Set-Based Algorithms for Combinatorial Test Set Generation

Ludwig Kampel, Dimitris E. Simos

Set-based Algorithms for Combinatorial Test Set Generation

Ludwig Kampel and Dimitris E. Simos

SBA Research, Vienna, Austria
`lkampel@sba-research.org, dsimos@sba-research.org`

Abstract. Testing is an important and expensive part of software and hardware development. Over the recent years, the construction of combinatorial interaction tests rose to play an important role towards making the cost of testing more efficient. Covering arrays are the key element of combinatorial interaction testing and a means to provide abstract test sets. In this paper, we present a family of set-based algorithms for generating covering arrays and thus combinatorial test sets. Our algorithms build upon an existing mathematical method for constructing independent families of sets, which we extend sufficiently in terms of algorithmic design in this paper. We compare our algorithms against commonly used greedy methods for producing 3-way combinatorial test sets, and these initial evaluation results favor our approach in terms of generating smaller test sets.

Keywords: Combinatorial testing, independent families of sets, set-based algorithms

1 Introduction

In modern software development testing plays an important role and therefore requires a large amount of time and resources. According to a report of the National Institute of Standards in Technology (NIST) [1], faults in software costs the U.S. economy up to \$59.5 billion per year, where these costs could be reduced by \$22.2 billion, provided better software testing infrastructure. Another report from NIST [11] shows that failures appear to be caused by the interaction of only few input parameters of the system under test (SUT). Combinatorial testing guarantees good input-space coverage, while reducing the resources needed for testing. In particular, it is a t-wise testing strategy whose key ingredient is a Covering Array (CA), a abstract mathematical object that provides coverage of all t-way interactions of a certain amount of input parameters, reducing the amount of tests that need to be executed. For their use in practice, the columns of CAs are identified with the input parameters of the SUT, where each entry in a certain column is mapped to a value of the corresponding parameter [12]. This way each row of the CA translates to a certain parameter value setting of the input model of the SUT which can be used as a test. Translating each row of a CA in this way, one obtains a *concrete test set* hence a CA can be regarded

as an *abstract combinatorial test set*. To reduce further the amount of resources needed for testing, one is interested to construct optimal CAs (e.g. arrays of a minimal size that provide maximal coverage). This software testing problem is tightly coupled with hard combinatorial optimization problems for CAs (shown to be NP-hard [17]).

Contribution. In this paper, we use a set-based method for constructing CAs based on independent families of sets (IFS) from [7]. There exists an equivalence between these two combinatorial objects which allowed us to use the two discrete structures interchangeably in terms of algorithmic design. In particular, we extend this set-based method with balancing properties that can impose restrictions on the cardinality of the appearing intersections. This (among other concepts) enabled us to define different building blocks that give rise to a family of algorithms based on IFSs (and consequently also for CAs). Furthermore, as a proof of concept we compared our algorithms against a widely used combinatorial strategy (the so-called IPO-strategy [15]) which bares similarities with our approach for constructing and extending CAs. Our initial results outperform this strategy for 3-way testing, generating better sized covering arrays.

Structure of the paper. In Section 2 we give some preliminaries for CAs, where we also review related algorithms and problems for the former objects. Afterwards, in Section 3 we describe a set-based method for constructing CAs and extend it with concepts necessary for devising an algorithmic concept later on Section 4, in which we also propose a variety of algorithms for generating CAs. Subsequently, in Section 5 we compare our algorithms against IPO-strategy greedy techniques for constructing CAs and comment on the evaluated results. Finally, 6 concludes the work and discusses future directions of work.

2   Problems and Algorithms for Covering Arrays

In this section we give a short overview of the needed definitions, as well as of related problems, related algorithms and work in general. In the following we frequently use the abbreviation $[N]$ for a set $\{1, \ldots, N\} \subseteq \mathbb{N}$ and also A^C denotes the complement $[N] \setminus A$ of A in $[N]$. The definitions given below are slightly different phrased as those given in [5], and can also be found in [13].

2.1   Preliminaries for Covering Arrays

Definition 1 (t-Independent Family of Sets). *A t-independent family of sets, $IFS(N; t, k)$, is a family $(A_1, \ldots, A_k)$ of k subsets of $[N]$, with the property that for each choice $\{i_1, \ldots, i_t\} \subseteq [k]$ of t different indices, for all $j \in [t]$ and for all $\bar{A}_{i_j} \in \{A_{i_j}, A_{i_j}^C\}$ it holds that $\bigcap_{j=1}^{t} \bar{A}_{i_j} \neq \emptyset$. The parameters t and k are called, respectively, the strength and the size of the IFS.*

We say that a family of sets is *t-independent* if it is an $IFS(N; t, k)$ for some value of N and k. Without loss of generality we only consider IFS over a underlying set $[N]$ with $N \in \mathbb{N}$.

$$
\begin{aligned}
A_1 &= \{6,7,8,9,10,11\} \leftrightarrow (0,0,0,0,0,1,1,1,1,1,1,0)^T = a_1 \\
A_2 &= \{1,2,3,6,7,8\} \quad\leftrightarrow (1,1,1,0,0,1,1,1,0,0,0,0)^T = a_2 \\
A_3 &= \{1,2,4,6,9,10\} \quad\leftrightarrow (1,1,0,1,0,1,0,0,1,1,0,0)^T = a_3 \\
A_4 &= \{1,2,5,7,9,11\} \quad\leftrightarrow (1,1,0,0,1,0,1,0,1,0,1,0)^T = a_4 \\
B &= \{1,2,5,8,9,10\} \quad\leftrightarrow (1,1,0,0,1,0,0,1,1,1,0,0)^T = b
\end{aligned}
$$

Table 1. The sets A_1, A_2, A_3, A_4 and B are considered as subsets of $[12]$. We identify them with their binary indicator vectors, i.e. vectors in $\{0,1\}^{12}$ that have 1 in position i if, and only if i is element of the corresponding set, and 0 otherwise.

Example 1. From Table 1 the family $\mathcal{A} = (A_1, A_2, A_3, B)$ is an $IFS(12;3,4)$, i.e. if we choose 3 sets of $\mathcal{A}$ or independently their complements, their intersection is nonempty. For example, $A_1 \cap A_3^C \cap B = \{8\} \neq \emptyset$ and $A_1 \cap A_2 \cap A_3^C = \{7,8\} \neq \emptyset$.

Definition 2 (binary t-Covering Array). *A $N \times k$ binary array M, denoted in column form as $M = (\mathbf{m_1}, \ldots, \mathbf{m_k})$, is a binary t-covering array, $CA(N;t,k)$, if M has the property that for each $\{i_1, \ldots, i_t\} \subseteq [k]$, the corresponding $t \times N$ sub array $(\mathbf{m_{i_1}}, \ldots, \mathbf{m_{i_t}})$ of M cover all binary t-tuples $\{0,1\}^t$, i.e. these tuples have to appear at least once as a row of the sub array $(\mathbf{m_{i_1}}, \ldots, \mathbf{m_{i_t}})$. In some cases M is also called a binary covering array of strength t.*

Remark 1. Covering arrays of fixed non-binary alphabet with size u are denoted with $CA(N;t,k,u)$ in the literature (e.g. see [5]). When $u = 2$ is clear from the context we simply use the notation introduced as above.

As the similarity of the former definitions of these combinatorial objects implies, there is a close relation between the two of them. For example, it is known that every $CA(N;t,k)$ is equivalent to an $IFS(N;t,k)$ (see for example [13] and [5], Remark 10.5).

Example 2. From Table 1 we take the vectors a_1, a_2, a_3 and b to form the array

$$
A = (a_1, a_2, a_3, b) = \begin{pmatrix} 0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,0 \\ 1\,1\,1\,0\,0\,1\,1\,1\,0\,0\,0\,0 \\ 1\,1\,0\,1\,0\,1\,0\,0\,1\,1\,0\,0 \\ 1\,1\,0\,0\,1\,0\,0\,1\,1\,1\,0\,0 \end{pmatrix}^T,
$$

which is equivalent to the IFS given in Example 1. The defining property of an IFS translates to the defining property of a binary CA. In this case, within each three selected columns of (a_1, a_2, a_3, b), each binary 3-tuple appears at least once. Therefore the given array A is a $CA(12;3,4)$. On the other hand, the IFS in Example 1 can be uniquely reconstructed from the array A, interpreting its columns as indicator vectors of subsets of $[12]$.

Definition 3. *The smallest number of rows N such that a binary $CA(N;t,k)$ exists is defined as $CAN(t,k) := \min\{N : \exists \ CA(N;t,k)\}$.*

Definition 4. *The largest number k such that a $IFS(N;t,k)$ exists is defined as $CAK(N;t) := max\{k : \exists\ IFS(N;t,k)\}$.*

For an overview of the vast amount of theoretical and computational problems that arise in the theory of CAs we refer to [4], [9]. Especially the problem of determining binary CAs with minimum amount of rows turns out to be NP-hard (see [17]).

2.2   Algorithms for Covering Arrays

The notorious difficulty of constructing optimal CAs has been the subject of many algorithmic approaches. The most related ones to our work are greedy methods such as AETG [2] and IPO [15]. AETG employs a randomized, greedy, one row at a time extension strategy. The IPO-strategy is to grow the covering array in both dimensions. Horizontal growth adds one column to the current array by its cells with entries in a greedy manner. Vertical extension is performed, by adding rows until the array is once again a CA. Adjusting the parameters of the IPO-strategy has been the subject of [8]. Finally, in [7] a method is proposed that produces exponentially sized IFS one set at a time. In terms of CAs this comes down to a *one column at a time* construction of a binary CA. As this method plays a pivotal role in our work, we further describe it in Section 3.

Due to space limitations, for other related works we refer the interested reader to a recent survey [18].

3   A Set-Based Method for Constructing CAs

In this section we elaborate on a set-based method for constructing CAs and extend it with concepts necessary for devising an algorithmic concept later on in Section 4.

Before the description of the method, we have to define some terms needed. It is well known that Orthogonal Arrays of index one are optimal CAs [5], i.e. within each selection of t columns each binary t-tuple appears exactly once.

Also when constructing a CA with as few rows as possible, one tends to not cover certain t-tuples multiple times; rather the target would be to cover as few t-tuples as possible more than once. Lets consider the case of a CA $(\mathbf{a_1}, \ldots, \mathbf{a_r})$, where only few t-tuples appear more than once within a certain choice $\mathbf{c} = (\mathbf{a_{i_1}}, \ldots, \mathbf{a_{i_t}})$ of t columns of that array. Since for each $(t-1)$-tuple $(u_1, \ldots, u_{t-1})$ there are exactly two binary t-tuples, that start with $(u_1, \ldots, u_{t-1})$, namely $(u_1, \ldots, u_{t-1}, 0)$ and $(u_1, \ldots, u_{t-1}, 1)$. We know that within $(a_{i_1}, \ldots, a_{i_{t-1}})$ each $(t-1)$-tuple appears at least twice, and only few of them appear more than twice. Of course, this argument holds for each choice of $(t-1)$ columns of $\mathbf{c}$.

Remark 2. Note as well that this argumentation can be iterated. From these thoughts we design a necessary condition when a column is allowed to be added to the current array. In particular, we want to ensure a minimum amount of *balance* among the columns of the array in the regard just described.

In light of the previous remark, we introduce the notion of α-balance.

Definition 5. *Let $\mathcal{A} = (A_1, \ldots, A_k)$ be a family of sets $A_i \subseteq [N] \; \forall i \in [k]$ and $\alpha = (\alpha_1, \ldots, \alpha_s) \in \mathbb{N}^s$, $s \leq k$. We say that A is α-balanced, if*

$$\forall i \in [s] \; \forall \{j_1, \ldots, j_i\} \subseteq [k] \; \forall \bar{A}_{j_r} \in \{A_{j_r}, A_{j_r}^C\} : \; \Big| \bigcap_{r=1}^{i} \bar{A}_{j_r} \Big| \geq \alpha_i. \qquad (1)$$

Note that if a family of sets is $(\alpha_1, \ldots, \alpha_s)$-balanced and $\alpha_s \geq 1$ then it is also s-independent.

Definition 6. *Let $B \subseteq [N]$, $\mathcal{A} = (A_1, \ldots, A_k)$ be a family of sets $A_i \subseteq [N] \; \forall i \in [k]$ and $\alpha = (\alpha_1, \ldots, \alpha_s) \in \mathbb{N}^s$. We say that B α-balanced with respect to $\mathcal{A}$, if the family $(A_1, \ldots, A_k, B)$ of sets is α-balanced.*

Example 3. Consider the family $\mathcal{F} = (A_1, A_2, A_3, A_4)$, constructed from the sets of Table 1. This family is a $(6, 3)$-balanced family of sets, i.e. each set, as well as its complement, has at least cardinality 6, and all intersections of any two sets of $\mathcal{F}$ (complements might be involved) have at least cardinality 3 (e.g. $A_1 \cap A_3^C = \{7, 8, 11\} \geq 3$). B is an example for a set that is not $(6, 3)$-balanced w.r.t. $\mathcal{F}$, since $A_3^C \cap B = |\{5, 8\}| = 2 < 3$.

4 A New Family of IFS Algorithms

In this section, we propose a variety of algorithms, IFS-ORIGIN, IFS-GREEDY and IFS-SCORE, based on independent families of sets. We call this class of algorithms collectively a family of IFS-Algorithms. In particular, we formalized and extended in terms of a combinatorial algorithmic design the method described earlier. Our design is comprised of the following five *building blocks*: *store*, *select*, *admissible*, *extend* and *update* which we state below.

- *Store:* The *store* is a data structure that serves as a resource, from which the sets to build the target IFS are chosen. It may be static, or dynamic.
- *Select:* A procedure that returns one element of the *store*, e.g. randomly or via a scoring function.
- *Admissible:* This procedure decides whether a certain element is allowed to be added to the current IFS or not under certain admissible criteria which can be based for example on the concept of α-balance.
- *Extend:* A procedure that extends the IFS at hand.
- *Update:* The procedure which updates the *store* in case latter is dynamic.

In the following we frequently use $F_2(N) := \{A \subseteq [N] | N \notin A \wedge |A| = \lceil N/2 \rceil\}$, which is a 2-independent family of sets of maximal size (cf. [10]). A comprehensive overview of the proposed algorithms via their building blocks, is given below in Table 4.

Table 2. Composition of the IFS-family algorithms.

Algorithm / Building blocks	IFS-ORIGIN	IFS-GREEDY	IFS-SCORE
Store	F_2	F_2	F_2
Select	SELECTRANDOM	SELECTNEXT	SELECTSCORE
Admissible	ADMISSIBLE$_\alpha$	ADMISSIBLE$_\alpha$	ADMISSIBLE
Extend	EXTEND	EXTEND	EXTEND
Update	UPDATE$_\alpha$	-	UPDATE

4.1 IFS-Origin

Firstly we give a short algorithmic description of the method proposed in [7] and extended in Section 3. We refer to it and its implementation as IFS-ORIGIN. The algorithm takes as input the size N of the underlying set and the strength t of the to be constructed IFS. The initial STORE, S_0, is set to be equal to $F_2(N)$ and the initial IFS, A_1, is set to be a random element of the STORE. This random initialization is justified because picking a different initial element boils down to permuting the first $[N-1]$ elements of $[N]$, which also respects Definition 1, and keeps $F_2(N)$ invariant under such permutations. From now on in each step i the IFS-ORIGIN traverses through the whole STORE S_{i-1} given at that time, updating it by removing all non-ADMISSIBLE$_\alpha$ (ADMISSIBLE$_\alpha$ checks for α-balance and t-independence) elements from it, which yields S_i. For the admissibility check the algorithm requires a vector $\alpha_{1\times(t-1)}$, which encodes the desired balance of i-tuples for $i = 1, \ldots, t-1$. Thereafter, now that S_i is left with only ADMISSIBLE$_\alpha$ elements, a random element is chosen and added to the IFS at hand, yielding A_{i+1}. The algorithm terminates when the STORE is empty.

4.2 IFS-Greedy

When being familiar with IFS-ORIGIN described above, one will realize, that this version, as was originally given in [7] lacks of a method to decide which of the elements in the remaining STORE should be added to the current array. In particular, this is done via a random pick, which in retrospect makes the UP-DATE of the STORE, which leaves the STORE with only ADMISSIBLE$_\alpha$ elements inside, unnecessary. The newly proposed IFS-GREEDY version bypass this decision problem by simply taking the next found ADMISSIBLE$_\alpha$ element of the STORE, having the advantage that the STORE has never to be updated. The initialization stays the same as in IFS-ORIGIN. After that IFS-GREEDY traverses the STORE only once, adding the first element that is ADMISSIBLE with respect to the already chosen ones and α (recall Definition 6). The STORE never gets updated.

4.3   IFS-Score

The overall structure of IFS-SCORE is the same as that of IFS-ORIGIN, but different building blocks SELECTSCORE and ADMISSIBLE are defined. To circumvent the problem of IFS-ORIGIN of picking a *random element* from the updated STORE, we calculated a score for each element of the STORE, that reflects α-balance, and add the one (or one of those, since ties may occur) with the least score. Each element is initialized with a score of zero and in the i-th step of the algorithm we calculate again a score for each element of the current STORE, S_{i-1}, as before. This has also the advantage that IFS-SCORE does not require α. Since we compute a score for each element, we already encounter the tuple balance of (A_i, b) to our selection and we do not need to previously dictate via α how often certain i-tuples have to appear. Therefore IFS-SCORE is the only algorithm in the proposed IFS-family that does *not* require an input of α. Consequently, in an element of the STORE passes the decision criterion of ADMISSIBLE, if and only if (A, b) is t-independent.

5   Results

As a proof of concept of our algorithmic design (cf. Section 4) we compared our implementations of the IFS-family of algorithms for $t = 3$ to two of the most commonly used greedy algorithms of the IPO-family, namely IPOG [14] and IPOG-F [6]. In addition, we evaluate our results versus the *current* best known upper bounds for $CAK(N; 3)$ (retrieved from [3], via $CAK(N; t) = max\{k | CAN(k, t) \leq N\}$, cf.[13]), that are combined results of algorithms and methods that are partly described in [18]. To the best of our knowledge the algorithms of the IPO-family are the only ones that generate CAs using a horizontal extension step similar to the one proposed in the IFS-family of algorithms.

Table 5 shows the amount of columns a binary CA of strength 3 can attain by either the respective algorithm compared or according to [3]. Table 5 starts with $N = 8$, since there are at least eight rows needed to cover all eight binary 3-tuples. It shows that the IFS-family of algorithms improves significantly over IPOG and IPOG-F in almost every case presented, as well that IFS-GREEDY and IFS-SCORE improve over IFS-ORIGIN. It is also worth pointing out that during our computations we obtained larger families, when running IFS-ORIGIN and IFS-GREEDY on more restrictive α-vectors than running them on less restrictive α-vectors. We believe the concept of admissibility via α-balance (and its requirement per different IFS algorithms) makes the difference versus IPOG and IPOG-F, since these algorithms lack of a balancing strategy during horizontal extension. Regarding our results, we want to highlight that IFS-SCORE is able to deliver almost the same size of output IFS as IFS-GREEDY without the need of an α-vector as input. On the other hand, IFS-SCORE is more complex than IFS-GREEDY and even IFS-ORIGIN due to score computations.

The values for IPOG-F in Table 5 are taken from [16]. For the experimental evaluation we run IPOG locally as it is implemented in ACTS, a CA generation tool provided by NIST [19]. For the input values of N in Table 5, IPOG and

IPOG-F were considerably faster than all three of our algorithms. We think that the extra computations are fully justified, since the IFS-family of algorithms outperforms IPOG and IPOG-F, in 14 out of the 18 documented cases in terms of output size of produced IFS (or columns of produced CAs respectively) and achieves the same size values in the other four. Especially, if we consider that in our experiments the main objective was to compare to the best bounds provided by greedy algorithms.

Table 3. Comparison of the amount of columns attained on N rows by different CA algorithms (larger is better). Information for the best lower bound for $CAK(N; t)$ where $t = 3$ is provided by Colbourn Tables [3]. The superscripts denote the α-vector that was used as input for the computation that yields the output IFS, where a=(4,2), b=(6,3), c=(8,4), d=(10,5).

N	IPOG-F	IPOG	IFS-ORIGIN	IFS-GREEDY	IFS-SCORE	Colbourn Tables
8	4	4	4^a	4^a	4	4
9	4	4	4^a	4^a	4	4
10	4	4	4^a	5^a	5	5
11	5	4	4^a	5^a	5	5
12	5	6	11^b	11^b	11	11
13	5	6	6^b	11^b	11	11
14	6	6	6^b	11^b	11	11
15	6	6	7^b	11^b	11	12
16	7	7	8^c	14^c	14	14
17	9	7	10^c	14^c	14	16
18	11	8	12^c	17^c	16	20
19	12	8	13^c	17^c	16	22
20	13	10	11^d	19^d	19	23
21	15	10	15^c	19^c	19	25
22	16	12	18^c	21^c	21	26
23	16	13	19^c	23^c	22	30
24	19	13	23^d	26^d	25	38
25	21	14	24^c	28^a	26	44

6   Conclusion and Future Work

In this paper, we present a family of set-based algorithms for covering arrays, which can be regarded as abstract combinatorial test sets, based on independent families of sets. Our algorithmic design is modular thanks to a variety of building blocks which can give rise to even more algorithms than the ones presented. As a proof of concept of our approach we compared the implementations of the proposed family against state of the art greedy algorithms that are also used in practice for 3-way testing. This initial evaluation shows, that our approach improves significantly, in terms of size, over the existing greedy algorithm, which translates to smaller test sets. As future work, we plan to enhance the functionality of our algorithms via extending it to produce combinatorial test sets over non-binary alphabets as well as conduct more experiments for test sets that can be used for higher strength interaction testing.

References

1. The economic impacts of inadequate infrastructure for software testing. National Institute of Standards and Technology, U.S. Department of Commerce (2002)
2. Cohen, D.M., Dalal, S.R., Fredman, M.L., Patton, G.C.: The aetg system: An approach to testing based on combinatorial design. Software Engineering, IEEE Transactions on 23(7), 437–444 (1997)
3. Colbourn, C.J.: Table for CAN(3,k,2) for k up to 10000. Available at `http://www.public.asu.edu/~ccolbou/src/tabby/3-2-ca.html`, Accessed on 2016-04-25
4. Colbourn, C.J.: Combinatorial aspects of covering arrays. Le Matematiche (Catania) 58(121-167), 0–10 (2004)
5. Colbourn, C.J., Dinitz, J.H.: Handbook of combinatorial designs. CRC press (2006)
6. Forbes, M., Lawrence, J., Lei, Y., Kacker, R., Kuhn, D.R.: Refining the in-parameter-order strategy for constructing covering arrays. Journal of Research of the National Institute of Standards and Technology 113, 287–297 (2008)
7. Freiman, G., Lipkin, E., Levitin, L.: A polynomial algorithm for constructing families of k-independent sets. Discrete mathematics 70(2), 137–147 (1988)
8. Gao, S.W., Lv, J.H., Du, B.L., Colbourn, C.J., Ma, S.L.: Balancing frequencies and fault detection in the in-parameter-order algorithm. Journal of Computer Science and Technology 30(5), 957–968 (2015)
9. Hartman, A., Raskin, L.: Problems and algorithms for covering arrays. Discrete Mathematics 284(13), 149 – 156 (2004)
10. Kleitman, D.J., Spencer, J.: Families of k-independent sets. Discrete Mathematics 6(3), 255–262 (1973)
11. Kuhn, D., Kacker, R., Lei, Y.: Practical combinatorial testing. NIST Special Publication 800-142 (2010)
12. Kuhn, D., Kacker, R., Lei, Y.: Introduction to Combinatorial Testing. Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series, Taylor & Francis (2013)
13. Lawrence, J., Kacker, R.N., Lei, Y., Kuhn, D.R., Forbes, M.: A survey of binary covering arrays. the electronic journal of combinatorics 18(1), P84 (2011)
14. Lei, Y., Kacker, R., Kuhn, D.R., Okun, V., Lawrence, J.: IPOG-IPOG-D: Efficient test generation for multi-way combinatorial testing. Softw. Test. Verif. Reliab. 18(3), 125–148 (2008)
15. Lei, Y., Tai, K.C.: In-parameter-order: A test generation strategy for pairwise testing. In: High-Assurance Systems Engineering Symposium, 1998. Proceedings. Third IEEE International. pp. 254–261. IEEE (1998)
16. NIST: Table for CA(3,k,2). National Institute of Standards and Technology,Available at `http://math.nist.gov/coveringarrays/ipof/tables/table.3.2.html`, Accessed on 2016-04-25
17. Seroussi, G., Bshouty, N.H.: Vector sets for exhaustive testing of logic circuits. Information Theory, IEEE Transactions on 34(3), 513–522 (1988)
18. Torres-Jimenez, J., Izquierdo-Marquez, I.: Survey of covering arrays. In: Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2013 15th International Symposium on. pp. 20–27. IEEE (2013)
19. Yu, L., Lei, Y., Kacker, R.N., Kuhn, D.R.: Acts: A combinatorial test generation tool. In: Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on. pp. 370–375. IEEE (2013)