing " IJCTE, 3 (3), pp. 388-397, 2011.

[101] VpTag, "Visual Pairwise Test Array Generator," http://vptag.sourceforge.net/ (Last accessed May, 2014)

[102] Williams A. W., Software Component Interaction Testing: Covergae Measurement and Generation of Configurations, Thesis, University of Ottawa, School of Information Technology and Engineering, 2002

[103] Yan J. and Zhang J., "Backtracking algorithms and search heuristics to generate test suites for combinatorial testing," Proc. COMPSAC, pp. 385-394, 2006.

[104] Yan J. and Zhang J., "A backtracking search tool for constructing combinatorial test suites," JSS, 81 (10), pp. 1681-1693, 2008.

[105] Bracho-Rios J., Torres-Jimenez J. and Rodriguez-Tello E., "A New Backtracking Algorithm for Constructing Binary Covering Arrays of Variable Strength," Proc. MICAI, pp. 397-407, 2009.

[106] Hartman A. and Raskin L. P., "Problems and algorithms for covering arrays," JDM, 284 (1-3), pp. 149-156, 2004.

[107] Sherwood G., "TestCover," http://testcover.com/pub/constex.php (Last accessed Feb, 2014)

[108] Kobayashi N., suchiya T. and Kikuno T., "A new method for constructing pair-wise covering designs for software testing," IPL, 81 (2), pp. 85-91, 2002.

[109] Hunter J. Hexawise tool. Available: https://hexawise.com/ (Last accessed)

[110] SigmaZone, "Pro-Test," http://www.sigmazone.com/protest.htm (Last accessed Feb, 2014)

[111] Lewis W. E., "SmartTest," http://www.smartwaretechnologies.com/ (Last accessed Feb, 2014)

[112] INC. B., "BenderRBT," 8.0, http://www.benderrbt.com/bendersoftware.htm (Last accessed Feb 2014)

[113] Project C., "Tcases," https://code.google.com/p/tcases/ (Last accessed 19th May 2014)

IX. APPENDIX

TABLE III. LIST OF TOOLS/ALGORITHMS FOR GENERATING TEST SUITES USING COMBINATORIAL TESTING CATEGORIZED ON THE BASIS OF TECHNIQUES

Generation Strategy	**Greedy Techniques**	**Meta heuristic Techniques**	**Adaptive Random and Adhoc Techniques**	**Hybrid Techniques**	**Algebraic Techniques**
Test Based Generation	AETG Web Service [26, 31, 32] Test Case Generator [68] mAETG_SAT [11, 69] ATGT[66] PICT [56] PictMaster[70] Exhaustive search-Intelligent Test case handler (WHITCH)[71] Jenny [63] Test Vector Generator (TVG) [72, 73] GVS [61] Union [44] Greedy [74] Density [75] ReqOrder in [76] CATS [77] Deterministic density algorithm [78] Density Based Greedy [79] DA-RO[80] DA-FO [80] ITTDG [62] AURA [81]	Particle Swarm Based Algorithm (OTAT)[89] Extremal optimization based algorithm [90] CASA [11, 51, 91] Genetic Algorithms–GAPTS [92] Genetic Algorithm Based [93, 94] GA based - PWiseGen[95] Ant Colony algorithms[93] Ant Colony System(ACS) [96] Harmony search strategy [10] Particle Swarm Test Generator VS-PSTG [38] HSTCG [97] Tabu Search [98]	IRPS [99] R2Way [53] ART-CT [52] Distance Based Technique [30] AllPairs [54]	Greedy Algorithm with Hill Climbing [39] Greedy Algorithm with Simulated annealing [39] Greedy Algorithm with Great Flood [39] Greedy Algorithm with Tabu Search [39]	

	Sequence Covering Array Generator [82] QICT [83] Cascade [84] IBM Focus [48] VarDens [85] G2way [86] GTWay [60] MT2Way [87] EPS2way [88]				
Parameter based generation	PairTest [37]. ParaOrder [75] ACTS [33] tTuples [64] CTWC [65] MIPOG [100] VpTag[101] TConfig (IPO based) [102] EXACT [103, 104] Branch and Bound [105]	Particle Swarm Based Algorithm(OPAT) [89]		IPOD (IPOG and Algebraic Technique)[34] Augmented Annealing-combines Simulated Annealing and Algebraic Technique[26, 55]	Tconfig [41] Combinatorial Test Services (CTS) [106] Test Cover [107] Algebraic method [108]

TABLE IV. TOOLS/ALGORITHM FOUND WITH NO DETAILED TECHNICAL INFORMATION

S.no	**Name of Tool Algorithm**	
1	T-Gen -SYS/3 - a Software Development Tool	[50]
2	Hexawise	[109]
3	ProTest	[110]
4	SmartTest	SmartTest [111]
5	ATD	[58]
6	BenderRBT	BenderRBT [112]
7	Tcases	[113]

TABLE V. SELECTION CRITERIA SUPPORTED BY THE TOOL/ALGORITHM

S. No	Algorithm/tool	Each Choice	Base Choice	Variable Strength	Uniform strength	Input output based	Distance based	Random Input	All Combinations
1	AETG Web Service [26, 31, 32]				Yes				
2	PairTest [37].				Yes				
3	mAETG_SAT [11, 69]				Yes				
4	ATGT[66]				Yes				
5	ACTS [33]		Yes	Yes	Yes				
6	tTuples [64]				Yes				
7	Particle Swarm Based Algorithm(OTAT) [89]				Yes				
8	Extremal optimization based algorithm [90]				Yes				
9	CASA [11, 51, 91]			Yes	Yes				
10	Particle Swarm Based Algorithm (OPAT) [89]				Yes				
11	CTWC [65]				Yes				
12	PICT [56]		Yes (weights)	Yes	Yes				
13	MT2Way [87]				Yes				
14	EPS2way [88]				Yes				
15	IRPS [99]				Yes				
16	G2way [86]				Yes				
17	GTWay[60]				Yes				
18	Intelligent Test case handler (WHITCH)[71]			Yes	Yes				
19	Jenny [63]				yes				
20	Test Vector Generator (TVG) [72] [73]			Yes	Yes	Yes			
21	Tconfig [41]				Yes				
22	TConfig (IPO based) [102]				Yes				
23	GVS [61]			Yes	Yes	Yes			
24	Union [44]					Yes			
25	Greedy [74]			Yes	Yes	Yes			
26	ReqOrder in [76]					Yes			
27	Density [75]			Yes	Yes	Yes			
28	ParaOrder [75]			Yes	Yes	Yes			
29	Genetic Algorithms [93, 94]				Yes				
30	Ant Colony algorithms[93]				Yes				
31	Genetic Algorithm - GAPTS [92]				Yes				
32	Ant Colony System(ACS) [96]			Yes	Yes				
33	Greedy Algorithm with Hill Climbing [39]				Yes				
34	Greedy Algorithm with Simulated annealing [39]				Yes				
35	Greedy Algorithm with Great Flood [39]				Yes				
36	Greedy Algorithm with Tabu Search [39]				Yes				
37	Combinatorial Test Services (CTS) [106]				Yes				
38	Augmented Annealing-combines Simulated Annealing and Algebraic Technique[55]				yes				
39	IPOD (IPOG and Algebraic Technique)[34].				Yes				
40	CATS [77]				Yes				

41	Test Cover [107]				Yes				
42	Algebraic method [108]				yes				
43	Deterministic density algorithm [78]				Yes				
44	Density Based Greedy [79]				Yes				
45	DA-RO[80]			Yes	Yes	Yes			
46	DA-FO [80]			Yes	Yes	Yes			
47	Test Case Generator [68],				Yes				
48	R2Way [53]				Yes				
49	ART-CT [52]				Yes		Yes		
50	MIPOG [100]				Yes				
51	ITTDG [62]			Yes	Yes	Yes			
52	AURA [81]			Yes	Yes	Yes			
53	Harmony search strategy [10]			Yes	Yes *				
54	Particle Swarm Test Generator VS-PSTG [38]			yes	Yes*				
55	HSTCG [97]			Yes	Yes*				
56	EXACT [103, 104]				Yes				
57	Branch and Bound [105]			Yes*	Yes				
58	Tabu Search [98]			Yes*	Yes				
59	Distance Based Technique [30]				Yes		Yes	Yes	
60	T-Gen SYS/3 - a Software Development Tool [50]	Yes							
61	Sequence Covering Array Generator [82]				Yes				
62	Hexawise [109]			Yes	Yes				
63	QICT [83]				Yes				
64	Cascade [84]			Yes	Yes				
65	AllPairs [54]				Yes				
66	ProTest[110]				Yes				
67	VpTag[101]				Yes				
68	PictMaster[70]		Yes (weights)		Yes				
69	SmartTest [111]				Yes				
70	ATD [58]				Yes				
71	BenderRBT [112]				Yes				
72	IBM Focus [48]		Yes (weights)	Yes	Yes				
73	PWiseGen[95]				Yes				
74	VarDens [85]			Yes*	Yes				
75	Tcases [113]	Yes		Yes	Yes			Yes	

* Information Not Available

TABLE VI. MAXIMUM COVERAGE STRENGTH SUPPORT

S.No	**Algorithm/tool**	**Maximum Strength support (t)**	**Number of parameters and values**
1	AETG Web Service [26, 31, 32]	2	MCA(N, t, 4^1,3^{39}, 2^{35})
2	PairTest [37]	2	MCA(N, t, 4^1, 3^{39}, 2^{35})
3	mAETG_SAT [11, 69]	3	CCA(N, t, 2^{158}, 3^8, 4^4, 5^1, 6^1, F)
4	ATGT[66]	2	MCA(N, t, 4^1,3^{39}, 2^{35})
5	ACTS [33]	6	MCA(N, t, 10^2, 4^1, 3^2, 2^7)
6	tTuples [64]	6	MCA(N, t, 4^5, 2^{13})
7	Particle Swarm Based Algorithm(OTAT) [89]	2	MCA(N, t, 4^1, 3^{39}, 2^{35})
8	Extremal optimization based algorithm [90]	2	CCA(N, t, 2^{158}, 3^8, 4^4, 5^1, 6^1, t)

9	CASA [11, 51, 91]	3	CCA(N, t, 3^1, 2^4,F)
10	Particle Swarm Based Algorithm (OPAT) [89]	2	MCA(N, t, 4^1, 3^{39}, 2^{35})
11	CTWC [65]	5*	Information Not Available
12	PICT [56]	6	VSCA(N,3,3^{15}{CA(6,3^9)})
13	MT2Way [87]	2	CA(N, t, 3^4)
14	EPS2way [88]	2	CA(N, t, 3^4)
15	IRPS [99]	2	MCA(N, t, 5^1, 3^8, 2^2)
16	G2way [86]	2	MCA(N, t, 5^1, 3^8, 2^2)
17	GTWay(OTAT iterative) [60]	<=12	MCA(N, t, 10^2, 4^1, 3^2, 2^7)
18	Intelligent Test case handler (WHITCH)[71]	6	VSCA(N, 2, 10^1, 9^1, 8^1, 7^1, 6^1, 5^1, 4^1, 3^1 , 2^1, {MCA(6, 7^1, 6^1, 5^1, 4^1, 3^1, 2^1)})
19	Jenny [63]	<=8)	MCA(N, t, 10^2, 4^1, 3^2, 2^7)
20	Test Vector Generator (TVG) [72] [73]	6	VSCA(N, 2, 10^1, 9^1, 8^1, 7^1, 6^1, 5^1, 4^1, 3^1 , 2^1, {MCA(6, 7^1, 6^1, 5^1, 4^1, 3^1, 2^1)})
21	Tconfig [41]	2	MCA(N, t, 4^1, 3^{39}, 2^{35})
22	TConfig (IPO based) [102]	<=4	MCA(N, t, 10^2, 4^1, 3^2, 2^7)
23	GVS [61]	<=12	MCA(N, t, 10^2, 4^1, 3^2, 2^7)
24	Greedy [74]	3	MCA(N, t, 10^1, 6^2,4^3,3^1)
25	Density [75]	3	MCA(N, t, 10^1, 6^2,4^3,3^1)
26	ParaOrder [75]	3	MCA(N, t, 10^1, 6^2,4^3,3^1)
27	Genetic Algorithms- GAPTS [92]	2	MCA(N, t, 3^4, 3^{13}, 2^{100}, 10^{20})
28	Ant Colony algorithms(ACA) [93]	3	MCA(N, t, 10^1, 6^2,4^3,3^1)
29	Genetic algorithm based algorithm [93, 94]	3	MCA(N, t, 10^1, 6^2,4^3,3^1)
30	Ant Colony System(ACS) [96]	3	VSCA(N,2,3^{20},10^2, {MCA(3,3^{20},10^2)})
31	Greedy Algorithm with Hill Climbing [39]	4	MCA(N, t, 2^{10},3^3,4^2,5^1)
32	Greedy Algorithm with Simulated annealing [39]	4	MCA(N, t, 2^{10},3^3,4^2,5^1)
33	Greedy Algorithm with Great Flood [39]	4	MCA(N, t, 2^{10},3^3,4^2,5^1)
34	Greedy Algorithm with Tabu Search [39]	4	MCA(N, t, 2^{10},3^3,4^2,5^1)
35	Combinatorial Test Services (CTS) [106]	4	CA(N, t, 10^8)
36	Augmented Annealing-combines Simulated Annealing and Algebraic Technique[55]	3	CA(N, t, 14^{14})
37	IPOD (IPOG and Algebraic Technique)[34].	6	CA(N, t, 4^{15})
38	CATS [77]	3	CA(N, t, 6^4)
39	Test Cover [107]	4*	Information Not Available
40	Algebraic method [108]	2	MCA(N, t, 4, 3^{39}, 2^{35})
41	Deterministic density algorithm [78]	2	MCA(N, t, 4^1, 3^{39}, 2^{35})
42	Density Based Greedy [79]	6	CA(N, t, 5^{10})
43	DA-RO[80]	3	MCA(N, t, 10^1, 6^2, 4^3, 3^1)
44	DA-FO [80]	3	MCA(N, t, 10^1, 6^2, 4^3, 3^1)
45	Test Case Generator [68]	2 , t-wise*	MCA(N, t, 5^1, 3^8, 2^2)
46	R2Way [53]	2	CA(N, t, 3^4)
47	ART-CT [52]	4	MCA(N, t, 2^5, 3^5)
48	MIPOG [100]	11	MCA(N, t, 5^7, 2^4)
49	ITTDG [62]	12	MCA(N, t, 10^2, 4^1, 3^2, 2^7)
50	AURA [81]	3	MCA(N, t, 10^1,6^1,4^3,3^1)
51	Harmony search strategy [10]	14	VSCA(N,3,3^{15},{CA(14,3^{14})})
52	Particle Swarm Test Generator VS-PSTG [38]	6	VSCA(N,2,3^{15},{CA(6,7^1,6^1,5^1,4^1,3^1,2^1)})
53	HSTCG [97]	7	VSCA(N,2,4^3, 5^3, 6^2, {CA(7, 4^3, 5^3, 6^2)})
54	EXACT [103, 104]	5	CA(N, t, 2^6)
55	Tabu Search [98]	6	MCA(N, t, 2^2,3^2,4^2,5^2)
56	Branch and Bound [105]	5	CA(N, t, 2^6)
57	Distance Based Technique [30]	5*	Information Not available
58	Sequence Covering Array Generator [82]	4	80 events
59	Hexawise [109]	6	Information obtained via chat with tool support
60	QICT [83]	2	MCA(N, t, 3^4, 3^{13}, 2^{100}, 10^{20})

61	Cascade [84]	2*	Information not available
62	AllPairs [54]	2, N-Wise*	MCA(N, t, 5^1, 3^8, 2^2)
63	ProTest [110]	2*	Information Not Available
64	VpTag[101]	2*	Information Not Available
65	PictMaster[70]	6*	Information Not Available
66	SmartTest [111]	2*	Information Not Available
67	ATD [58]	2 *N-wise*	Information Not Available
68	BenderRBT [112]	2*	Information Not Available
69	IBM Focus [48]	2	MCA(N, t, 4^1,3^{39},2^{35})
70	PWiseGen[95]	2	MCA(N, t, 4^1,3^{39},2^{35})
71	Tcases [113]	3*	Information Not Available
72	VarDens [85]	4	CA(N, t, 5^{10})

*Information Not Available

TABLE VII. CONSTRAINT HANDLING SUPPORT

S.No	**Algorithm/tool**	**Representation of the constraint (*Forbidden tuples, allowed tuples or full constraint (logical expression*)**	**Constraint handling Mechanism** **1.Constraints handled before executing test generation algorithm** **2. Replacing the invalid test cases** **3. Constraints handled by implementing an algorithm** **4. Constraints handled using SAT Solvers**
1	AETG Web Service [26, 31, 32]	Forbidden Tuples using if else expressions	Constraints handled by implementing algorithm
2	mAETG_SAT [11, 69]	Forbidden tuples converted into Boolean Formula	zChaff or MiniSAT SAT solver integrated into AETG algorithm. Solvers compute the constraints and AETG generates the test suites.
3	ATGT[66]	Full constraint support using prepositional logic	The combinatorial testing is represented as propositional logic problem including constraints (forbidden tuples) and SAL Constraint Solver is used to handle constraints and generate the test suite
4	ACTS [33] (IPOG-C [36]	Full constraint support using Boolean, relational and arithmetic operators based expressions	CHOCO Constraint Solver is integrated with the algorithm and is frequently called to handle constraints
5	tTuples [64]	Forbidden Tuples as Logical constraints	Greedy algorithm modified to handle constraints
6	Extremal optimization based algorithm [90]	Not enough information available	MiniSat Solver integrated with the Extremal Optimization algorithm
7	CASA [11, 51, 91]	Forbidden tuples converted into Boolean Formula	zChaff SAT Solver is integrated with the Simulated annealing algorithm
8	CTWC [65]	Forbidden tuples	Constraint handled by implementing an algorithm
9	PICT [56]	Full constraint support using Logical Expressions are used to define constraints	Forbidden tuples are obtained from logical expressions and then algorithm is implemented for handling constraints
10	Intelligent Test case handler (WHITCH) [71]	Forbidden tuples [11]	Information Not available
11	Jenny [63]	Forbidden tuples expressed as string of numbers and characters	Constraint handled by implementing an algorithm
12	Test Vector Generator (TVG) [72, 73]	Full Constraint support with Logical Expressions using	Constraint handled by implementing an algorithm

		relational operators	
13	Combinatorial Test Services (CTS) [106]	Forbidden Tuples	Constraints handled by implementing an algorithm.
14	CATS [77]	Allowed Tuples	Constraints handled before executing test generation algorithm
15	Test Cover [107]	Allowed Tuples	No description but it can be assumed that constraints are handled before giving to algorithm
16	Test Case Generator [68]	Full Constraint support as Logical Expressions	Constraints handled by implementing an algorithm
17	Harmony search strategy [10]	Information not available *	Constraints handled by implementing an algorithm
18	HSTCG [97]	Full constraint support. Paper discusses that the approach supports complex constraints with no further discussion	Constraints handled by implementing an algorithm
19	Distance Based Technique [30]	Forbidden tuples	Constraints handled by implementing an algorithm
20	T-Gen SYS/3 - a Software Development Tool [50]	Information not available *	No information available *
21	Sequence Covering Array Generator [82]	Forbidden tuples (excluded sequences)	No information available *
22	Hexawise [109]	Forbidden tuples	No information available *
23	Cascade [84]	Full constraint support using Boolean, relational and arithmetic operators based expressions	A pseudo-Boolean optimization (PBO) solver called clasp is used to handle constraints and optimize coverage. Constraint solving and optimization is integrated
24	ProTest [110]	Information not available*	Information not available*
25	VpTag[101]	Full constraint support. Paper discusses that the approach supports complex constraints with no further discussion	Information not available*
26	PictMaster[70]	Full constraint support using Logical Expressions are used to define constraints	Forbidden tuples are obtained from logical expressions and then algorithm is implemented for handling constraints
27	SmartTest [111]	Full constraint support using Logical Expressions are used to define constraints	Information not available*
28	BenderRBT [112]	Full constraint support using Logical Expressions are used to define constraints	Information not available*
29	IBM Focus [48]	Full constraint support using Boolean Expressions in Java Syntax	Constraints handled by implementing an algorithm
30	Tcases [113]	Full constraint support. Properties are assigned to values and conditions are defined which are finally converted to Boolean expressions	Information not available*
31	AllPairs [54]	Information Not Available *	Information not available *
32	Augmented Annealing [26, 55]	Forbidden tuples	Constraints are handled before giving input to algorithms (as disjoint rows in the form of seeds)

*Information Not Available

TABLE VIII. MIXED COVERING ARRAY SUPPORT

S.No	Algorithm/tool	Mixed Covering Array Support
1	AETG Web Service [26, 31, 32]	Yes
2	PairTest [37]	Yes
3	mAETG_SAT [11, 69]	Yes
4	ATGT[66]	Yes
5	ACTS [33]	Yes
6	tTuples [64]	Yes
7	Particle Swarm Based Algorithm(OTAT) [89]	Yes
8	Extremal optimization based algorithm [90]	Yes
9	CASA [11, 51, 91]	Yes
10	Particle Swarm Based Algorithm (OPAT) [89]	Yes
11	CTWC [65]	Information Not Available *
12	PICT [56]	Yes
13	MT2Way [87]	Yes
14	EPS2way [88]	Yes
15	IRPS [99]	Yes
16	G2way [86]	Yes
17	GTWay(OTAT iterative) [60]	Yes
18	Intelligent Test case handler (WHITCH)[71]	Yes
19	Jenny [63]	Yes
20	Test Vector Generator (TVG) [72, 73]	Yes
21	Tconfig [41]	Yes
22	TConfig (IPO based) [102]	Yes
23	GVS [61]	Yes
24	Union [44]	Yes
25	Greedy [74]	Yes
26	ReqOrder in [76]	Yes
27	Density [75]	Yes
28	ParaOrder [75]	Yes
29	Genetic Algorithms- GAPTS [92]	Yes
30	Ant Colony algorithms(ACA) [93]	Yes
31	Genetic algorithm based algorithm [93, 94]	Yes
32	Ant Colony System(ACS) [96]	Yes
33	Greedy Algorithm with Hill Climbing [39]	Yes
34	Greedy Algorithm with Simulated annealing [39]	Yes
35	Greedy Algorithm with Great Flood [39]	Yes
36	Greedy Algorithm with Tabu Search [39]	Yes
37	Combinatorial Test Services (CTS) [106]	Yes
38	Augmented Annealing-combines Simulated Annealing and Algebraic Technique[55]	No
39	IPOD (IPOG and Algebraic Technique)[34].	Yes
40	CATS [77]	No
41	Test Cover [107]	Information Not Available *
42	Algebraic method [108]	Yes
43	Deterministic density algorithm [78]	Yes
44	Density Based Greedy [79]	Yes
45	DA-RO[80]	Yes
46	DA-FO [80]	Yes
47	Test Case Generator [68]	Yes
48	R2Way [53]	Yes
49	ART-CT [52]	Yes
50	MIPOG [100]	Yes
51	ITTDG [62]	Yes

52	AURA [81]	Yes
53	Harmony search strategy [10]	Yes
54	Particle Swarm Test Generator VS-PSTG [38]	Yes
55	HSTCG [97]	Yes
56	EXACT [103, 104]	Yes
57	Tabu Search [98]	Yes
58	Branch and Bound [105]	No
59	Distance Based Technique [30]	Information Not available
60	T-Gen SYS/3 - a Software Development Tool [50]	Information Not available
61	Sequence Covering Array Generator [82]	Information Not Available
62	Hexawise [109]	Information Not Available
63	QICT [83]	Yes
64	Cascade [84]	Information not available
65	AllPairs [54]	Yes
66	ProTest[110]	Information Not Available
67	VpTag[101]	Information Not Available
68	PictMaster[70]	Information Not Available
69	SmartTest [111]	Information Not Available
70	ATD [58]	Information Not Available
71	BenderRBT [112]	Information Not Available
72	IBM Focus [48]	Yes
73	PWiseGen[95]	Yes
74	Tcases [113]	Information Not Available
75	VarDens [85]	No