Combinatorial testing in software projects

Mario Brčić and Damir Kalpić*
* Faculty of Electrical Engineering and Computing, Zagreb, Croatia
mario.brcic@fer.hr, damir.kalpic@fer.hr

Abstract- Software systems continuously grow in size and code complexity, the latter most evident through greater component interconnectedness. This leaves more space for bugs which introduce risks such as exposure to security threats. Combinatorial testing looks for interaction failures in order to improve the system security and effectiveness guarantees. One of the most effective test selection approaches under combinatorial testing are experimental design extensions for software testing. Covering array test sets are compact while maintaining at the same time complete combinatorial coverage up to the desired level. Smaller test sets with customizable level of assurance can drive testing costs down substantially. The paper presents a survey of research into combinatorial testing suite factors while also identifying possible future research ideas.

I. INTRODUCTION

Software systems continuously grow in size and code complexity. Both factors contribute to greater danger of programming mistakes that lead to unexpected results. Such mistakes introduce various forms of risk that include security risk due to exploits, irreversible impacts in safety-critical systems, negative perception of product and company, etc. NIST report [1] estimates the cost of inadequate software testing to US economy to be in the range 22.2-59.5 billion USD per year despite the allocation of substantial project resources to the testing.

All said underlines the importance of adequate testing, which tries to find faults and identify their causes with minimal allocation of budget and time. Combinatorial testing (CT) selects test cases sampling out different combinations of parameter values. One of the most effective test set creation approaches under CT are experimental design extensions for software testing and such will be the focus of this work. Combinatorial testing methods produce compact test sets with desired characteristics that, if done properly, can greatly reduce testing costs, while simultaneously guaranteeing the required level of product faultlessness. In short, we can balance cost and risk by selecting a covered interaction level. The illustration of compactness of the generated covering array test sets for a system with 100 binary input variables is given in Tab.1 where the test set sizes were derived using the data from [2].

Exhaustive testing is infeasible, but [3] and [4] empirically demonstrated that all the known failures in a variety of domains were triggered by interactions of 6 or less parameters, as depicted in Fig.1. These results are very important and encouraging because the generation of test sets with this bounded interaction strength is tractable and the resultant test sets are substantially smaller than the exhaustive ones.

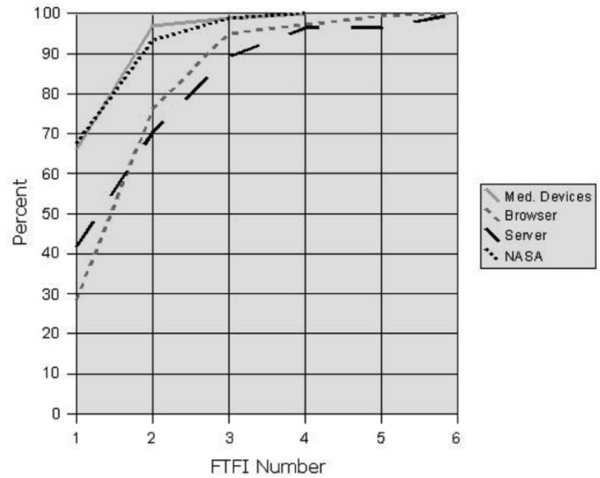

Figure 1 Cumulative distribution of triggered faults with increasing test interaction covering strength, taken from [4]

This paper is organized as follows; combinatorial testing components are covered in part 2, tools and applications are listed in part 3. Section 4 concludes the paper.

II. COMBINATORIAL TESTING

Combinatorial testing is a black-box (I/O based, functional) technique that seeks to detect faults caused by parameter interactions, hence also covering interaction between system components.

The typical measure used for comparison of combinatorial test sets is the combinatorial coverage. The t-way combinatorial coverage is the percentage of t-tuples occurring in the test set, relative to the total number of possible t-tuples. Generated combinatorial test using covering arrays cover 100% parameter interactions up to the desired interaction strength, while the higher interaction levels are only partially covered (<100%). Typical test sets under CT are: covering arrays, orthogonal, mixed level and variable strength covering arrays.

Covering array CA_{λ} (N; t, k, v), is an array with N cases and k parameters that satisfies the constraint that each t-tuple occurs at least λ times, with usual value for software testing of λ=1 if not stated otherwise. t denotes the strength of completely covered interaction level, k is the set of parameters, and v is the domain of all the parameters. Rows of an array list test cases and columns list the parameters. Lower bound for the number of tests in minimal CA is v^t, while the upper bound for $v^t \rightarrow \infty$ and $k \rightarrow \infty$ [5] is:

$$(t-1)\, v^t \log(k)\, (1+o(1)) \qquad (1)$$

TABLE I. TEST SET SIZES FOR DIFFERENT COVERED INTERACTION STRENGTHS

Covered interaction strength (t)	# of test cases in test set
1	2
2	10
3	33
4	98
5	202
6	718
100 (exhaustive)	$2^{100} \approx 1.267*10^{30}$

Orthogonal array OA_λ (N; t, k, v), is a covering array with the constraint that all the t-tuples have to occur exactly λ-times each (i.e. they are balanced). Orthogonal arrays don't exist for some combinations of parameters (t,k,v) and if they do exist, they are harder to generate than covering arrays and are usually larger. The positive fact for OA is that they enable much easier fault identification. The subsection D covers the general fault identification for covering arrays. Cohen et al. in [6] conclude that the property of balance is not necessary for software testing, unlike statistical experiments and that CAs are a more efficient solution than OAs.

Mixed-level covering arrays MCA_λ (N; t, k, $(v_1,v_2,\ldots v_n)$) is a generalization of covering array that permits different domains for input parameters.

Variable strength covering array VCA_λ (N; T, k, v) is a generalization of covering array that defines in T, the matrix of covering the strength requirements, different interaction covering strengths for different groups of parameters.

All of the listed covering arrays ignore the sequence of elements, an aspect that may be important in testing event driven systems, such as GUI. *Sequence covering array* SCA_λ (N; S, t) introduces the focus on t-length sequences of elements from finite set S and every such sequence has to occur in at least λ test case of the sequence covering array. Elements of t-length sequences do not have to be necessarily adjacent in the test case. The set S represents events that are sequenced in the order of triggering in the test case. The basic variant of sequence covering arrays permits only one triggering of the same event per test case. For more details on this subject, we refer the reader to [7], [8] and [9].

A. Modelling

The elements of a test model for the combinatorial test are: parameters, values, interaction and constraints.

In order to set properly the model elements, data should be gathered from the project documentation, interviews with experts and possibly by system structural insights.

First, the parameters have to be identified. Configuration parameters contain all configurations for the system and input parameters contain all of the user interface parameters.

Values for each parameter have to be determined. If a parameter can attain only few possible values, we can use them directly in tests. On the other hand, direct use of values for discrete parameters with many possible values or continuous parameters is prohibitive as it would yield intractable the test generation and testing. In the latter case we can use equivalence partitioning, boundary value analysis, category partition and domain testing that produce only a handful of test values for each parameter [10]. In case of equivalence partitioning, some partitions may be more important so we can select more values from them for better covering. The discretization in this step offers the trade-off tuning; more values create greater test sets but possibly offer finding more faults. Structural analysis can help to identify important parameter values for testing.

Interactions should be listed in order to make the generated cover arrays more efficient. Some parameters do not interact with any other parameter; there can be different interaction clusters; we may want to cover different sets of parameters with different strengths to better treat those parameters that are expected to exhibit higher interaction. Careful interactions specification can make test sets as well as generation more efficient.

Constraints are a very important aspect of our system because they define the space of impossible parameter value combinations. The inclusion of constraint in the procedure of test case generation is important because, if ignored, many impossible test cases would be obtained, improperly executed and the test coverage would be reduced, due to valid parameter combinations covered by only that test.

The modelling phase is usually done manually, following heuristic rules covered in previous works. Grindal and Offut in [11] presented eight-step structured method for creation of input parameter models for combinatorial testing. Authors in [12] presented a method for identification of categories and choices from UML activity diagram. Example system requirements were given in [13] and combinatorial testing was applied to them in a tutorial style. The formalization, creation of validation procedures and automation of the modelling phase are the goals that are yet to be accomplished. Automation needs better extraction of pertinent data from available resources such as: project documentation, interviews with experts (possibly using Natural Language Processing), existing system specifications (UML etc.) or even partially from the implementation. The level of abstraction should be customizable. The optional automation of abstraction level selection, possibly based on smaller pilot tests and on comparison of promising options, would be very useful for large systems. This automated procedure could be coupled with automated generation of model based test oracles, also a distant research goal, to provide a complete testing support.

B. Generating test

Minimal covering array generation problem is NP-complete [14], [15]. Some of the commonly used covering arrays have pre-computed best known solutions,

[2] lists best known sizes for arrays up to covering strength t=6.

Seeding is an inclusion of fixed test case or a partial test case into the set. The test cases to be included are based on domain knowledge.

The order of test cases in the set, in which they will be executed, is also an important aspect of test set generation. We can be indifferent to the order of cases, but recently there is an incentive to optimize the order in such way that it would find faults as soon as possible. Definition of prioritization is given in [16] where it was used for regression testing, retesting the system after modifications. However, there is a hard problem of defining effective prioritization function. Bryce and Colbourn in [17] presented a greedy algorithm that produces prioritized test set with support for constraints and seeding. Bryce and Memon in [18] used prioritization of sequence covering arrays for GUI testing. The applied prioritization functions were based on the length of test cases and on the covering interaction strength.

Constraint inclusion greatly complicates the generation of test sets in combinatorial testing. There are several ways to deal with constraints: ignore them (as do most of existing algorithms), require explicit enumeration of forbidden combinations, introduce soft constraints without guarantees, or to completely address hard constraints (rarely) [19]. Even in the case of a small number of constraints, procedures that do not consider constraints may generate a substantial number of invalid test cases. Invalid test cases contain many valid combinations that can be covered only by these cases, which leads to decrease of the test coverage. Creation of new algorithms that incorporate constraints is an interesting area of research. Cohen et al. in [20] used Satisfiability Testing (SAT) to prune the search space of the greedy algorithm and Calvanga in [21] created an algorithm based on Satisfiability Modulo Theories (SMT) that enables easier expressing of complex constraints.

Methods for generating the covering arrays can be categorized into next groups: greedy constructive heuristics, metaheuristics, mathematical, hybrid methods [19] and other computational approaches. Coulbourn in [22] and Kuliamin and Petukhov in [5] have presented a survey of methods for constructing covering arrays with the focus on algebraic methods. The latter survey states time and space complexity for all of the algorithms.

Greedy constructive heuristics iteratively create test sets following the locally optimal choices at each stage. These methods can be combined with seeding as they iteratively extend the test set and some algorithms efficiently support prioritization and constraints. Greedy methods have unfavourable spatial complexity, as they have to keep the list of uncovered t-tuples in order to make their decisions. There are two families of algorithms: row generation based and parameter based methods. *Row generation methods* create test set by adding new rows one-by-one in stages to the set in such a way that they cover as many as possible of yet uncovered combinations. AETG [6], CATS [23] and density algorithm for pairwise tests [24] are members of this family. *Parameter-based methods* build test set by both column and row expansion. The test case is always a CA between the stages. In each stage a new parameter (column) is added to the test set and values for that parameter are filled in so that it covers as many as possible t-tuples. After that, new rows are added to cover the rest of t-tuples and the procedure continues until all parameters are added. These greedy algorithms in fact have better time complexity than the former. In-parameter-order (IPO) family of algorithms is using this approach, see [15], [25].

Metaheuristics take up existing test set, in most cases not necessarily CA, and try to improve it through various transformations in an effort to find the minimal covering array. These methods are generally not efficient in the presence of constraints. Many approaches were used for CA and VCA generation, here listing only some. Tabu search was applied to generation of mixed variable strength covering arrays in [26]. Authors in [27] used harmony search for generation of 2-way covering arrays and their method outperformed other popular approaches on the benchmarks. A particle swarm optimization method for generation of VCAs was presented in [28]. Authors in [29] used simulated annealing for binary valued parameters and found new best solutions for various numbers of input parameters. They have also used the result on one of their instances with recursive construction methods to produce further new best solutions for some bigger instances.

Mathematical methods use some convenient features in CA parameter configuration as basis for their functioning. Methods can be categorized as direct, reductive or recursive. Direct methods use the relation between the covering arrays and combinatorial constructs (e.g. Latin squares) for construction of simpler CAs. For a restricted class of CAs they can build minimal solutions. Recursive methods construct solution from smaller CAs or other combinatorial constructs. Reductive methods generate solution from bigger/stronger CAs using different modifications. Mathematical methods are fast and efficient, even producing minimal and near minimal solutions in restricted classes of the problem. However, in a general case they can produce worse solutions than greedy methods and these methods do not support prioritization and constraints.

Hybrid methods combine different approaches in an attempt to reap potentially synergic effects. Row generating greedy heuristics and metaheuristics were combined in [30]. Cohen et al. in [31] used mathematical methods with simulated annealing to find new lower bounds for some problems.

Other computational approaches are different from all the above listed ones and they are rarely used. Williams and Probert formulated minimal CA generation problem as {0,1} integer programming problem in [32]. Such formulations can be solved using available solvers to get exact solutions. Authors in [33] created an exact algorithm for minimal CA generation based on combination of backtracking and SAT.

Random sampling constructive method has been categorized under combinatorial testing strategies as well, by [10]. It samples test cases by some predefined distribution. Generated test sets do not have convenient properties as CA test sets but these methods are used because of their simplicity. They are also interesting as benchmarks for other methods (check subsection E).

C. The test oracle problem

A generated covering array contains only different value combinations for parameters, but in order to carry on with the test, the expected system outputs should be resolved in order to enable the assessment of faultiness. The test oracle problem is not exclusively related only to combinatorial testing. There are many possible solutions to the test oracle problem, for example: human oracles, crash test, embedded assertions, formal model based tests, etc. Human oracles are the alternative that is very common, but also the one better to avoid, due to costs and impracticality for larger systems.

Crash tests are the most trivial oracles that simply detect system crash or easily detectable system failure.

Embedded assertions are widely used method for testing. Assertions are embedded into the code stating various relations between the data. Some modern languages have full support for specification of assertions, such as Eiffel [34], and there are tools which have extend the language capabilities, such as JML (Java Modelling Language) that was used with success for testing smart cards in [35] using combinatorial testing.

Model based oracles ([36], [37]) are the most complicated but they also provide the most complete solution to the oracle problem. An additional cost for manual creation of mathematical model of the system is incurred in this approach, but the ability to create automatically large test sets is gained. The model checker with supplied system model can create expected system outputs that complete test cases. Kuhn, Kacker and Lei in [38] used NuSMV model checker with ACTS test case generator. Most of the empirical work has been done with smaller models in well-known domains. Questions about scalability and performance of this category of oracles have been raised. Model creation from the source code is an interesting research venue.

GUI test oracles based on AI planning were used in [39] with the formal model of the system containing objects and actions from which it inferred the expected internal state. This approach is limited to GUI testing.

Info Fuzzy Networks (IFN) were used as oracles in [40] but only for regression testing of unchanged components.

Artificial Neural Networks (ANN) based oracles, first proposed in [41], are the area of recent research. However, they have the problem that they need training sets with expected outputs in order to be trained. For that reason, they were mainly used for regression testing of unchanged components or simple problems with smaller I/O domains. Shahamiri in [42] used four artificial neural networks with I/O relationship analysis, which tackled the problem of expected outputs for ANN training set. The effectiveness was shown on a web-based car insurance application using mutation testing. To our best knowledge, ANN based oracles have not yet been used with combinatorial testing.

Log file analysis, proposed in [43], provides oracles based on simple state machines using collected log files, which should contain relevant data. Authors in [44] presented the test oracle framework based on the log file analysis.

D. Fault identification

After the test execution, a set of failing test cases is available, but there remains the problem of fault identification that speeds up the fault localization in the system and it is indispensible in cases when there is no source code available for debugging. In a realistic scenario, a system can have many parameters meaning that the set of potential faulty parameter combinations in the faulty case can be enormous. The failure identification aims to find, often with additional retesting, all the fault triggering combinations and it greatly helps in the debugging process, especially if fault identification and debugging were utilized in automatic fault localization in code.

Classification tree was used in [45] to identify faults from test results. This idea was used as part of procedure for adaptive test generation in [46] that enables easier identification of faults with further retesting. Shi et al. in [47] proposed novel debugging method for pair-wise testing that singles out potential fault-causing factors from test results using set analysis. Additional biased (complementary) retesting can identify failure-causing combinations. Continuing on that work, authors in [48] present a method of failure identification based on minimal failure-causing schema alongside the CT methodology. Iterative adaptive procedure was presented in [49]. Ghandehari et al. in [50] introduced ranked retesting approach to the fault identification, where the ranking was based on suspiciousness measures of environment and pertinent combinations.

E. CA vs. random testing

Combinatorial testing using covering arrays, was compared to the random testing in several papers, but conclusions between these studies were contradictory. In [51] the conclusion is that there is no significant difference in fault detection effectiveness and that test suites from both methods have a similar number of test combinations covered. The combinatorial coverage was a comparison criterion in [52]. It was shown that covering array approach produced better results for a limited number of test cases per set, t-way CAs finds more t-way faults than random testing, but that random testing, due to its unfocused nature and larger test sets finds more faults in higher interaction levels.

As the covering array generation is a NP-complete problem, random testing could be preferred in situations where there are no means to acquire/generate test cases quickly, with many parameters and/or with high interaction coverage strengths. More research is needed into this aspect of the trade-off.

III. APPLICATIONS

Combinatorial testing support tools mostly use greedy algorithms. The full list of CT tools is available at [53].

Applications of combinatorial testing were reported in a number of papers, listing here only a few. Nair et al. in [54] tested a small operation support system in AT&T. Smith et al. in [55] used pair-wise testing on remote planner agent on Deep Space 1 mission. Krishnan et al. in [56] described the testing on a mobile phone application. Lei et al. in [57] presented CT approach to concurrent programs. Authors in [58] created CT approach for testing buffer overflow vulnerabilities. Unified model for GUI and web applications testing, with an empirical study, was demonstrated in [59].

Empirical studies that quantify the effects of combinatorial testing on final project quality would be welcome.

IV. CONCLUSION

A combinatorial testing using covering arrays involves cost vs. risk trade-off that enables to completely focus the efforts to important regions of the search space. Despite inconclusive empirical studies, combinatorial testing, if done properly, can be very effective in finding interaction failures, especially in situations where the execution of each test case is expensive and/or number of test cases is limited. It is possible that some faults occur only in higher interaction levels than covered by testing. The concerns regarding such faults can be somewhat relieved by empirical findings from a study of projects in several domains that state the absence of triggered faults in interaction levels higher than 6 and by a considerable falling trend of triggered faults when increasing the interaction strength (Fig.1). Maybe some of the undetected faults could be easily found using a different approach to testing or having more insight into the structure of the implementation could single out additional interesting test cases in higher interaction levels. For the faults uncovered even by hybrid approaches, we rely on virtual intractability of testing in higher interaction levels, culminating with exhaustive testing as well as a very low probability of their accidental triggering.

Some potential research venues have been identified. Automation and optimization of test model creation and test oracle generation to the fullest possible extent as well as their verification are important steps to wider acceptance. There is a need for empirical studies of CT effectiveness compared to other methods as well as identification of environment where this approach would be the best. Test case generation methods can be improved. Most research is currently done for covered interaction strength of 2 and, to a lesser extent, 3. Better inclusion of constraints and prioritization in efficient test set generation, better covering array generation for t>2, generation of (mixed) variable strength and sequence covering arrays as well as responding to changes in requirements comprise viable research agenda.

REFERENCES

[1] G. Tassey, "The Economic Impacts of Inadequate Infrastructure for Software Testing," National Institute of Standards and Technology, RTI Project Number 7007.011, 2002.

[2] C. Colbourn, "Covering array tables for t=2,3,4,5,6." [Online]. Available:http://www.public.asu.edu/%7Eccolbou/src/tabby/catable.html. [Accessed: 07-Mar-2012].

[3] D. R. Kuhn and M. J. Reilly, "An Investigation of the Applicability of Design of Experiments to Software Testing," in *Proceedings of the 27th Annual NASA Goddard Software Engineering Workshop (SEW-27'02)*, Washington, DC, USA, 2002, p. 91–.

[4] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software fault interactions and implications for software testing," *IEEE Transactions on Software Engineering*, vol. 30, no. 6, pp. 418–421, Jun. 2004.

[5] V. Kuliamin and A. Petukhov, "A survey of methods for constructing covering arrays," *Programming and Computer Software*, vol. 37, no. 3, pp. 121–146, May 2011.

[6] D. M. Cohen, S. R. Dalal, A. Kajla, and G. C. Patton, "The Automatic Efficient Test Generator (AETG) system," in *, 5th International Symposium on Software Reliability Engineering, 1994. Proceedings*, 1994, pp. 303–309.

[7] D. R. Kuhn, J. M. Higdon, J. F. Lawrence, R. N. Kacker, and Y. Lei, "Combinatorial Methods for Event Sequence Testing," 2012.

[8] D. R. Kuhn, R. Kacker, and Y. Lei, "Practical Combinatorial Testing," National Institute of Standards and Technology, Washington, DC, USA, NIST Special Publication 800-142, Oct. 2010.

[9] K. Z. Zamli, R. R. Othman, and M. H. M. Zabil, "On sequence based interaction testing," 2011, pp. 662–667.

[10] M. Grindal, J. Offutt, and S. F. Andler, "Combination testing strategies: a survey," *Software Testing, Verification and Reliability*, vol. 15, no. 3, pp. 167–199, Sep. 2005.

[11] M. Grindal and J. Offutt, "Input parameter modeling for combination strategies," in *Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering*, Anaheim, CA, USA, 2007, pp. 255–260.

[12] T. Y. Chen, Sau-Fun Tang, Pak-Lok Poon, and T. H. Tse, "Identification of Categories and Choices in Activity Diagrams," presented at the Fifth International Conference on Quality Software, 2005. (QSIC 2005), 2005, pp. 55–63.

[13] C. Lott, A. Jain, and S. Dalal, "Modeling requirements for combinatorial software testing," in *Proceedings of the 1st international workshop on Advances in model-based testing*, New York, NY, USA, 2005, pp. 1–7.

[14] G. Seroussi and N. H. Bshouty, "Vector sets for exhaustive testing of logic circuits," *IEEE Transactions on Information Theory*, vol. 34, no. 3, pp. 513–522, May 1988.

[15] Y. Lei and K.-C. Tai, "In-Parameter-Order: A Test Generation Strategy for Pairwise Testing," in *The 3rd IEEE International Symposium on High-Assurance Systems Engineering*, Washington, DC, USA, 1998, pp. 254–261.

[16] G. Rothermel, R. H. Untch, Chengyun Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, Oct. 2001.

[17] R. C. Bryce and C. J. Colbourn, "Prioritized interaction testing for pair-wise coverage with seeding and constraints," *Information and Software Technology*, vol. 48, no. 10, pp. 960–970, Oct. 2006.

[18] R. C. Bryce and A. M. Memon, "Test suite prioritization by interaction coverage," in *Workshop on Domain specific approaches to software test automation: in conjunction with the 6th ESEC/FSE joint meeting*, New York, NY, USA, 2007, pp. 1–7.

[19] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Computing Surveys*, vol. 43, no. 2, pp. 11:1–11:29, Jan. 2011.

[20] M. B. Cohen, M. B. Dwyer, and Jiangfan Shi, "Constructing Interaction Test Suites for Highly-Configurable Systems in the Presence of Constraints: A Greedy Approach," *IEEE Transactions on Software Engineering*, vol. 34, no. 5, pp. 633–650, Sep. 2008.

[21] A. Calvagna and A. Gargantini, “Combining Satisfiability Solving and Heuristics to Constrained Combinatorial Interaction Testing”, Springer Berlin / Heidelberg, 2009, pp. 27–42.
[22] C. Colbourn, “Combinatorial aspects of covering arrays,” *Le Matematiche*, vol. 58, pp. 121–167, 2004.
[23] G. Sherwood, “Effective Testing of Factor Combinations,” presented at the Third International Conference on Software Testing, Analysis & Review, Washington, DC, USA, 1994.
[24] C. Colbourn and M. Cohen, “A Deterministic Density Algorithm for Pairwise Interaction Coverage,” in *Proc. of the IASTED Intl. Conference on Software Engineering*, 2004, pp. 242–252.
[25] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, “IPOG-IPOG-D: efficient test generation for multi-way combinatorial testing,” *Softw. Test. Verif. Reliab.*, vol. 18, no. 3, pp. 125–148, Sep. 2008.
[26] L. Gonzalez-Hernandez, N. Rangel-Valdez, and J. Torres-Jimenez, “Construction of Mixed Covering Arrays of Variable Strength Using a Tabu Search Approach,” in *Combinatorial Optimization and Applications*, vol. 6508, Springer Berlin / Heidelberg, 2010, pp. 51–64.
[27] Abdul Rahman A. Alsewari, “A harmony search based pairwise sampling strategy for combinatorial testing,” *International Journal of the Physical Sciences*, vol. 7, no. 7, Feb. 2012.
[28] B. S. Ahmed and K. Z. Zamli, “A variable strength interaction test suites generation strategy using Particle Swarm Optimization,” *J. Syst. Softw.*, vol. 84, no. 12, pp. 2171–2185, Dec. 2011.
[29] J. Torres-Jimenez and E. Rodriguez-Tello, “New bounds for binary covering arrays using simulated annealing,” *Information Sciences*, vol. 185, no. 1, pp. 137–152, Feb. 2012.
[30] Xiang Chen, Qing Gu, Xinping Wang, Ang Li, and Daoxu Chen, “A Hybrid Approach to Build Prioritized Pairwise Interaction Test Suites,” in *International Conference on Computational Intelligence and Software Engineering, 2009. CiSE 2009*, 2009, pp. 1–4.
[31] M. B. Cohen, C. J. Colbourn, and A. C. H. Ling, “Augmenting Simulated Annealing to Build Interaction Test Suites,” in *Proceedings of the 14th International Symposium on Software Reliability Engineering*, Washington, DC, USA, 2003, p. 394–.
[32] A. W. Williams and R. L. Probert, “Formulation of the Interaction Test Coverage Problem as an Integer Program,” in *Proceedings of the IFIP 14th International Conference on Testing Communicating Systems XIV*, Deventer, The Netherlands, The Netherlands, 2002, p. 283–.
[33] J. Yan and J. Zhang, “Backtracking Algorithms and Search Heuristics to Generate Test Suites for Combinatorial Testing,” in *Computer Software and Applications Conference, Annual International*, Los Alamitos, CA, USA, 2006, vol. 1, pp. 385–394.
[34] B. Meyer, *Object-Oriented Software Construction*, 2nd ed. Prentice Hall, 2000.
[35] L. du Bousquet, Y. Ledru, O. Maury, C. Oriat, and J.-L. Lanet, “A Case Study in JML-Based Software Validation,” in *Proceedings of the 19th IEEE international conference on Automated software engineering*, Washington, DC, USA, 2004, pp. 294–297.
[36] P. Ammann and P. E. Black, “Abstracting formal specifications to generate software tests via model checking,” presented at the Digital Avionics Systems Conference, 1999. Proceedings. 18th, St Louis, MO, 1999, vol. B.6-6 vol.2, p. 10.A.6–1–10.A.6–10.
[37] V. Okun, P. E. Black, and Y. Yesha, “Testing with Model Checker: Insuring Fault Visibility,” *WSEAS TRANS. SYS*, vol. 2, p. 77–82, 2003.
[38] D. R. Kuhn, R. N. Kacker, and Y. Lei, “Automated Combinatorial Test Methods: Beyond Pairwise Testing,” *Crosstalk, Journal of Defense Software Engineering*, vol. 21, no. 6, pp. 22–26, Jun. 2008.
[39] A. M. Memon, M. E. Pollack, and M. L. Soffa, “Automated test oracles for GUIs,” *SIGSOFT Softw. Eng. Notes*, vol. 25, no. 6, pp. 30–39, Nov. 2000.
[40] M. Last, M. Friedman, and A. Kandel, “The data mining approach to automated software testing,” in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA, 2003, pp. 388–396.
[41] M. Vanmali, M. Last, and A. Kandel, “Using a neural network in the software testing process,” *International Journal of Intelligent Systems*, vol. 17, no. 1, pp. 45–62, 2002.
[42] S. R. Shahamiri, W. M. N. W. Kadir, S. Ibrahim, and S. Z. M. Hashim, “An automated framework for software test oracle,” *Information and Software Technology*, vol. 53, no. 7, pp. 774–788, Jul. 2011.
[43] J. H. Andrews, “Testing using log file analysis: tools, methods, and issues,” in *13th IEEE International Conference on Automated Software Engineering, 1998. Proceedings*, 1998, pp. 157–166.
[44] Dan Tu, Rong Chen, Zhenjun Du, and Yaqing Liu, “A Method of Log File Analysis for Test Oracle,” in *International Conference on Scalable Computing and Communications; Eighth International Conference on Embedded Computing, 2009. SCALCOM-EMBEDDEDCOM’09*, 2009, pp. 351–354.
[45] C. Yilmaz, M. B. Cohen, and A. Porter, “Covering arrays for efficient fault characterization in complex configuration spaces,” in *Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, New York, NY, USA, 2004, pp. 45–54.
[46] E. Dumlu, C. Yilmaz, M. B. Cohen, and A. Porter, “Feedback driven adaptive combinatorial testing,” in *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, New York, NY, USA, 2011, pp. 243–253.
[47] L. Shi, C. Nie, and B. Xu, “A software debugging method based on pairwise testing,” in *Proceedings of the 5th international conference on Computational Science - Volume Part III*, Berlin, Heidelberg, 2005, pp. 1088–1091.
[48] C. Nie and H. Leung, “The Minimal Failure-Causing Schema of Combinatorial Testing,” *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 4, pp. 15:1–15:38, Sep. 2011.
[49] Ziyuan Wang, Baowen Xu, Lin Chen, and Lei Xu, “Adaptive Interaction Fault Location Based on Combinatorial Testing,” in *2010 10th International Conference on Quality Software (QSIC)*, 2010, pp. 495–502.
[50] L. S. G. Ghandehari, Y. Lei, T. Xie, D. R. Kuhn, and R. Kacker, “Identifying Failure-Inducing Combinations in a Combinatorial Test Set,” Arlington, USA, 2012.
[51] P. J. Schroeder, P. Bolaki, and V. Gopu, “Comparing the fault detection effectiveness of n-way and random test suites,” in *2004 International Symposium on Empirical Software Engineering, 2004. ISESE ’04. Proceedings*, 2004, pp. 49– 59.
[52] D. R. Kuhn, R. N. Kacker, and Y. Lei, “Random vs. Combinatorial Methods for Discrete Event Simulation of a Grid Computer Network,” presented at the Mod Sim World 2009, Virginia, USA.
[53] J. Czerwonka, “Pairwise Testing - Available Tools.” [Online]. Available: http://www.pairwise.org/tools.asp. [Accessed: 13-Mar-2012].
[54] V. N. Nair, D. A. James, W. K. Erlich, and J. Zevallos, “A Statistical Assessment of Some Software Testing Strategies and Application of Experimental Design Techniques,” *Statistica Sinica*, vol. 8, no. 1, pp. 165–184, 1998.
[55] B. Smith and M. S. Feather, “Challenges and Methods in Testing the Remote Agent Planner,” *IN PROC. 5TH INT.NL CONF. ON ARTIFICIAL INTELLIGENCE PLANNING AND SCHEDULING (AIPS*, vol. 2000, p. 254–263, 2000.
[56] R. Krishnan, S. M. Krishna, and P. S. Nandhan, “Combinatorial testing: learnings from our experience,” *SIGSOFT Softw. Eng. Notes*, vol. 32, no. 3, pp. 1–8, May 2007.
[57] Y. Lei, R. H. Carver, R. Kacker, and D. Kung, “A combinatorial testing strategy for concurrent programs,” *Softw. Test. Verif. Reliab.*, vol. 17, no. 4, pp. 207–225, Dec. 2007.
[58] Wenhua Wang, Yu Lei, Donggang Liu, D. Kung, C. Csallner, Dazhi Zhang, R. Kacker, and R. Kuhn, “A combinatorial approach to detecting buffer overflow vulnerabilities,” in *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, 2011, pp. 269–278.
[59] R. C. Bryce, S. Sampath, and A. M. Memon, “Developing a Single Model and Test Prioritization Strategies for Event-Driven Software,” *IEEE Transactions on Software Engineering*, vol. 37, no. 1, pp. 48–64, Feb. 2011.