Journal of Computer Science 7 (9): 1375-1385, 2011
ISSN 1549-3636

A Review of Covering Arrays and Their Application to Software Testing

Bestoun S. Ahmed and Kamal Z. Zamli
Software Engineering Group, School of Electrical and Electronic Engineering,
University Sains Malaysia, 14300 Nibong Tebal, Seberang Perai Selatan,
Pulau Pinang, Malaysia

Abstract: Problem statement: As a complex logic system, software may suffer from different source of faults. Those faults can be avoided by applying different testing processes. It appears recently that the interaction among the system factors represents a common source of faults. Software function properly, all input factors and their interactions of the software need to be tested i.e., exhaustive testing. Random testing, in another hand, doesn't guarantee the coverage of all factors interaction. **Approach:** Covering Arrays (CAs) are mathematical objects used as platform or structure to represent the interactions of factors for a given system. The uses of CAs become important to reduce the test cases by covering all t-interactions of the system factors at least one time. **Results:** This study focuses exclusively on the applications of the CAs in software interaction testing. We provide an overview of CAs notations, types and construction methods. **Conclusion:** We reviewed the recent applications of CAs to software testing and discuss the future possible directions of the research. The research in this area seems to be an active research direction for the coming years.

Key words: Covering array, mixed covering array, interaction testing, testing processes, software testing, software system, parameters' values, construction methods, meta-heuristics

INTRODUCTION

Nowadays, our dependencies on software are straightforward and increasing, as many kinds of software have become a part of our daily lives. Unlike the old days, the development lifecycle of these software systems passes through several stages and comprehends different activities that need to harmonize carefully to meet the required user's specifications. Generally, those activities can be classified as two important activities, which are: activities to construct the software product and activities to check the quality of the produced software (Baresi and Pezze, 2006).

Although the construction of the product is important, however, checking the quality, which called the "quality process", represents the most important part of the software development lifecycle as it is spans through the whole cycle. The quality process gained importance because each development stage may suffer from different errors and faults, which must be detected as early as possible in order to prevent its propagation in the whole software and reduce the cost of verification. To regulate and show knowledge of the quality process, different levels of testing are used by the quality engineers during the software development lifecycle. This testing process helps to provide a realistic and practical way to analyze and understand the behavior of the produced software under test and to manage or mitigate the faults, risk and failure of the system (Agarwal *et al*., 2010). This can be tiresome, as when a computer game doesn't work properly, or it can be disastrous, resulting in the loss of life.

Unexpected Interactions among software system components represent a common source of software fault (Williams and Probert, 2001; Zamli and Isa, 2008). This risk is increasing when the numbers of software components are increased tremendously. To reduce this risk and ensure the quality of such software, the manufacture may need to test all the interaction among the components. For example, a complex software system with P components, each of which having two values, the manufacture needs 2^P test cases to test the software exhaustively. This in turn infeasible in practice due to different factors, including time, cost and resource constrains (Cohen *et al*., 2007; Williams and Probert, 2002). To this end, it is desirable to have a subset of all possible test interactions that have a high potential to uncover faults. Covering Arrays (CAs) recently appeared as an alternative to exhaustive testing by representing all the interactions of the components in a minimized array, which can be used as a test suite. In fact, by using this effective property of CAs, several applications in different areas of research have been introduced in the literature not only in software testing.

Corresponding Author: Kamal Z. Zamli, Software Engineering Group, School of Electrical and Electronic Engineering, University Sains Malaysia, 14300 Nibong Tebal, Seberang Perai Selatan, Pulau Pinang, Malaysia

However, in this study, we address the use of CAs in software testing exclusively because it is the wider research area in this direction.

The rest of this study is organized as follows. We give the definitions and mathematical notation first. Then we discuss the methods used in literature to construct the CAs. In addition, we declare the recent use of CAs in different software testing applications. Finally, we discuss the possible future directions of the research then we present the conclusions of the study.

Definitions and Preliminaries: Consider a software system with P parameters (components or factors), where P = {P_1, P_2, ….. P_j}. Each parameter P_j can take one of the possible values of V, where V= {v_1, v_2, ….. , v_i}. To test such a system, test cases could be constructed by assigning values to parameters then apply on the system. For example, the test case [$v_{(1,1)}$, $v_{(1,2)}$, $v_{(2,3)}$], considers three parameters of the system (P_1, P_2, P_3), by taking the first value v_1 of the first parameter, the first value v_1 for the second parameter and the second value v_2 for the third parameter.

An interaction "I" of the parameters is a set of values assigned to distinct parameters, considering the strength of interaction (t) among the parameters' values. We say that I=t-way if all the t-interactions of the parameters' values are taken. For example, if the system consists of three parameters P_1, P_2 and P_3, then the 2-way interaction (or pairwise) of the parameters is the representative values of {(P_1.P_2), (P_1,P_3), (P_2,P_3)}. To represent these interactions systematically, Orthogonal Array is introduced in an early stage.

Definition 1: An Orthogonal Array, denoted by OA_λ (N; t, p, v), is an array of size N and p components on v values and strength t, in which for every N×t sub-array, the t-interaction elements occur exactly λ times, where $\lambda=N/v^t$ (Ronneseth and Colbourn, 2009; Cheng, 1980; Beizer, 1990).

OA is often too restrictive because it requires the parameters' values to be uniform. In other words, it is required the same number of v_i for all the P_j parameters, to occur in the OA exactly one time, which is difficult when the parameters are growing. In addition, most of the time in practice, the values of the parameters are not uniform. To overcome these limitations, the covering array is emerged.

Definition 2: A Covering Array, denoted by CA_λ (N;t,p,v), represents an array of size N on v values, such that every N×t sub-array contains all ordered subsets from the v values of size t at least λ times and p is the number of components (Colbourn, 2008; Yilmaz *et al*., 2004a).

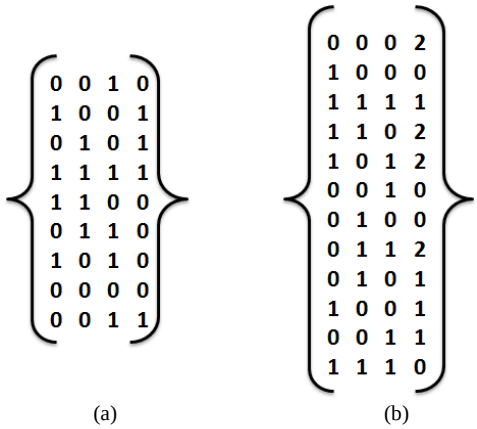


Fig. 1: The Representation of CA and MCA

Based on the CA construction, each column j (corresponding to a parameter P), contain all values of the corresponding parameters and every possible t-way interaction of the parameters' values is covered at least one time by a row. Hence, λ=1 and the notation becomes CA (N;t,k,v), because it is normally sufficient for each t-interaction to occur once in the CA. Figure 1a represents a CA with the notation CA (9; 3, 2^4) of size nine (i.e. nine rows) for a system with four parameters, each of which having two values.

Similar to the OA, in the CA all the parameters contain the same number of values, but the t-interactions in CA can occur more than one time. However, when the number of parameters' values varies, this can be handled by Mixed Covering Array (MCA), denoted by MCA (N;t,p,(v_1,v_2,…v_p)) (Colbourn *et al*., 2006). The notation can also be represented by MCA (N;t,p,v^k). Figure 1b represents a MCA with the notation MCA (12; 3, 2^3 3^1) of size 12 for a four parameters system, with a combination of four parameters having three values and five parameters having four values, to cover the four-way interactions. Building from CA and MCA notations, the variable-strength Covering Array is emerged.

Definition 3: A Variable-strength Covering Array, denoted by VSCA (N; t,p, (v_1, v_2, …, v_p) , C) represents an N×p mixed level covering array of strength t containing C, a vector of covering arrays and a subset of the p columns each of strength >t (Yilmaz *et al*., 2004b; Cohen, 2004).

Throughout this study, we use the term "main-strength" to describe the strength of VSCA and "sub-strength" to describe the strength of C. Here for example, VSCA (12; 2, 2^3 3^2, {CA (3, $2^2$$3^1$)}) represents a test suite of size 12 for a system of pair-wise interactions of five parameters in the main configuration

with a combination of three parameters having two values and two parameters having three values. In addition, a strength three sub-configuration of three parameters is available, with a combination of one parameter having three values and two parameters having two values.

As noted, despite the existence of MCA and VSCA in different constructions from the CA, but both are based on the CA. Hence, throughout this study, we refer to them as "CAs", unless there is a need to mention about a specific type of them.

CAs construction methods: As the CAs are used in testing processes, the first point to be considered in any construction method is the size minimization. It is desirable for any construction method to construct CAs to cover all the required t-interactions with a minimum number of rows. However, mathematically, the construction of optimum CAs is an NP-complete problem. Hence, it is difficult to find a unified construction method to construct optimum CAs all the time (Lei and Tai, 1998). Therefore, it is normal to find a method that can achieve minimum CAs sizes for some interactions, parameters, or values, while it cannot achieve that for others. To this end, it has been appeared that it is desirable for a method to construct minimum sizes with reasonable time in most cases. This in turn leads to the developments of different methods for construction.

Generally, CAs are constructed computationally to ensure minimization in terms of size and time. Mostly, the construction methods are called strategies because they are combination of different algorithms. We can classify the developed strategies under two general categories: (a) one-parameter-at-a-time strategies and (b) one-row-at-a-time strategies (Grindal *et al*., 2005). In case of one-parameter-at-a-time, the strategy constructs the array by adding one parameter's value to the array and check for the coverage of t-interactions "greedily". The greedy algorithm chooses the parameters' values that can cover more t-interactions. In-Parameter-Order-General (IPOG) (Lei *et al*., 2007) and its improvements, IPO-s (Calvagna and Gargantini, 2009), IPOG-F and IPOG-D (Lei *et al*., 2008) are the most recent strategies that adopt this construction method. Recently, Klaib *et al*. (2010) also proposed a tree based strategy for one-parameter-at-a-time test generation method.

Whereas, in case of one-row-at-a-time, in addition of using the greedy algorithm the strategy constructs one row and check for its coverage. The rows that are cover more t-interactions are chosen to form the final array. The Automatic Efficient Test Generator (AETG) (Cohen *et al*., 1997), mAETG (Cohen, 2004), Pairwise Independent Combinatorial Testing (PICT) (Czerwonka, 2006), Deterministic Density Algorithm (DDA) (Bryce and Colbourn, 2007; 2009), Classification-Tree Editor eXtended Logics (CTE-XL) (Lehmann and Wegener, 2000; Yu *et al*., 2003), Test Vector Generator (TVG) (Tung and Aldiwan, 2000) and Jenkins (2010) represent the most well-known strategies using this construction method.

Recently, the construction of CAs viewed as an optimization problem. As part of one-row-at-a-time construction method, meta-heuristics are used to achieve optimum number of rows. Different optimization methods are used in strategies for construction. From the published results, it has been appeared that these strategies can achieve better sizes in most cases but with longer construction time than other strategies (Afzal *et al*., 2009). So far, Simulated Annealing (SA) (Cohen, 2004; Stardom, 2001), Genetic Algorithm (GA) (Shiba *et al*., 2004), Ant Colony Algorithm (ACA) (Shiba *et al*., 2004) and Tabu Search (TS) (Nurmela, 2004), have been successfully implemented for small-scale interaction strength. We have also implemented Particle Swarm Optimization recently in a strategy named Particle Swarm Test Generator (PSTG) (Ahmed and Zamli, 2010).

Most of the strategies support the construction of CA and MCA. However, few strategies support the construction of VSCA. A number of the aforementioned strategies have started to support VSCA construction (e.g., PICT, IPOG, TVG, CTE-XL and SA). In addition to these strategies, a number of new strategies emerged to particularly construct VSCA using the published techniques. Wang *et al*. (2008) adopted the DDA algorithm for a strategy called Density to construct VSCA. Wang *et al*. (2008) also adopted the IPOG algorithm for a strategy called ParaOrder. Moreover, Chen *et al*. (2009) adopt the ACA algorithm for a strategy called Ant Colony System (ACS) to construct VSCA. Compared with CA and MCA, VSCA covers more t-interactions because of the sub-strength used in addition to the main-strength. Hence, the time of construction normally is more than CA and MCA in all the strategies. However, here also those strategies used meta-heuristics achieved better sizes most of the times (Afzal *et al*., 2009).

Applications: CAs have been used recently in different areas of research. In addition to its use in software testing, it has been used in hardware testing (Borodai and Grunskii, 1992), gene expression regulation (Shasha *et al*., 2001), advance material testing (Cawse, 2003), performance evaluation of communication systems (Hoskins *et al*., 2005) and many other research areas. Each area contains different applications. Software testing represents the wider area of research for CAs

application. The aim of using CAs in the form of interaction testing is to find faults in the application under test. Here, we review four main areas, which are: components interaction testing, GUI interaction testing, test case prioritization and regression testing as well as fault characterization.

Components interaction testing: In the old days, the trend by the software industries was to produce small software to achieve a specific aim. Nowadays, this phenomenon has been changed. The trend now is to produce software systems, in which they consist of individual programs working on individual components and connect these components together to collaborate. This collaboration leads to achieve a unified software system to serve different needs by the users, for example communications systems, storage systems, or e-commerce systems. Testing the components and programs of these systems individually is desirable and may leads to find different faults. However, it has been appeared that a common source of fault in the whole system comes from the unexpected interaction among the individual components of the system itself (Williams and Probert, 2001). As an example, we consider a software system based on the Internet in Table 1.

The system in Table 1 contains four components or parameters. The system may use different payment server, web server, user browser and business database. In order to test the system, it is required to test all the combinations or interactions among the components, but the tester may need 2^4 test cases for test. For this system, this number of test cases seems to be reasonable. However, if we have similar system but with 8 components, each of which having 4 variables, the test may need to test 4^8 =65,536 test cases, which seems to be infeasible for testing, due to time, cost and resources limitations. Using CA with 2-way, 3-way or above interaction seems to be a compromise method to guarantee the coverage of all interactions possibilities with minimum number of test cases. For example, all the 2-way interactions of the components for the system in Table 1 can be represented by a CA with size six as in Table 2. In additions, the 2-way interaction for the example 4^8 can be covered by a CA with size 27.

Different research proposed this solution. Williams and Probert (2001) proposed this solution formally and studied the coverage of interactions for a particular interaction strength. In another research (Williams and Probert, 1996), they proposed this solution for testing network interface using 2-way interaction. Although these researches proposed this solution, it was not applied practically on a real software system.

Table 1: A software system based on internet

Payment server	Web server	User browser	Business database
Master card	iPlanet	Chrome	SQL
Visa card	Apache	Mozilla	Oracle

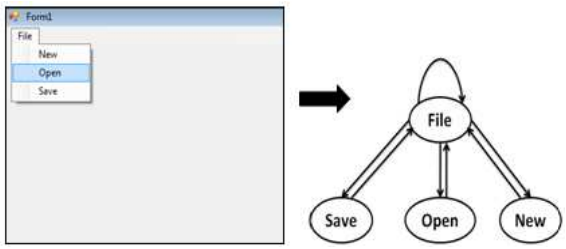
Table 2: 2-way Interaction for the System in Table 1

Payment server	Web server	User browser	Business database
Master card	iPlanet	Mozilla	SQL
Visa card	Apache	Chrome	Oracle
Master card	iplanet	chrome	oracle
Visa card	Apache	Mozilla	SQL
Visa card	iPlanet	Mozilla	Oracle
Master card	Apache	Chrome	SQL

Hoskins *et al*. (2005) applied CAs on a four categorical factors software system. The aim was to use CAs to measure the effect of type of database management system, platform, programmatic interface and type of indexing on the cache hit rate, number of page outs per second and number of physical reads per second. The results were compared with full factorial data and D-optimal designs construction. From the achieved results, CAs outperforms D-optimal designs and exhibit lower variance. The results support the contention that a CA whose strength is one less than the number of factors can be expected to outperform D-optimal designs.

GUI interaction testing: Graphical User Interface (GUI) has become practically the means of user interaction with any software. GUI testing is a process of software testing to test the GUI of software to ensure that it meets the required user specification (Memon, 2002). Most of the techniques used for GUI testing are incomplete and used ad hoc or manual testing. However, to formalize these kind of testing, recently, there are three main directions of research, which are using finite state machines (Robinson and White, 2008), pre and post-conditions (Li *et al*., 2007) and directed graph models (Memon and Xie, 2005) techniques. CAs have been used with graph models to effectively test GUI.

Graph models used the Event-Flow Graph (EFG) to model all possible event sequences that may be executed on a GUI (Huang *et al*., 2010). In such a model, each event in the GUI is represented by a node N and the relationship among the events is represented by an edge. For example, the GUI in Fig. 2a has four events, which are: File, New, Open and Save; whereas, Fig. 2b is the representation of the GUI in node and edge form. For two nodes GUI n_x and n_y, the edge from n_x to n_y means that, along some execution path the event represented by n_y may be performed immediately after the event represented by n_x (Huang *et al*., 2010). This relationship is called follows. Hence the directed edges in the EFG are represented by a set E of ordered pairs (e_x, e_y), where $\{e_x, e_y\} \subseteq N$ and $(e_x, e_y) \in E$ if e_y follows e_x.



(a) (b)

Fig. 2: A simple GUI example to represent the EFG

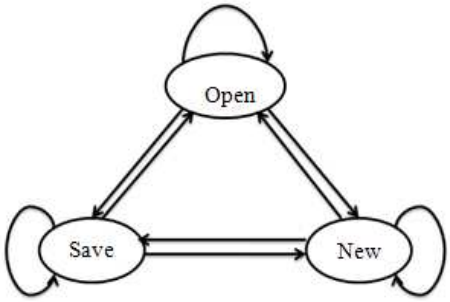


Fig. 3: The representation of EIG model

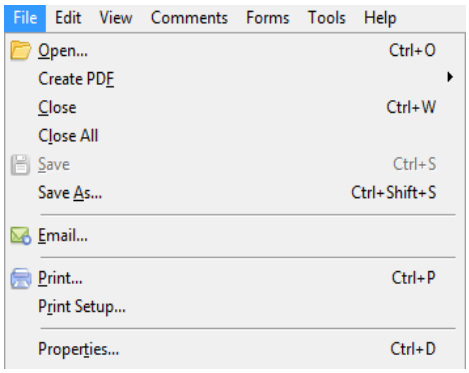


Fig. 4: Illustration of the GUI Constraints

To achieve more compact and efficient GUI model, Event-Interaction Graph (EIG) is emerged (Xie and Memon, 2008). In this model, the events to open or close menus, or open windows are not considered. Hence, menu-opening "File" event for the GUI in Fig. 2a is neglected. To generate the EIG model from the EFG model in Fig. 2b it is required first to delete the "File" event because it is a menu-open event, then for all remaining events, the even after "File" is replaced by the "File" in the edge. In other words, each edge (e_x,File) is replaced with the edge (e_x, e_y) for each occurrence of edge (File, e_y) and for all e_y, delete all edges (File, e_y). Figure 3 represents the EIG model for the GUI in Fig. 2a.

To derive test cases for the EIG model, a common technique is to derive a test case for each EIG edge, which is called "smoke test" (Memon and Xie, 2005; Yuan *et al*., 2010). For example, (Open,New), (Open,Save) and (New,Save) represent three smoke test cases with strength 2 for the EIG in Fig. 3. When the length of the sequence grows, the numbers of smoke test cases will grow exponentially exactly like components interaction testing. Here CAs come to face to reduce these smoke test cases systematically. For example, if we have five locations in the GUI each of them has three events; we need 3^5 or 243 test cases to test the GUI exhaustively. However, using the CAs properties, we can systematically sample those events. Using 2-way interaction, for example, we can test by only 11 test cases and the notation will be CA(11;2,5,3).

To derive the events in any GUI, normally GUI Ripper is used. The use of GUI ripper helps to traverse a GUI under test automatically and extracts the events (Huang *et al*., 2010). The problem with this ripping process is that some parts of the retrieved information may be incomplete or incorrect, which leads to introduce some constraints (Yuan *et al*., 2010). For example, some event needs another event to be executed before it is enabled or sometimes two events cannot be executed consecutively. As in the case of Fig. 4, we cannot run the "Save" event without running the "Save As…" event first.

Most of the research efforts concentrate on the generation of the test cases and how to solve the constraint problem. Yuan and Memon (2007) developed a new feedback-based technique for GUI testing. The technique depends on creating and executing an initial seed test suite for the software under test. The EIG model of the GUI is used to generate the seed test suite and then automatic test case re-player is used to execute it. A feedback is used to supplement the test suite when it is executing, by generating additional test cases. The relationship between pairs of events is identified to capture how the events are related to each other (Yuan and Memon, 2007). The empirical study reveals to the fact that although using this method, there are still infeasible test cases cannot be run in the test suite.

In another research Memon tried to solve this problem by repairing the unusable test cases (Memon, 2008). The study based on determining the usable and unusable test cases automatically from the test suite then determine the unusable test cases that can be repaired so that they can be executed. The repairing transformations are used to repair the test cases. Although useful and effective, from the results, it has been appeared that there are still many kinds of constraints that should be solved and dealt with.

Huang *et al*. (2010) developed a method to automatically repair GUI test suites and generating new test cases that are feasible. The genetic algorithm is used in the research to evolve new test cases to increase the test suite's coverage. The algorithm produces effective results for different types of constraints. The research showed that the genetic algorithm outperforms the random algorithm and trying to achieve the same goal in almost all cases.

More recently Yuan *et al*. (2010) used the aforementioned researches to define new criteria for GUI testing grounded in CAs in more detail. The research incorporated "context" into the criteria in terms of event combinations, sequence length and by including all possible positions for each event. The criteria are based on both the efficiency (measured by the size of the test suite) and the effectiveness (the ability of the test suites to detect faults). The study conducts more empirical studies than before using eight applications. The results of those studies showed that increasing the interaction strength t and by controlling the relative positions of events, large number of faults can be detected compared with earlier techniques.

Test case prioritization and regression testing: Regression testing is a type of software testing aims to find uncovered new errors after changes have been made to the software (Gu *et al*., 2010). This testing process is based on the fact that as the software is upgraded or developed, the occurrence of similar faults is frequent. This in turn leads to keep those test cases that detected faults in earlier version of the software to re-executing them after developing a new version of the same software. Using this method for testing helps to verify that the changes of specific software have not caused the software inadvertent side-effects and the software still meets its requirements (Rothermel and Harrold, 1996).

Test case prioritization techniques, in another hand, aimed to increase the effectiveness of test cases by scheduling them for execution to increase the rate of fault detection (Rothermel *et al*., 2001). This in turn gives the estimation of how quickly faults are detected in the testing process. Detecting the faults earlier during the testing process provide faster feedback for the tester and let him to begin correcting defects earlier than might otherwise be possible (Rothermel *et al*., 2001).

Most of the time, prioritization techniques are associated with regression testing as the information from previous execution of test cases can be used for the ordering and sequencing processes (Rothermel *et al*., 2001). Both of regression and prioritization are dependent on the test suite selected for the process, especially the initial test suite (Qu *et al*., 2007). CAs have been used with this kind of testing effectively because of optimized size. By using the generated CAs as test suites, first, an extensive prioritization has been applied to the test suite and then it is applied for the regression testing.

There are few researches introduced the prioritization with CAs. Bryce and Colbourn (2006) present an algorithm for prioritizing the test suites that based on CAs. Although the research showed impressive results for prioritization using interaction coverage; however, the algorithm is not applied on real software to illustrate its effectiveness. Qu *et al*. (2007) used multiple versions of two software subjects to examine the application and effectiveness of CAs for finding faults with regression testing. Before applying the test suites on the two subject software, the suites have been prioritized. Several different algorithms and methods used to control the prioritization. During the empirical study, the interactions used are between t =2 and 5. The results of the empirical study showed that the CAs are effective to reduce the test space and find the faults. Using prioritization with CAs improves the ability to detect faults early in certain subjects. The results showed also that for the first subject software, most of the faults are covered by t=3; however, for the second subject software, most of the faults covered when t=5.

Later on, Qu *et al*. (2008) applied the prioritization in regression testing with additional subjects. The research study several versions of an open source text editor. The results showed that using prioritization with CAs can impact the fault finding ability of regression test suites by as much as 70%.

Fault characterization: Fault characterization (sometimes called failure diagnosis), is a mechanism or method used to find and locate faults in given software. With the increase of software complexity, some kinds of faults appeared that could not diagnose by the traditional methods. This is happened frequently when the system configuration spaces are large and resources are limited. For example, some systems like "Apache", can have hundreds of options that could not be tested extensively because of the large software configuration space and thus leads to the inability of characterizing some faults.

Fault characterization helps to identify the cause of a specific fault and save a great time by fixing the fault quickly. In other words, fault characterization process helps to determine which specific configuration or setting of the system causing a specific failure (Yilmaz *et al*., 2004a).

Yilmaz *et al*. (2004b) developed a distributed continuous quality assurance process framework called "Skoll" (Memon *et al*., 2004). The process is supported by tools to leverage the widespread computing resources of worldwide users automatically. The aim was to incrementally, opportunistically and efficiently improve the quality of software. Skoll divided the quality assurance process into sub-tasks, then the tasks are distributed around the world to different client machines using the client-server communication. Each client downloads the software under test from a central code repository and uses a given configuration to test. To complete the overall quality process, the results are returned back to a central site that collects results and fused together (Yilmaz *et al*., 2004a).

One important implemented task in skoll is the fault characterization. This characterization process of faults is done by testing different configurations and features of the software under test and feed the result of the testing to a classification tree analysis. The output of this classification tree analysis would be a model to describe the options and setting that best forecast failure (Yilmaz *et al*., 2004b). Here, CAs used for generating the configurations' models for skoll. In this way, all the combinations of the options and are appearing in the provided configurations and will greatly reduce the cost of fault characterization, without compromising its accuracy.

Yilmaz *et al*. (2004a) evaluated the uniform CA with interaction strength between 2 and 6. The results showed that even low strength CA can achieve reliable fault characterization compared with those achieved by exhaustive testing. By increasing the interaction strength of CA, the research reported more precise fault characterization, but with more test suite sizes. The research concludes that the fault characterization could be improved in term of accuracy with low cost if the low strength CA be used.

Yilmaz *et al*. (2006) extend the above research to include more empirical studies. For the first time, the research reports the application of VSCA to tests the effects of using it practically. The use of VSCA allows testing stronger interactions for subspaces where they are needed (i.e., in high-risk subspaces) and keeping a lower strength of interaction across the entire space (Yilmaz *et al*., 2006). The research reports the same finding for the CA as in the case of the original research. In addition, the research reported that the use of VSCA reduces the cost of the fault characterization process without compromising its accuracy. Moreover, the research showed that use of VSCA improved that accuracy of the fault characterization process with the same cost of CA.

Future research directions: There are many researches on CAs. Most of the researches are concentrating on the CAs construction methods, algorithms, or strategies. The researches for investigating the applications of CAs in software testing are still in the beginning. As we can see from the aforementioned applications, there are few researches dealing with the application of CAs. Although all the areas need further investigations and assessments, there are new directions that can be impressive for applications. Based on that, our recommendations to the future research focus could be in following two general directions mainly:

- Further assessments to the existing researches: although the exiting researches are achieving good results, different areas need more assessments and improvements. One of those areas is the components' interaction testing. So far, the effectiveness of using CAs in this area is not clear. Although it is studied theoretically in many researches, there is little evidence showing its effectiveness. It seems to be encouraging area for empirical studies. An interesting direction, for example, is the use of CAs in e-commerce software systems. It seems to be interesting if a research study the effect of the component interactions on some performance criteria practically. In addition, the application of VSCA is an open research direction also. We can note from the study, the application of VSCA is not investigated clearly. So far, from the literature, there is only one research apply and investigate the effectiveness of the VSCA practically. This could be also applied with e-commence systems by taking stronger interaction strength among some special related components in the system, for example.
- Discovering new directions of software testing applications: CAs could be applied in a different way for software testing. There is a need to study and investigate new software testing directions using CAs. One important direction is to combine the CAs features with other software testing methods. As we mentioned previously, CAs have been used with regression testing and test case prioritization. It also could be useful if the CAs features are used with fault localization techniques. Recently, fault localization has become an active research area. So far, the use of CAs not investigated with fault localization. Wong *et al*. (2010) investigate a new fault localization method using code coverage heuristic. In other recently researches, interaction testing using CAs have been

effective for improving code coverage using some empirical studies (Zamli *et al*., 2011; Klaib *et al*., 2008). This in turn could be an important motivation for using CAs with code coverage heuristics to improve the fault localization as conducted by Wong *et al*. (2010).

CONCLUSION

The applications of CAs and interaction testing have been an active research area recently. In this study, we aimed to demonstrate the CAs and report their existing applications to software testing. To understand the CAs use in the applications, we first illustrate their types, notations and construction methods. Then, we reviewed several recent applications of CAs to software testing. We briefly mention some of those applications and the achieved results to illustrate the effectiveness of CAs in those applications. The research in this area seems to be an active research direction for the coming years. To this end, in this study we also give different research directions for the future and we also suggest some important ideas for the coming researches.

ACKNOWLEDGEMENT

This research is partially funded by the generous grant ("Investigating T-Way Test Data Reduction Strategy Using Particle Swarm Optimization Technique") from the Ministry of Higher Education (MOHE) and the USM research university grants ("Development of Variable-Strength Interaction Testing Strategy for T-Way Test Data Generation"). The first author, Bestoun S. Ahmed, is a recipient of the USM fellowship.

REFERENCES

Afzal, W., R. Torkar and R. Feldt, 2009. A systematic review of search-based testing for non-functional system properties. Inf. Software Technol., 51: 957-976. DOI: 10.1016/J.INFSOF.2008.12.005

Agarwal, B.B., S.P. Tayal and M. Gupta, 2010. Software Engineering and Testing: An Introduction.1st Edn., Jones and Bartlett Learning, Hingham, ISBN: 1934015555, pp: 515.

Ahmed, B.S. and K.Z. Zamli, 2010. PSTG: A t-way strategy adopting particle swarm optimization. Proceeding of 4th Asia International Conference on Mathematical/Analytical Modelling and Computer Simulation, May, 26-28, IEEE Xplore Press, Kota Kinabalu, Malaysia, pp: 1-5. DOI: 10.1109/AMS.2010.14

Baresi, L. and M. Pezze, 2006. An introduction to software testing. Elect. Notes Theo. Comput. Sci., 148: 89-111. DOI: 10.1016/J.ENTCS.2005.12.014

Beizer, B., 1990. Software Testing Techniques. 2nd Edn., Van Nostrand Reinhold, New York, ISBN: 0442206720, pp: 550.

Borodai, S.Y. and I.S. Grunskii, 1992. Recursive generation of locally complete tests. Cybernet. Syst. Anal., 28: 504-508. DOI: 10.1007/BF01124983

Bryce, R.C. and C.J. Colbourn, 2006. Prioritized interaction testing for pair-wise coverage with seeding and constraints. Inform. Software Technol., 48: 960-970. DOI: 10.1016/J.INFSOF.2006.03.004

Bryce, R.C. and C.J. Colbourn, 2007. The density algorithm for pairwise interaction testing. Software Test. Verificat. Reliabil., 17: 159-182. DOI: 10.1002/STVR.365

Bryce, R.C. and C.J. Colbourn, 2009. A density-based greedy algorithm for higher strength covering arrays. Software Test. Verificat. Reliabil., 19: 37-53. DOI: 10.1002/STVR.393

Calvagna, A. and A. Gargantini, 2009. IPO-s: Incremental generation of combinatorial interaction test data based on symmetries of covering arrays. Proceedings of the IEEE International Conference on Software Testing, Verification and Validation Workshops, April. 1-4, IEEE Xplore Press, Denver, CO, pp: 10-18. DOI: 10.1109/ICSTW.2009.7

Cawse, J.N., 2003. Experimental Design for Combinatorial and High Throughput Materials Development. 1st Edn., Wiley-Interscience, Hoboken, ISBN: 0471203432, pp: 317.

Chen, X., Q. Gu, A. Li and D. Chen, 2009. Variable strength interaction testing with an ant colony system approach. Proceedings of the Asia-Pacific Software Engineering Conference. Dec. 1-3, IEEE Xplore Press, Penang, pp: 160-167. DOI: 10.1109/APSEC.2009.18

Cheng, C.S., 1980. Orthogonal arrays with variable numbers of symbols. Ann. Statist., 8: 447-453. DOI: 10.1214/AOS/1176344964

Cohen, D.M., S.R. Dalal, M.L. Fredman and G.C. Patton, 1997. The AETG system: an approach to testing based on combinatorial design. IEEE Trans. Software Eng., 23: 437-444. DOI: 10.1109/32.605761

Cohen, M.B., 2004. Designing Test Suites for Software Interaction Testing. 1st Edn., Computer Science-University of Auckland, New Zealand, pp: 171.

Cohen, M.B., M.B. Dwyer and J. Shi, 2007. Interaction testing of highly-configurable systems in the presence of constraints. Proceedings of the 2007 International Symposium on Software Testing and Analysis, (ISSTA '07), ACM, New York, pp: 129-139. DOI: 10.1145/1273463.1273482

Colbourn, C.J., 2008. Strength two covering arrays: Existence tables and projection. Discrete Math., 308: 772-786. DOI: 10.1016/J.DISC.2007.07.050

Colbourn, C.J., S.S. Martirosyan, G.L. Mullen, D. Shasha and G.B. Sherwood *et al*., 2006. Products of mixed covering arrays of strength two. J. Combinat. Designs, 14: 124-138. DOI: 10.1002/JCD.20065

Czerwonka, J., 2006. Pairwise testing in real world practical extensions to test case generator. Microsoft Research.

Grindal, M., J. Offutt and S.F. Andler, 2005. Combination testing strategies: A survey. Software Test. Verificat. Reliabil., 15: 167-199. DOI: 10.1002/STVR.319

Gu, Q., B. Tang and D. Chen, 2010. Optimal regression testing based on selective coverage of test requirements. Proceedings of the International Symposium on Parallel and Distributed Processing with Applications, Sep. 6-9, IEEE Xplore Press, Taipei, pp: 419-426. DOI: 10.1109/ISPA.2010.62

Hoskins, D.S., C.J. Colbourn and D.C. Montgomery, 2005. Software performance testing using covering arrays: Efficient screening designs with categorical factors. Proceedings of the 5th International Workshop on Software and Performance, (WOSP '05), ACM, New York, pp: 131-136 DOI: 10.1145/1071021.1071034.

Huang, S., M.B. Cohen and A.M. Memon, 2010. Repairing GUI test suites using a genetic algorithm. Proceedings of the 3rd IEEE International Conference on Software Testing, Verification and Validation, April, 6-10, IEEE Xplore Press, Paris, 245-254. DOI: 10.1109/ICST.2010.39

Jenkins, B., 2010. Jenny Download Web Page. http://burtleburtle.net/bob/math/jenny.html

Klaib, M.F.J., K.Z. Zamli, N.A.M. Isa, M.I. Younis and R. Abdullah, 2008. G2Way a backtracking strategy for pairwise test data generation. Proceedings of the 2008 15th Asia-Pacific Software Engineering Conference, Dec. 3-5, IEEE Xplore Press, Beijing, pp: 463-470.DOI: 10.1109/APSEC.2008.49

Klaib, M.F.J., S. Muthuraman, N. Ahmad and R. Sidek, 2010. Tree based test case generation and cost calculation strategy for uniform parametric pairwise testing. J. Comput. Sci. 6: 542-547. DOI: 10.3844/JCSSP.2010.542.547

Lehmann, E. and J. Wegener, 2000. Test case design by means of the CTE XL. Proceeding of the 8th European International Conference on Software Testing, Analysis and Review, (STAR'00), Citeseer, Kopenhagen Denmark, pp:

Lei, Y. and K.C. Tai, 1998. In-Parameter-Order: A Test Generation Strategy for Pairwise Testing. Proceedings of the 3rd IEEE International Symposium on High-Assurance Systems Engineering, Nov. 13-14, IEEE Xplore Press, Washington, DC, USA, pp: 254-261. DOI: 10.1109/HASE.1998.731623

Lei, Y., R. Kacker, D.R. Kuhn, V. Okun and J. Lawrence, 2007. IPOG: A general strategy for t-way software testing. Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, March, 26-29, IEEE Xplore Press, Tucson, AZ, pp. 549-556. DOI: 10.1109/ECBS.2007.47

Lei, Y., R. Kacker, D.R. Kuhn, V. Okun and J. Lawrence, 2008. IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing. Software Test. Verificat. Reliabil., 18: 125-148. DOI: 10.1002/STVR.381

Li, P., T. Huynh, M. Reformat and J. Miller, 2007. A practical approach to testing GUI systems. Emp. Software Eng., 12: 331-357. DOI: 10.1007/S10664-006-9031-3

Memon, A., A. Porter, C. Yilmaz, A. Nagarajan and D. Schmidt *et al*., 2004. Skoll: Distributed continuous quality assurance. Proceedings of the 26th International Conference on Software Engineering, IEEE Computer Society, Washington, DC., pp: 459-468.

Memon, A.M. and Q. Xie, 2005. Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software. IEEE Trans. Software Eng., 31: 884-896. DOI: 10.1109/TSE.2005.117

Memon, A.M., 2002. GUI testing: pitfalls and process. Computer, 35: 87-88. DOI: 10.1109/MC.2002.1023795

Memon, A.M., 2008. Automatically repairing event sequence-based GUI test suites for regression testing. ACM Trans. Software Eng. Methodol. DOI: 10.1145/1416563.1416564

Nurmela, K.J., 2004. Upper bounds for covering arrays by tabu search. Discrete Applied Math., 138: 143-152. DOI: 10.1016/S0166-218x(03)00291-9

Qu, X., M.B. Cohen and G. Rothermel, 2008. Configuration-aware regression testing: an empirical study of sampling and prioritization. Proceedings of the 2008 International Symposium on Software Testing and Analysis, ACM, New York, pp: 75-86. DOI: 10.1145/1390630.1390641

Qu, X., M.B. Cohen and K.M. Woolf, 2007. Combinatorial interaction regression testing: a study of test case generation and prioritization. Proceedings of the IEEE International Conference on Software Maintenance, Oct. 2-5, IEEE Xplore Press, Paris, pp: 255-264. DOI: 10.1109/ICSM.2007.4362638

Robinson, B. and L. White, 2008. Testing of user-configurable software systems using firewalls. Proceedings of the 19th International Symposium on Software Reliability Engineering, Nov. 10-14, IEEE Xplore Press, Seattle, WA, pp: 177-186. DOI: 10.1109/ISSRE.2008.46

Ronneseth, A.H. and C.J. Colbourn, 2009. Merging covering arrays and compressing multiple sequence alignments. Discrete Applied Math., 157: 2177-2190. DOI: 10.1016/J.DAM.2007.09.024

Rothermel, G. and M.J. Harrold, 1996. Analyzing regression test selection techniques. IEEE Trans. Software Eng., 22: 529-551. DOI: 10.1109/32.536955

Rothermel, G., R.H. Untch and C. Chu, 2001. Prioritizing test cases for regression testing. IEEE Trans. Software Eng., 27: 929-948. DOI: 10.1109/32.962562

Shasha, D.E., A.Y. Kouranov, L.V. Lejay, M.F. Chou and G.M. Coruzzi, 2001. Using combinatorial design to study regulation by multiple input signals. a tool for parsimony in the post-genomics era. Plant Physiol., 127: 1590-1594. DOI: 10.1104/PP.010683 PMid: 11743103 PMCid:1540192

Shiba, T., T. Tsuchiya and T. Kikuno, 2004. Using artificial life techniques to generate test cases for combinatorial testing. Proceedings of the 28th Annual International Computer Software and Applications Conference, Sep. 28-30, IEEE Xplore Press, Hong Kong, pp: 72-77. DOI: 10.1109/CMPSAC.2004.1342808

Stardom, J., 2001. Metaheuristics and the Search for Covering and Packing Array Department of Mathematics. M.Sc, Thesis, Simon Fraser University.

Tung, Y.W. and W.S. Aldiwan, 2000. Automating test case generation for the new generation mission software system. Proceedings of the IEEE Aerospace Conference, Mar. 18-25, IEEE Xplore Press, Big Sky, MT , USA, pp: 431-437. DOI: 10.1109/AERO.2000.879426

Wang, Z., B. Xu and C. Nie, 2008. Greedy heuristic algorithms to generate variable strength combinatorial test suite. Proceedings of The Eighth International Conference on Quality Software, Aug. 13-13, IEEE Xplore Press, Oxford, pp: 155-160. DOI: 10.1109/QSIC.2008.52

Williams, A.W. and R.L. Probert, 1996. A practical strategy for testing pair-wise coverage of network interfaces. Proceedings of the the 7th International Symposium on Software Reliability Engineering, Oct. 30-Nov, 2, IEEE Xplore Press, White Plains, NY, USA, pp: 246-254. DOI: 10.1109/ISSRE.1996.558835

Williams, A.W. and R.L. Probert, 2001. A measure for component interaction test coverage. Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications, June, 25-29, IEEE Xplore Press, Beirut , Lebanon, pp: 304-311. DOI: 10.1109/AICCSA.2001.934001

Williams, A.W. and R.L. Probert, 2002. Formulation of the interaction test coverage problem as an integer program. Proceedings of the IFIP 14th International Conference on Testing Communicating Systems XIV, Kluwer, B.V. Deventer, The Netherlands, The Netherlands, pp: 283-283. http://portal.acm.org/citation.cfm?id=748164

Wong, W.E., V. Debroy and B. Choi, 2010. A family of code coverage-based heuristics for effective fault localization. J. Syst. Software, 83: 188-208. DOI: 10.1016/J.JSS.2009.09.037

Xie, Q. and A.M. Memon, 2008. Using a pilot study to derive a GUI model for automated testing. ACM Trans. Software Eng. Methodol., 18: 1-35. DOI: 10.1145/1416563.1416567

Yilmaz, C., M.B. Cohen and A. Porter, 2004a. Covering arrays for efficient fault characterization in complex configuration spaces. ACM SIGSOFT Software Eng. Notes, 29: 45-54. DOI: 10.1145/1013886.1007519

Yilmaz, C., M.B. Cohen and A.A. Porter, 2004b. Covering arrays for efficient fault characterization in complex configuration spaces. Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis, ACM, New York, pp: 45-54. DOI: 10.1145/1007512.1007519

Yilmaz, C., M.B. Cohen and A.A. Porter, 2006. Covering arrays for efficient fault characterization in complex configuration spaces. IEEE Trans. Software Eng., 32: 20-34. DOI: 10.1109/TSE.2006.8

Yu, Y.T., S.P. Ng and E.Y.K. Chan, 2003. Generating, selecting and prioritizing test cases from specifications with tool support. Proceedings of the 3rd International Conference on Quality Software. Nov. 6-7, IEEE Xplore Press, USA., pp: 83-90. DOI: 10.1109/QSIC.2003.1319089

Yuan, X. and A.M. Memon, 2007. Using GUI run-time state as feedback to generate test cases. Proceedings of the 29th International Conference on Software Engineering, May, 20-26, IEEE Xplore Press, Minneapolis, MN, pp: 396-405. DOI: 10.1109/ICSE.2007.94

Yuan, X., M. Cohen and A. Memon, 2010. GUI interaction testing: incorporating event context. IEEE Trans. Software Eng., pp: 1-1.DOI: 10.1109/TSE.2010.50

Zamli, K.Z., Isa, N.A.M., 2008. JTst - An automated unit testing tool for java program. Am. J. Applied Sci. 5, 77-82. DOI: 10.3844/AJASSP.2008.77.82

Zamli, K.Z., M.F.J. Klaib, M.I. Younis, N.A.M. Isa and R. Abdullah, 2011. Design and implementation of a t-way test data generation strategy with automated execution tool support. Inform. Sci. 181: 1741-1758 DOI: 10.1016/J.INS.2011.01.002