Constraints: the Future of Combinatorial Interaction Testing

Justyna Petke*
*Centre for Research, Evolution, Search and Testing
University College London, Email: j.petke@ucl.ac.uk

***Abstract*—Combinatorial Interaction Testing (CIT) has gained a lot of attention in the area of software engineering in the last few years. CIT problems have their roots in combinatorics. Mathematicians have been concerned with the NP-complete problem of finding minimal covering arrays (in other words, minimal CIT test suites) since early nineties. With the adoption of these techniques into the area of software testing, an important gap has been identified - namely consideration of real-world constraints. We show that indeed finding an efficient way of handling constraints during search is the key factor in wider applicability of CIT techniques.**

I. Introduction

Many real-world software systems are highly configurable. It is usually infeasible to test all the possible configurations. In order to avoid running into this *combinatorial explosion problem*, Combinatorial Interaction Testing (CIT) techniques have been developed specifically for such systems. CIT combines all t-combinations of parameter inputs or configuration options in a systematic way so that we know we have tested a measured subset of the input or configuration space. Different parameter values can be set, for instance, via user interface. An example would be web browser configurations, as shown in Table I.

TABLE I
Web Browser Configurations

Load content	Notify pop-up blocked	Cookies	Warn before add-ons install	Remember downloads
Allow	Yes	Allow	Yes	Yes
Restrict	No	Restrict	No	No
Block		Block		

Historically, CIT problems come from the field of combinatorics and are usually represented as a covering array (CA): $CA(N; t, v_1^{k_1} v_2^{k_2} ... v_m^{k_m})$, where N is the size of the array, t is its strength, sum of $k_1, ..., k_m$ is the number of parameters and each v_i stands for the number of values for each of the k_i parameters in turn. Suppose we want to generate a pairwise interaction test suite for an instance presented in Table I. It has 5 parameters, two of which can take 3 values ('Allow', 'Restrict' and 'Block'), while others have two choices of values ('Yes' and 'No'). The pairwise CIT problem is then formulated as: $CA(N; 2, 3^1 2^1 3^1 2^2)$. Furthermore, in order to test all combinations one would need $3 * 2 * 3 * 2 * 2 = 72$ test cases. If, however, we cover all interactions between any two parameters, then we only need 9 test cases. Such a test suite is called a 2-way or pairwise test suite. The goal is to find the smallest covering array (or, in other words, CIT test suite) that covers all possible combinations of values, i.e. *interactions* between any set of t parameters.

Generation of a minimal CIT test suite is a very challenging task, in fact, the complexity is NP-complete. Therefore, exact methods, like the one by Hnich et al. from 2006 [1] have not been as successful as heuristics in real-world applications. There are several approaches for covering array generation. The two most popular ones use either simulated-annealing (SA) or a greedy algorithm. The SA-based approach is believed to produce smaller test suites, while the greedy one is regarded to be faster [2]. Among other search-based methods for CIT test suite generation are genetic and ant colony algorithms [3]. A comprehensive survey on Combinatorial Interaction Testing has been conducted by Nie and Leung in 2011 [4]. More recently, Khalsa and Labiche reported the abundance of tools and algorithms available for CIT [5].

There have been several studies showing that CIT test suites are able to discover all the known interaction faults of the system under test [6], [7], [8], [9]. More importantly, CIT is a light-weight, black-box testing technique. It is able to efficiently test and identify the cause of faults in a real-world system without any knowledge of the inner workings of the system under test. Thereofore, Combinatorial Interaction Testing has been used successfully as a system level test method [6], [7], [8], [9], [10], [11], [12], [13].

II. Constraints in CIT

Even though CIT has been successful in discovering existing faults, in situations where there are hundreds of parameters like in the case of Software Product Lines [14], sometimes generation of even pairwise interaction test suite is a challenging task. Furthermore, such a test suite might be simply too big to be used in practice. Furthermore, until recently, higher-strength CIT testing, that is, 3-way, 4-way and higher, was deemed unfeasible in real-world situations [15].

However, in most CIT applications, the problem domain is constrained: some interactions are simply infeasible due to these constraints [1], [16], [17], [18]. The nature and description of such constraints is highly domain specific, yet taking account of them is essential in order for CIT to be usable in practice. Any CIT approach that fails to take account of constraints will produce many test cases that are either unachievable in practice or which yield expensively misleading results (such as false positives). An example of such a *hard* *constraint* for the example shown in Table I is: “Warn before add-ons install” must always be turned on, that is, it can only take the “Yes” value.

Another type of constraint, often referred to as a *soft constraint* [16] may also have a role to play. Soft constraints are combinations of options that a tester believes do not need to be tested together (based either on their knowledge of the test subject and/or by a static analysis). Catering for such constraints will not improve test effectiveness, but it may improve efficiency. An example would be testing the “find” function that searches through a file looking for a particular pattern. If an empty file is supplied, it is not necessary to test searches of all possible patterns of words, since, given an empty file, an error should be thrown in each case. However, there has been little work on CIT with this type of soft constraint.

By prohibiting certain parameter-value interactions, constraints may significantly reduce the search space for a CIT test suite generation algorithm. This has been exemplified in recent work [15]. In particular, it has been shown that it is the combination of soft and hard constraints that allows for a very popular CIT-test generation algorithm, based on simulated-annealing, to scale to real-world problems. Furthermore, it made higher-strength testing, in particular, testing of all value-combinations of any 5 parameters, feasible.

III. Future Research Directions

Even though Combinatorial Interaction Testing is a relatively mature research area, the constraint handling techniques leave room for improvement. Last years' survey of CIT tools [5] reveals that many tools simply do not implement any constraint handling method. One of the challenges has been efficient integration of constraint handling within the search process. For example, when using a heuristic such as simulated-annealing, a test case mutation can result in constraint violation. Therefore, an additional check needs to be made.

Arguably the first method deployed is the one that filters forbidden interactions. In other words, a list of disallowed parameter-value combinations is kept in memory. Each generated test case is then verified against this set of forbidden assignments and/or disallowed interactions are prevented during the search process. For example, the greedy-based ACTS tool [19], implements this method.

Alternatively one can choose a Boolean satisfiability or constraint solver to resolve constraints [2], [20]. However, these tools have been used as black-boxes. Especially in the case of constraint solvers, the right configuration might drastically speed-up the solving process.

We believe that further research into constraint handling methods during search in generation of Combinatorial Interaction Testing test suites can not only speed-up existing tools, but also reveal new algorithms efficient for higher-strength CIT testing.

References

[1] B. Hnich, S. D. Prestwich, E. Selensky, and B. M. Smith, “Constraint models for the covering test problem,” *Constraints*, vol. 11, no. 2-3, pp. 199–219, 2006.

[2] B. J. Garvin, M. B. Cohen, and M. B. Dwyer, “Evaluating improvements to a meta-heuristic search for constrained interaction testing,” *Empirical Software Engineering*, vol. 16, no. 1, pp. 61–102, 2011.

[3] T. Shiba, T. Tsuchiya, and T. Kikuno, “Using artificial life techniques to generate test cases for combinatorial testing,” in *28th International Computer Software and Applications Conference (COMPSAC 2004), Design and Assessment of Trustworthy Software-Based Systems, 27-30 September 2004, Hong Kong, China, Proceedings*, 2004, pp. 72–77.

[4] C. Nie and H. Leung, “A survey of combinatorial testing,” *ACM Comput. Surv.*, vol. 43, no. 2, pp. 11:1–11:29, 2011.

[5] S. K. Khalsa and Y. Labiche, “An orchestrated survey of available algorithms and tools for combinatorial testing,” in *25th IEEE International Symposium on Software Reliability Engineering, ISSRE 2014, Naples, Italy, November 3-6, 2014*, 2014, pp. 323–334.

[6] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, “The AETG system: an approach to testing based on combinatorial design,” *IEEE Transactions on Software Engineering*, vol. 23, no. 7, pp. 437–444, 1997.

[7] C. Yilmaz, M. B. Cohen, and A. Porter, “Covering arrays for efficient fault characterization in complex configuration spaces,” *IEEE Transactions on Software Engineering*, vol. 31, no. 1, pp. 20–34, January 2006.

[8] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, “Software fault interactions and implications for software testing,” *IEEE Trans. Software Eng.*, vol. 30, no. 6, pp. 418–421, 2004.

[9] X. Qu, M. B. Cohen, and K. M. Woolf, “Combinatorial interaction regression testing: A study of test case generation and prioritization,” in *ICSM*. IEEE, 2007, pp. 255–264.

[10] M. B. Cohen, C. J. Colbourn, P. B. Gibbons, and W. B. Mugridge, “Constructing test suites for interaction testing,” in *Proceedings of the International Conference on Software Engineering*, May 2003, pp. 38–48.

[11] D. R. Kuhn and V. Okun, “Pseudo-exhaustive testing for software,” in *SEW*. IEEE Computer Society, 2006, pp. 153–158.

[12] S. Sampath, R. C. Bryce, G. Viswanath, V. Kandimalla, and A. G. Koru, “Prioritizing user-session-based test cases for web applications testing,” in *ICST*. IEEE Computer Society, 2008, pp. 141–150.

[13] X. Qu, M. B. Cohen, and G. Rothermel, “Configuration-aware regression testing: an empirical study of sampling and prioritization,” in *Proceedings of the International Symposium On Software Testing and Analysis*, 2008, pp. 75–86.

[14] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. L. Traon, “Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test suites for large software product lines,” *CoRR*, vol. abs/1211.5451, 2012.

[15] J. Petke, M. B. Cohen, M. Harman, and S. Yoo, “Efficiency and early fault detection with lower and higher strength combinatorial interaction testing,” in *European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13*. Saint Petersburg, Russian Federation: ACM, August 2013, pp. 26–36.

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

[17] M. B. Cohen, M. B. Dwyer, and J. Shi, “Interaction testing of highly-configurable systems in the presence of constraints,” in *ISSTA*, D. S. Rosenblum and S. G. Elbaum, Eds. ACM, 2007, pp. 129–139.

[18] ——, “Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach,” *IEEE Trans. Software Eng.*, vol. 34, no. 5, pp. 633–650, 2008.

[19] 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, 2008.

[20] M. Banbara, H. Matsunaka, N. Tamura, and K. Inoue, “Generating combinatorial test cases by efficient SAT encodings suitable for CDCL SAT solvers,” in *Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17, Yogyakarta, Indonesia, October 10-15, 2010. Proceedings*, 2010, pp. 112–126.