A Post-optimization Strategy for Combinatorial Testing: Test Suite Reduction through the Identification of Wild Cards and Merge of Rows

Loreto Gonzalez-Hernandez[1], Jose Torres-Jimenez[1], Nelson Rangel-Valdez[2], and Josue Bracho-Rios[1]

[1] Information Technology Laboratory, CINVESTAV-Tamaulipas
Km. 5.5 Carretera Cd. Victoria-Soto la Marina, 87130, Cd. Victoria Tamps., Mexico
[2] Polytechnic University of Victoria
Km. 5.5 Carretera Cd. Victoria-Soto la Marina, 87138, Cd. Victoria Tamps., Mexico
agonzalez@tamps.cinvestav.mx, jtj@cinvestav.mx, nrangelv@upv.edu.mx,
jbracho@tamps.cinvestav.mx
http://www.tamps.cinvestav.mx/~agonzalez/
http://www.tamps.cinvestav.mx/~jtj/

Abstract. The development of a new software system involves extensive tests on the software functionality in order to identify possible failures. It will be ideal to test all possible input cases (configurations), but the exhaustive approach usually demands too large cost and time. The test suite reduction problem can be defined as the task of generating small set of test cases under certain requirements. A way to design test suites is through interaction testing using a matrix called Covering Array, CA$(N; t, k, v)$, which guarantees that all configurations among every t parameters are covered. This paper presents a simple strategy that reduces the number of rows of a CA. The algorithms represent a post-optimization process which detects wild cards (values that can be changed arbitrarily without the CA losses its degree of coverage) and uses them to merge rows. In the experiment, 667 CAs, created by a state-of-the-art algorithm, were subject to the reduction process. The results report a reduction in the size of 347 CAs (52% of the cases). As part of these results, we report the matrix for $CA(42; 2, 8, 6)$ constructed from $CA(57; 2, 8, 6)$ with an impressive reduction of 15 rows, which is the best upper bound so far.

Keywords: Combinatorial testing, Test suite reduction, Covering arrays, Wild cards, IPOG-F.

1 Introduction

Software systems are widely used in our society and they are present in such daily activities as social networking and mobile devices; furthermore they take an important role in scientific and technological development. Therefore, the

I. Batyrshin and M. González Mendoza (Eds.): MICAI 2012, Part II, LNAI 7630, pp. 127–138, 2013.

quality of life of our society is greatly influenced by the software reliability; on the contrary, software failures can cause large losses in the economy [22] or even affect the health or life of people, as stipulated in the records of the Therac 25 crash [18] and the failure of the Ariane 5 rocket [19].

The quality of a software system is highly related to software testing [12]. Software testing has the aim to detect existing defects in a software system, such that the bugs can be corrected before it begins to be used.Various kinds of techniques and methods to ensure software quality have been developed in order to detect different types of failures, one classification involves the *white-box* and the *black-box* test strategies [4]. The white-box approach uses an internal perspective of the software. This approach requires that the tester has programming skills to identify all execution paths through the source code. In contrast, black-box is a functional testing technique, it takes an external perspective of the test object to derive test cases. Taking into account the input configuration, the tester determines if the output is the correct, i.e. the software component must correctly process the input data and provides the expected output depending on the specific task that it performs. In this paper, the term software testing is referred to black-box testing.

During software testing each test case indicates a configuration. In this context, a software system is constituted by components that contain a set of k parameters. A parameter is defined as an element that serves as software input, receiving an unique value from a set of v possible values; therefore each software component has v^k possible configurations. A configuration indicates the setting values for each of the k parameters to execute a test case. It will be ideal to test all input cases; however the exhaustive approach is usually infeasible in terms of time and budget because if the number of parameters increases, the number of configuration grows exponentially. Due to this reason, another alternative to create an effective test suite has to be used.

The test suite reduction problem can be defined as the task of generating a set of test cases, as small as possible, under certain requirements that must be satisfied to provide the desired testing coverage of a system. A way to design test suites is through interaction testing (also called Combinatorial Testing), in which a matrix that involves all the possible combination of symbols that the factors of a system can take, under a certain interaction level.

Combinatorial Testing (CT) is an alternative that can be used for software testing which has been widely applied to construct different instances [5,11]. CT is based on empirical results of different kinds of software which indicate that all their identified failures were triggered by unexpected interactions among at most 6 parameters [14,15]. Based on this premise, CT implements a model-based testing approach using combinatorial objects where a covering array (CA) is one of the most used. A $\mathrm{CA}(N; t, k, v)$ is an $N \times k$ matrix that contains the test suite for the software under test. Every row indicates a test case and the columns represent the k parameters of the software, which can take v symbols. CAs offer a level of coverage t (strength), since every $N \times t$ sub-array includes, at least once, all the ordered subsets from v symbols of size t.

Even a CA significantly reduces the number of test cases, the problem to construct optimal CAs, also known in the literature as *Covering Array Construction*, is consider highly combinatorial [6].

Due to the complexity of the CAC, several techniques have been implemented although they do not necessarily provide the optimal number of rows. Among these strategies are: a) algebraic methods [13,7], b) recursive methods [9,8], c) greedy methods [10,3], d) metaheuristics [23,24] and e) exact methods [1]. A most detailed explanation of all these approaches can be found in recent surveys [16,17].

Among the techniques that provide the fastest results are greedy methods, being one of the best known the IPOG-F algorithm (In-Parameter-Order General). IPOG-F is the primary algorithm used in ACTS tool (Advanced Combinatorial Testing System). It was developed by the NIST, an agency of the United States Government that works to develop tests, test methodologies, and assurance methods; and which within its programs includes a research committee allocated to CT[1].

A post optimization process can be developed through the identification of symbols that are unnecessary in the test suite already constructed, those symbols (which will be named as wild cards), can be substituted by any other one without affecting the coverage of the matrix. In this paper we present a post-optimization process to reduce the size of CAs. This test suite reduction uses two algorithms refereed as `wcCA` and `FastRedu`. The algorithm `wcCA` detects wild cards (symbols that can be changed arbitrarily such that a CA does not lose its level of coverage). If the number of wild cards is greater than zero, `FastRedu` tries to merge compatible rows (two rows are compatible if for each column the two symbols involved are identical or one of them is a wild card) this action allows the reduction of rows from the original CA.

Test suite reduction has been widely studied by several researchers [2,21]; however, the proposed techniques are primary focused on test suites constructed by approaches different to CT, so the features of those test suites do not include the level of coverage of a CA which indicates that every v^t tuples appear in the $\binom{k}{t}$ combinations of parameters at least once. To our knowledge, the only other work that advocates to reduced the size of a CA using a post-optimization process is presented by Colbourn [20].

To test the performance of this approach, the algorithms were tested using 667 CAs created by the deterministic algorithm IPOG-F and obtained from the NIST website [2].

This paper is organized in the following Sections. Section 2 presents the definitions of CA and wild cards, Section 3 provides an explanation of one technique that has been used for post-optimization process of CAs. Section 4 gives an overview of the proposed algorithms `wcCA` and `FastRedu`. After that, Section 5 shows the design of the experiment and the results. Finally, Section 6 summarizes the main contribution of this paper.

[1] `http://csrc.nist.gov/groups/SNS/acts/index.html`
[2] `http://math.nist.gov/coveringarrays/`

2 Background of Covering Arrays

2.1 Definition of CA

A CA, denoted by CA($N; t, k, v$), is a matrix of size $N \times k$ and strength t where each column has v distinct symbols; and every $N \times t$ sub-array contains all combinations of v^t symbols at least once. When a CA is used for software testing, every column indicates the corresponding parameter of the software under testing and the symbols in the column specify the values for such parameter. Each row represents a test case, i.e. the configuration for an experimental run. A CA is optimal if it has the smallest possible number of rows, the value of N is known as the *Covering Array Number* and is formally defined as

$$\mathrm{CAN}(t, k, v) = min\{N | \exists \ \mathrm{CA}(N; t, k, v)\}$$

To illustrate the use of CAs suppose that we have a system with 3 parameters each with 2 possible values labeled as 0 and 1 respectively as shown in Table 1.

Table 1. System with 3 parameters each with 2 possible values

	O.S.	**Web browser**	**Database**
0 →	Linux	Mozilla Firefox	MySQL
1 →	Windows	Internet Explorer	Oracle

The exhaustive approach demands $2^3 = 8$ configurations, but instead of this, we can use a CA with $t = 2$, i.e. it covers all configurations between pairs of parameters, thus we only need 4 test cases as shown in Table 2. Every row indicates the configuration of a test case.

Table 2. Mapping of the CA(4;2,3,2) to the corresponding pair-wise test suite

	O.S.	**Web browser**	**Database**
1 1 0 →	Linux	Internet Explorer	MySQL
1 0 1 →	Linux	Mozilla Firefox	Oracle
0 1 1 →	Windows	Internet Explorer	Oracle
0 0 0 →	Windows	Mozilla Firefox	MySQL

In the example shown in Table 2 the total of combinations between pairs of parameters is $\binom{k}{t} = \binom{3}{2} = 3$ being these {O.S., Web browser}, {O.S., Database} and {Web browser, Database}. Each parameter has 2 settings giving 4 possible combinations for each pair of them. The definition of a CA implies that every

$N \times t$ sub-array contains all possible combinations of v^t symbols *at least once*, based on this fact, for the tuple {O.S., Web browser} all the combinations {0,0}, {0,1}, {1,0}, {1,1} (mapped to the corresponding settings) are covered, the same way for any pair of selected parameters.

2.2 Detection of Unnecessary Symbols (Wild Cards) in a CA

Within the definition of CA, the indication *at least once* means that a combination can be covered *more than once*, i.e. there is the possibility that some symbols be changed arbitrarily without the CA losses its degree of coverage. These symbols are referred as wild cards. They are exemplified in Table 3 using asterisks *.

Table 3. Detection of wild cards in the CA(5;2,3,2)

A	**B**	**C**		**A**	**B**	**C**
0	0	0		0	0	0
1	0	1		1	0	1
0	1	1	$\rightarrow$	0	1	1
1	0	0		*	*	*
1	1	0		1	1	0

The array on the right side is still a CA due to for all pairs of columns {A, B}, {A, C} {B, C} the combinations {0,0}, {0,1}, {1,0}, {1,1} are covered. In this example all symbols in the third row are wild cards, it means that the row can be deleted to obtain a new CA(4;2,3,2) which improves the size of the original.

Wild cards have different applications, one of them is the possibility to merge compatible rows in a CA (reducing its number of rows), in this way, many CAs can be constructed by strategies that demand less time than exact approaches with the possibility of reducing their size through a post-optimization strategy. Another alternative to use wild cards is shown in the work of Colbourn, *et al.* [9] which explains how use them to construct CAs with larger number of columns respect to the input ones through algebraic constructions.

3 Related Work

An algorithm to find wild cards was presented in Nayery *et al.* [20]. The main idea consists in searching elements of the initial array CA($N;t,k,v$) that do not affect coverage of tuples of cardinality t. These elements can be replaced by symbol *, after that, the row containing the greatest number of * is moved to the last place in the array. This number is stored. For all remaining rows containing *, look through all elements of row r where * occurs (let this element occur in column c). If the value in the last row of column c is *, replace element of row r in column c by a random value. Otherwise, replace it by the value in the last row of column c. 8. Permute all rows, except for the last one, in a random way to obtain a new initial array. The detailed explanation of this algorithm is presented in [20].

4 Proposed Approach

The methodology in this paper involves the use of two algorithms. The purpose of the first one is to find wild cards in a given CA using a greedy strategy. The second algorithm reduces the number of rows of the CA resulting of the first step. The next sections describe in detail both algorithms.

4.1 Wild Card Identification Algorithm (wcCA)

This section presents the Wild Card Identification Algorithm (or `wcCA`) for the identification of wild cards. The process to find wild card symbols in a $\mathrm{CA}(N; t, k, v)$ is simple and it is described in the following paragraphs.

The algorithm first determines the number of rows that covers the different combinations of symbols for each t-way interaction (a t-way interaction is a combination of t columns of the CA). Whenever a combination of symbols in a t-way interaction is solely covered by a row, the elements of the row involved in the corresponding columns are marked as fixed. Finally, it selects in a greedy way one covering row for those combination of symbols that are covered more than once; the result of the selection will produce that the chosen row will have fixed the columns for that combination of symbols. Those columns that weren't fixed during this process are wild cards of the input matrix. The Algorithm 2 presents the pseudocode for `wcCA`.

The input for `wcCA` is a matrix $\mathcal{M}_{N\times k}$ that is a $\mathrm{CA}(N; t, k, v)$. The output is a matrix $\mathcal{M}'_{N\times k}$ where each cell $m'_{i,j}$ is a wild card if it is assigned an `UNFIXED` value. The function `Fixing` identifies which combination of symbols are covered once. The structures $\mathcal{M}_C$ and $\mathcal{M}_R$ refer to the sets of columns and rows of $\mathcal{M}$, respectively. The variable M is a combination of t columns (*tuples*); it is used in combination with the structure $\texttt{solelyCovered}_{nc}$ to keep track of which combination of symbols nc are covered only once (the value of $\texttt{solelyCovered}_{nc}$ contains the row that covers nc). The function $\texttt{SymbolCombination}(\mathcal{M}, r, M)$ returns an integer value that identify which combination of symbols is located in the t columns identified by M in the row r. The function $\texttt{fixSymbols}(\mathcal{M}', r, M)$ set to the value `FIXED` the set of columns M in the row r. The function $\texttt{Fixed}(\mathcal{M}', r)$ returns the number of fixed columns. Finally, the structure $\texttt{coveredBy}_{nc}$ is filled with the result of the greedy criterion to untie combination of symbols and selects one row to cover them. In general, the algorithm `wcCA` can identify wild cards in time $O(N\binom{k}{t})$.

4.2 Reduction Algorithm Using Wild Cards (FastRedu)

The size of a CA can be reduced by merging compatible rows. Two rows are compatible if for each column they have the same symbol or at least one of the

Algorithm 1. Function Fixing, procedure prior to the wcCA Algorithm

```
Fixing(M_{N×k}, N, t, k, v)
Output: M'_{N×k}
M'_{N×k} ← UNFIXED;
solelyCovered_{v^t} ← ∅;
foreach {M | M ⊂ M_C, |M| = t} do
    foreach r ∈ M_R do
        v ← SymbolCombination(M, r, M);
        if solelyCovered_v = ∅ then
            solelyCovered_v ← r;
        end
        else
            solelyCovered_v ← NOT;
        end
    end
    foreach v ∈ V^t do
        if solelyCovered_v ≠ NOT then
            r ← solelyCovered_v;
            fixSymbols(M', r, M);
        end
    end
end
return M';
```

Algorithm 2. Algorithm for the identification of wild cards in a CA

```
wcCA(M_{N×k}, N, t, k, v)
Output: M'_{N×k}
M'_{N×k} ← Fixing(M, N, t, k, v);
coveredBy_{v^t} ← ∅;
foreach {M | M ⊂ M_C, |M| = t} do
    foreach r ∈ M_R do
        v ← SymbolCombination(M, r, M);
        r* ← coveredBy_v;
        if Fixed(M', r) > Fixed(M', r*) then
            coveredBy_v ← r;
        end
    end
    foreach v ∈ V^t do
        r ← coveredBy_v;
        fixSymbols(M', r, M);
    end
end
return M';
```

rows has a wild card. Example of compatible rows are shown in Table 4; here, a wild card is identified by an asterisk *. The example presents two rows that are compatible and the row resulting from merging both rows. The row resulting from the merging process will be the one with the greatest number of wild cards.

Algorithm 3 presents the pseudocode for the reduction algorithm (FastRedu). This algorithm is quite simple. It tests every combination of two rows (r_i, r_j) (lines 3 and 4), where $i < j$ and $r_i, r_j \in \mathcal{M}_R$, and verifies if they are compatible (line 5). Whenever a combination of rows (r_i, r_j) is compatible, they are merged in row r_j and the other row is marked as `unnecessary` in the CA (lines 6 and 7). All the rows marked as unnecessary will be deleted.

Table 4. Example of pairs of compatible rows

(a)	(b)	(c)	(d)
1 1 0 1	0 1 1 0 1	0 * * 0 1	* 1 1 0 1
0 1 1 0 1	0 1 1 * 1	0 1 1 0 1	* 1 1 0 1
0 1 1 0 1	0 1 1 * 1	0 * * 0 1	* 1 1 0 1

Algorithm 3. Algorithm to merge compatible rows in order to reduce the size of a CA

```
FastRedu(M_{N×k}, N, t, k, v)
Output: M'_{N×k}
M'_{N×k} ← wcCA(M, N, t, k, v);
M'_{N×k} ← markFixed(M, M');
for i = 1 to N do
    for j = i + 1 to N do
        if areCompatible(M'_{i,*}, M'_{j,*}) then
            mergeRows(M'_{i,*}, M'_{j,*});
            markRow(M'_{i,unnecesary});
        end
    end
end
return M';
```

The algorithm FastRedu runs in $O(N^2)$ time, where N is the number of rows of the CA.

5 Experimental Design

The wcCA and FastRedu algorithms were implemented in C language and compiled with gcc. The experiment was carried out in a CPU Intel Core 2 Duo at 1.5 GHz, 2 GB of RAM and Ubuntu 8.10 Intrepid Ibex Operating System. The CAs used in the experiment were generated by the deterministic algorithm IPOG-F [10] and obtained from the NIST webpage [3]. The alphabets v of the instances vary from 2 to 6, columns k from 3 to 32 and strengths t from 2 to 6 giving a whole of 667 CAs.

The experiment was conducted using the algorithms wcCA and FastRedu in the following post-optimization process: firstly, we use wcCA to find wild cards in the 667 CAs; after that, the resulted matrices from wcCA are given to FastRedu to merge compatible pairs of rows.

A summary of the main results obtained from the experiment is shown in Table 5. It can be seen more than 52% (347/667) of the input CAs reduced their size through the post-optimization process. Note the increasing trend on the percentage of improvements respect to the strength (t) of the CA; this suggests

[3] http://math.nist.gov/coveringarrays/ipof/ipof-results.html

Table 5. Improved cases over the total of input CAs after the post-optimization process

(a) Improved cases

v	t=2	t=3	t=4	t=5	t=6	**Total**
2	0/30	3/29	9/28	11/27	20/26	43/140
3	0/30	6/29	14/28	25/27	25/26	70/140
4	2/30	5/29	18/28	26/27	26/26	77/140
5	3/30	13/29	25/28	27/27	19/19	87/133
6	3/30	13/29	27/28	27/27	-	70/114
Total	8/150	40/145	93/140	116/135	90/97	**347/667**

(b) Percentage of improved CAs

v	t=2	t=3	t=4	t=5	t=6	**Total**
2	0	10.34	32.14	40.74	76.92	30.71
3	0	20.69	50.00	92.59	96.15	50.00
4	6.67	17.24	64.29	96.30	100	55.00
5	10.00	44.83	89.29	100	100	65.41
6	10.00	44.83	96.43	100	-	61.40
Total	5.33	27.59	66.43	85.93	92.78	**52.02**

Table 6. Minimum and maximum spent time (in sec.) for the post-optimization process

(a) Spent time by the algorithm `wcCA`

v \ t	**2**		**3**		**4**		**5**		**6**	
	min	max	min	max	min	max	min	max	min	max
2	0	0.02	0	0.03	0	0.5	0	8.91	0.01	109.89
3	0	0.02	0	0.08	0	2.9	0	79.81	0	1653.1
4	0	0.01	0	0.18	0	9.34	0.01	414.87	0.02	12783.4
5	0	0.03	0.01	0.41	0	25.18	0.01	1658.21	0.06	8615.71
6	0	0.03	0	0.65	0.01	68.64	0.03	4548.92	—	

(b) Spent time by the algorithm `FastRedu`

v \ t	**2**		**3**		**4**		**5**		**6**	
	min	max	min	max	min	max	min	max	min	max
2	0	0	0	0	0	0.01	0	0.01	0	0.03
3	0	0	0	0	0	0.01	0	0.08	0	1.08
4	0	0	0	0.01	0	0.07	0	1.54	0	67.53
5	0	0	0	0.01	0	0.29	0.02	19.18	0.04	384.66
6	0	0	0	0.03	0	1.2	0.03	157.89	—	

that the possibility of decreasing rows in CAs (constructed by IPOG-F) grows along with the value of t.

Table 7 shows an alternative analysis of the results derived from our experiment. In this new analysis we group the CAs by the number of their columns and their strength. Every group of t contains the different values of the alphabet for each CA. Every cell of the this Table shows the number of rows reduced in the corresponding CA. As seen in the last column, the number of improved cases is mostly concentrated in $k \leq 12$.

The results in Table 7 indicate an impressive reduction in the case CA(57;2,8,6), whose size was reduced by 15 rows. These cases are an example of the reduction of size for CAs that can be obtained through an algorithm to merge rows using wild cards (like `wcCA`, presented in this paper) using as input CAs constructed by the deterministic algorithm IPOG-F. The CA(42;2,8,6) which was obtained by this process is shown in Table 8.

Summarizing, the performance of `FastRedu` as a post-optimization algorithm to reduce CAs shows an improvement in the CA size when the value of v and/or the value of t increases, i.e. the greater the values of t or v are, the higher the possibility to reduce rows from a CA generated by IPOG-F.

Table 7. Number of reduced rows for each CA

	t=2					*t=3*					*t=4*					*t=5*					*t=6*				**Improved**
k \ v	**2**	**3**	**4**	**5**	**6**	**2**	**3**	**4**	**5**	**6**	**2**	**3**	**4**	**5**	**6**	**2**	**3**	**4**	**5**	**6**	**2**	**3**	**4**	**5**	**cases**
3	0	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0
4	0	0	0	1	0	0	0	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4
5	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	-	-	-	-	-	-	-	-	-	9
6	0	0	0	1	0	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1	-	-	-	-	16
7	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	16
8	0	0	0	0	15	1	0	0	1	1	1	2	1	1	1	1	3	1	1	1	1	1	1	2	18
9	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	2	16
10	0	0	0	0	0	0	1	1	1	1	1	2	1	1	5	1	1	1	1	1	1	1	2	3	18
11	0	0	0	0	0	0	0	0	0	0	0	2	1	5	1	0	1	1	4	1	1	2	2	5	12
12	0	0	0	1	0	0	1	1	1	1	0	1	2	3	1	1	3	4	2	1	1	5	1	1	18
13	0	0	0	0	1	0	0	0	0	0	0	2	3	1	1	0	1	1	1	1	6	2	6	1	13
14	0	0	0	0	0	0	0	0	0	0	1	1	5	1	1	1	5	1	2	1	1	1	1	3	14
15	0	0	0	0	1	0	0	0	0	1	0	1	2	1	12	0	1	4	1	1	1	1	2	1	14
16	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	0	6	2	1	1	1	2	3	1	12
17	0	0	0	0	0	0	0	0	0	0	0	0	3	1	1	0	1	1	1	3	1	3	4	1	11
18	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	4	1	1	1	7	2	3	13
19	0	0	0	0	0	0	0	0	1	1	0	1	1	2	1	0	2	2	3	1	1	4	5	2	14
20	0	0	0	0	0	0	0	0	0	1	0	0	0	2	6	1	3	6	5	1	1	8	2	1	12
21	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	2	1	1	0	7	5	1	11
22	0	0	0	0	0	0	0	0	0	1	0	0	0	2	1	0	1	2	6	1	1	3	3	1	11
23	0	0	0	0	0	0	0	0	2	0	0	0	0	0	1	0	3	3	1	1	0	3	15	1	9
24	0	0	1	0	0	0	0	0	0	1	0	0	1	1	2	0	1	1	2	1	0	2	3	1	12
25	0	0	1	0	0	1	0	0	1	0	0	0	0	3	1	1	2	1	4	2	1	1	1	1	14
26	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	1	1	2	0	6	0	5
27	0	0	0	0	0	0	0	0	1	0	0	0	0	2	1	0	1	3	1	1	1	3	2	0	10
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	3	10	13	0	1	1	0	6
29	0	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0	1	2	4	4	1	3	2	0	11
30	0	0	0	0	0	0	1	0	0	0	1	0	1	1	2	0	2	1	2	1	1	2	6	0	12
31	0	0	0	0	0	0	0	0	0	0	0	0	0	2	3	1	1	1	2	7	0	2	3	0	9
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	1	2	4	2	0	1	12	0	7
Improved cases	0	0	2	3	3	3	6	5	13	13	9	14	18	25	27	11	25	26	27	27	20	25	26	19	347

Table 8. The CA(42;2,8,6)^T that was created after the reduction of CA(57;2,8,6) using `wcCA` and `FastRedu`

*	*	*	*	*	*	0	2	2	1	1	1	5	2	4	0	4	4	0	0	3	2	4	5	5	5	0	3	5	2	4	4	5	3	0	1	1	2	3	3	3	1
*	*	*	*	*	*	0	5	4	4	3	2	4	2	2	5	0	1	1	2	4	1	3	0	2	3	3	3	1	0	4	5	5	1	4	0	5	3	0	5	2	1
5	0	3	2	4	1	0	1	5	2	1	4	0	0	3	3	5	1	4	2	3	2	0	2	1	4	5	2	3	4	4	2	5	0	1	3	0	3	1	4	5	5
5	4	0	2	3	1	0	4	2	1	0	4	3	1	5	1	1	4	2	3	4	0	2	4	2	1	4	5	5	5	0	3	0	1	5	2	5	3	3	2	0	3
5	4	3	0	2	1	0	0	4	3	4	5	5	2	0	5	4	2	1	4	0	5	5	1	3	0	3	2	4	3	1	3	2	3	2	2	1	1	5	4	1	0
5	4	3	2	1	0	0	5	0	4	2	0	2	3	4	2	1	2	4	5	1	1	3	3	1	5	1	0	0	2	5	0	4	5	3	5	1	4	4	3	2	3
5	4	3	2	0	1	0	0	3	0	5	2	1	5	1	4	2	3	5	3	5	1	0	5	4	3	1	4	0	4	4	5	2	2	2	1	3	2	3	1	0	4
0	4	3	2	5	1	0	3	5	0	5	3	3	2	5	5	3	0	3	1	1	4	1	5	0	4	2	3	2	1	2	4	1	5	4	4	2	0	2	0	4	1

6 Conclusions

We present a post-optimization strategy to reduce the size of a CA. The post-optimization process reduces the number of rows of a CA through the merging of rows. The strategy to merge rows is performed in two steps. The first one consists on identifying wild cards (symbols that can be changed arbitrarily such that a CA does not lose its level of coverage) with `wcCA` algorithm. The second step merges compatible pairs of rows through `FastRedu` algorithm; two rows are compatible if they share the same symbols in each of their columns or at least one of them is a wild card.

The algorithm to identify wild cards (`wcCA`) runs in $O(N\binom{k}{t})$ steps, where N is the size of the CA, and t is the strength. The algorithm to merge rows runs in time $O(N^2)$.

The post-optimization process was tested with 667 CAs constructed by the state-of-the-art algorithm IPOG-F. The results show a reduction in 52% of the instances. The CA(57;2,8,6) reduced its size by 15 rows, an impressive reduction if we consider that the new CA(42;2,8,6) is the best upper bound so far.

The improved cases were analyzed in terms of t, k, v. The improvement that can be achieved by the `FastRedu` algorithm increased with the strength t. An slightly small improvement can also be perceived when the alphabet v is increased. With respect to the number of columns k, the best improvements are concentrated in values of $k \leq 12$.

In conclusion, the quality of the CAs generated by IPOG-F can be improved significantly through our approach with a high probability when the values of t and v are large.

References

1. Bracho-Rios, J., Torres-Jimenez, J., Rodriguez-Tello, E.: A New Backtracking Algorithm for Constructing Binary Covering Arrays of Variable Strength. In: Aguirre, A.H., Borja, R.M., Garciá, C.A.R. (eds.) MICAI 2009. LNCS, vol. 5845, pp. 397–407. Springer, Heidelberg (2009)
2. Bryce, R.C., Colbourn, C.J.: Prioritized interaction testing for pair-wise coverage with seeding and constraints. Information and Software Technology 48(10), 960–970 (2006)
3. Bryce, R.C., Colbourn, C.J.: The density algorithm for pairwise interaction testing: Research articles. Software Testing, Verification and Reliability 17, 159–182 (2007)
4. Burnstein, I.: Practical software testing: a process-oriented approach. Springer Professional Computing (2003) ISBN: 0-387-95131-8
5. Changhai, N., Hareton, L.: A survey of combinatorial testing. ACM Computing Surveys (CSUR) 43, 11:1–11:29 (2011)
6. Colbourn, C.J.: Combinatorial aspects of covering arrays. Le Matematiche (Catania) 58, 121–167 (2004)
7. Colbourn, C.J.: Covering arrays from cyclotomy. Designs, Codes and Cryptography 55, 201–219 (2010)
8. Colbourn, C.J., Martirosyan, S., Trung, T., Walker II., R.A.: Roux-type constructions for covering arrays of strengths three and four. Designs, Codes and Cryptography 41, 33–57 (2006), doi:10.1007/s10623-006-0020-8
9. Colbourn, C.J., Martirosyan, S.S., Mullen, G.L., Shasha, D., Sherwood, G.B., Yucas, J.L.: Products of mixed covering arrays of strength two. Journal of Combinatorial Designs 14(2), 124–138 (2006)
10. Forbes, M., Lawrence, J., Lei, Y., Kacker, R.N., 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(5), 287–297 (2008)
11. Gonzalez-Hernandez, L., Rangel-Valdez, N., Torres-Jimenez, J.: Construction of mixed covering arrays of strengths 2 through 6 using a tabu search approach. In: Discrete Mathematics, Algorithms and Applications, pp. 1–20 (to appear, 2012)

12. Hartman, A., Ben, I.: Graph Theory, combinatorics and algorithms interdisciplinary applications, 4th edn. Springer, New York (2005) ISBN 13:9780387243474
13. Ji, L., Yin, J.: Constructions of new orthogonal arrays and covering arrays of strength three. Journal of Combinatorial Theory Series A 117, 236–247 (2010)
14. Kuhn, D.R., Reilly, M.J.: 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), SEW 2002, pp. 91–95. IEEE Computer Society, Washington, DC (2002)
15. Kuhn, D.R., Wallance, D.R., Gallo Jr., A.M.: Software fault interactions and implications for software testing. IEEE Transactions on Software Engineering 30, 418–421 (2004)
16. Kuliamin, V., Petukhov, A.: A survey of methods for constructing covering arrays. Programming and Computer Software 37, 121–146 (2011)
17. Lawrence, J., Kacker, R.N., Lei, Y., Kuhn, D.R., Forbes, M.: A survey of binary covering arrays. Electronic Journals of Combinatorics 18, 84 (2011)
18. Levenson, N.G., Turner, C.S.: An investigation of the therac-25 accidents. IEEE Computer 26, 18–41 (1993)
19. Lions, J.L.: Ariane 5, flight 501, report of the inquiry board. European Space Agency (July 1996)
20. Nayeri, P., Colbourn, C.J., Konjevod, G.: Randomized Postoptimization of Covering Arrays. In: Fiala, J., Kratochvíl, J., Miller, M. (eds.) IWOCA 2009. LNCS, vol. 5874, pp. 408–419. Springer, Heidelberg (2009)
21. Sampath, S., Bryce, R.C.: Improving the effectiveness of test suite reduction for user-session-based testing of web applications. Information and Software Technology 54(7), 724–738 (2012)
22. Tassey, G.: The economic impacts of inadequate infrastructure for software testing. Technical report, National Institute of Standards and Technology (May 2002)
23. Torres-Jimenez, J., Rangel-Valdez, N., Gonzalez-Hernandez, L.: An exact approach to maximize the number of wild cards in a covering array. In: Submitted in Mexican International Conference on Artificial Intelligence, Puebla, México, November 26-December 4 (2011)
24. Torres-Jimenez, J., Rodriguez-Tello, E.: Simulated annealing for constructing binary covering arrays of variable strength. In: IEEE Congress on Evolutionary Computation CEC 2010, July 18-23, pp. 1–8 (2010)