Metaheuristic Post-Optimization of the NIST Repository of Covering Arrays

Jose Torres-Jimenez[1], Arturo Rodriguez-Cristerna[1]
[1]*CINVESTAV-Tamaulipas, Information Technology Laboratory,*
Km. 5.5 Carretera Cd., Victoria, Tamaulipas, Mexico
jtj@cinvestav.mx arodriguez@tamps.cinvestav.mx

Abstract. Construction of Covering Arrays (CA) with minimum possible number of rows is challenging. Often the available CA have redundant combinatorial interaction that could be removed to reduce the number of rows. This paper addresses the problem of removing redundancy of CA using a metaheuristic post-optimization (MPO) approach. Our approach consists of three main components: a redundancy detector (RD); a row reducer (RR) ; and a missing-combinations reducer (MCR). The MCR is a metaheuristic component implemented using a simulated annealing algorithm. MPO was instantiated with $21,964$ CA taken from the National Institute of Standards and Technology (NIST) repository. It is a remarkable result that this instantiation of MPO has delivered 349 new upper bounds for these CA.

Key words. Covering Arrays ; NIST Repository of Covering Arrays ; Metaheuristic Post-processing Algorithms

1 Introduction

The use of software has permeated many areas of human activity, so the reliability of software has become important worldwide. It is estimated that software testing consumes about 50% of the cost of developing a new piece of software. A 2002 NIST report [23] indicates that the cost of an inadequate infrastructure for software testing was in the range of \$22.2 to \$59.5 billion (US dollars). Reducing this cost is not only important but the design and implementation of adequate software testing procedures is critical for the reliability of many electronic and mechanical systems, even more so in complex and important systems, such as space shuttles [16].

According to Myers *et al.* [17] functional software testing methods may be divided into two main categories: white-box testing and black-box testing. The design of white-box testing suites requires source code of the software under examination. Some testing strategies based on the white-box approach are: statement coverage, decision coverage, condition coverage, decision-condition coverage and multiple-condition coverage. The building of test suites using white-box strategies is more challenging than for black-box strategies, since white-box strategies are based on knowledge of the internal structure of the system. Furthermore, if the system is modified, then tests must be redesigned to satisfy the new version of the system. On the other hand, the design of black box testing suites does not require source code of the software under examination. It compares actual behaviour against expected behaviour based on the functionality and the specification of the software system under examination. Some black-box testing strategies are: exhaustive input testing, equivalence partitioning, boundary-value analysis, cause-effect graphing, error guessing, and *combinatorial interaction testing*.

It is easy to construct test suites using a random black-box approach, but they rarely cover a large percentage of the functionality of the system under examination. A black-box approach that covers 100 percent of the functionality is the exhaustive approach, but it is impractical in most cases because too many tests are required. As an example: if we need to design a test suite for a system that has 20 parameters and each parameter has 10 possible values, it would require 10^{20} tests; however, using a combinatorial interaction testing approach that covers the combinations of all pairs of parameter values, the test suite will require only 155 tests. The number of tests required with combinatorial interaction testing grows logarithmically according to the number of parameters [11]. Empirical studies in software testing have shown that combinatorial interaction testing is a useful approach [14, 4]. The mathematical objects that support combinatorial interaction testing are Covering Arrays (CA) and Mixed Covering Arrays (MCA).

CA and MCA are combinatorial structures that have been used successfully in various areas. The most reported application of CA and MCA is in the design of test suites for software combinatorial interaction testing [7, 8], which is based on the concept that software faults are caused by unexpected combinatorial interactions of certain size between components. Another ap-

0	1	2	2	0	1	1	2	0
0	1	2	1	2	0	2	0	1
0	1	2	0	1	2	0	1	2
0	0	0	1	1	1	2	2	2

Figure 1: Transposed matrix of a $CA(9; 2, 4, 3)$.

0	1	1	0	0	1
0	1	0	1	1	0
0	1	0	1	0	1
0	0	1	1	2	2

Figure 2: Transposed matrix of an $MCA(6; 2, 4, 3^1 2^3)$.

plication is found in the field of parameter fine-tuning of metaheuristic algorithms [12, 19, 22, 20].

A CA, denoted by CA(N; t, k, v), is an N $\times$ k array, where every entry of the array takes values from a set of symbols of size v, such that every N $\times$ t sub-array contains at least once all possible v^t t-tuples of symbols. An MCA is a generalization of a CA where the alphabets of the columns could have different cardinalities. The test cases are represented by the rows, the parameters are represented by the columns, the parameter values are taken from the set $\{0, 1 \ldots, \text{v}-1\}$ which is called the alphabet, and t is the strength or combinatorial interaction degree between parameters covered by the CA. Figure 1 shows an example of a CA(9;2,4,3), and an MCA(6;2,4,$3^1 2^3$) is shown in figure 2.

The Covering Array Number (CAN) is the minimum N such that for fixed k, v, and t a CA exists. The CAN is denoted by CAN(t,k,v). The construction of CA with N=CAN(t,k,v) is a challenging problem whether we use mathematical structures or metaheuristic algorithms.

When we have non-optimal CA (i.e. a CA with N > CAN(t,k,v)), it usually has many t-tuples that are covered more than once. This fact presents the opportunity to reduce number of rows of CA, given that it may then be possible to identify redundant rows [18] that can be removed.

In this paper we introduce a Metaheuristic Post-Optimization (MPO) approach to reduce the size of a CA by exploiting redundant elements in CA.

MPO is composed of three main components: a) a redundancy detector (RD); a row reducer (RR); and a missing-combination reducer (MCR) implemented using a simulated annealing algorithm (the metaheuristic component of our approach). MPO was extensively tested using $21,964$ CA (taken from the CA NIST repository). We have improved almost all $21,964$ CA, but the most remarkable result is that MPO has set 349 new upper bounds for these CA.

The remaining of the paper is structured in three sections. In section 2 we present in detail MPO approach giving details of the redundancy detector, row reducer and missing-combination reducer components. In section 3 we present the results of instantiating the MPO with the whole National Institute of Standards and Technology repository of covering arrays. Finally in section 4 we present some conclusions.

2 Metaheuristic Post-Optimization (MPO)

In this section we present implementation details of the MPO approach. We firstly give an overview of the whole process of MPO, secondly, we present details of each of the components RD, RR, MCR.

2.1 Design of MPO Approach

The design and implementation of MPO approach is briefly described in algorithm 1, where it can be observed that it has three components and two main loops. The inner loop executes the components: *Redundancy Detector* (RD) and *Row Reducer* (RR). After the inner loop is executed, the *Missing-Combinations Reducer* (MCR) runs. When the MCR (implemented with a simulated annealing (SA) algorithm) is not able to make the number of missing combinations equal to zero, MPO ends.

MPO (algorithm 1) receives as input $\mathcal{A}$ =CA(N;t,k,v) and gives as output $\mathcal{B}$=CA(N'; t, k, v) with N' $\leq$ N and no missing t-wise combinations. The function τ computes the number of missing t-wise combinations of the parameter passed to it. τ has temporal complexity $O(N\binom{k}{t})$ (a more detailed description of how to compute the missing interactions for CA was presented by Avila-George et al. [3]).

Algorithm 1 Metaheuristic Post-Optimization (MPO) algorithm.

input : $\mathcal{A} = CA(N; t, k, v)$
output: $\mathcal{B} = CA(N'; t, k, v) | N' \leq N$
begin
  $\mathcal{B}' \leftarrow \mathcal{A}$
  repeat
    repeat
      $\mathcal{B}' \leftarrow$ *Redundancy Detector* $(\mathcal{B}')$; **if** $(\mathcal{B}' = 0)$ **then** $\mathcal{B} \leftarrow \mathcal{B}'$
      $\mathcal{B}' \leftarrow$ *Row Reducer*$(\mathcal{B}')$; **if** $(\mathcal{B}' = 0)$ **then** $\mathcal{B} \leftarrow \mathcal{B}'$
    until $\tau(\mathcal{B}') > 0$;
    $\mathcal{B}' \leftarrow$ *Simulated Annealing* $(\mathcal{B}')$; **if** $(\mathcal{B}' = 0)$ **then** $\mathcal{B} \leftarrow \mathcal{B}'$
  until $\tau(\mathcal{B}') > 0$;
  return $\mathcal{B}$
end

2.2 Redundancy Detector (RR)

The goal of the Redundancy Detector (RD) algorithm is to find a large number of redundant entries in the CA given as input. RD does its job by doing three scans of the input, the first two scans visiting all t-wise combinations of the matrix (each scan with temporal complexity $O(N\binom{k}{t})$), the third scan visiting all elements of the matrix, searching for rows that are totally redundant (with temporal complexity $O(Nk)$). The total temporal complexity is $O(2N\binom{k}{t} + Nk)$.

The purpose of the first scan is to set as 'Fixed Symbol' (FS) cells that participate in t-wise combinations covered only once, and as 'Possible Redundant Cell' (PRC) all other cells. The second scan works with cells marked as PRC and decides which cells transform to the status of FS, while making sure coverage property (all t-wise combinations must be covered at least once) is satisfied, and number of cells with status of PRC is maximized. The third scan removes all rows in which all elements have status of PRC.

2.3 Row Reducer (RR)

The Row Reducer algorithm receives as input a CA and works in a greedy manner searching for a row i such that its removal minimizes missing combinations. In the worst case RR tests all rows of CA, but when a row with no missing t-wise combinations is found RR ends. If this is not the case the row removed is the one that gives the fewest number of missing t-wise combinations.

The logic of the operation of the RR algorithm is simple: replace the FS cells of the i-th row in all PRC cells of remaining rows, and then verify number of missing t-wise combinations (after removal of row i). RR removes the first row that minimizes missing t-wise combinations. The worst case temporal complexity of the algorithm is $O(N(N-1)\binom{k}{t})$ for the determination of row to be removed, and $O(Nk)$ for the removal of the row. Then the total temporal complexity of the RR algorithm is $O(N(N-1)\binom{k}{t} + Nk)$.

2.4 Missing-Combinations Reducer (MCR)

The MCR component of MPO is in charge of reducing to zero the number of missing t-tuples of the input parameter (a matrix with missing combinations). We decided to implement MCR using a simulated annealing (SA) algorithm given that SA has been applied succesfully for solving related problems [2, 6, 24, 21]. The core elements of the SA are: the neighbourhood functions $\mathcal{NF}$; and the cooling schedule.

We used two neighbourhood functions $\mathcal{NF}_1$ and $\mathcal{NF}_2$. $\mathcal{NF}_1$ searches for a random missing t-tuple and sets one such tuple in every row, selecting the row that gives the fewest number of missing t-wise combinations. $\mathcal{NF}_2$ selects t cells in a row (cells and rows are selected randomly) then tests every v^t possible t-tuple in those cells, and selects combination that gives the lower number of missing t-wise combinations. SA uses a mixture of the two neighbourhood functions in such a way that $\mathcal{NF}_1$ is applied with a probability pr and consequently $\mathcal{NF}_2$ is applied with a probability $1 - pr$.

The cooling schedule configuration in SA involves [1, 15]: (a) an initial temperature ($temp_0$); (b) a decreasing function to reduce the temperature value; (c) an ending temperature ($temp_f$); and (d) a finite number of iterations of the local search at the same temperature (L) (L size of a Markov chain [5]). The parameters of the cooling schedule control the behaviour of the algorithm and therefore affect drastically the quality of the final solution. We selected static geometric cooling schedule controlled by a parameter α. The parameter L is static during execution of the algorithm [2, 24]. Also a parameter called frozen factor (ff) was added to control number of temperature reductions without improvement towards solution, which works as an alternative termination criterion that is triggered when search stagnates.

SA algorithm (algorithm 2) is based on definition given by Kirkpatrick et al. [13]. Parameter values were selected after a parameter fine-tuning, and they are: $temp_0 = 1$; $\alpha = 0.99$; L=Nkv^2; $pr = 0.5$; $temp_f = 1 \times 10^{-14}$; and $FE = 11$.

Algorithm 2 Simulated Annealing algorithm.

input : $\mathcal{A}_C A(N; t, k, v), pr, temp_0, temp_f, L, \alpha, FE$
output: $\mathcal{A}'' \quad | \quad \tau(\mathcal{A}'') \leq \tau(\mathcal{A})$
begin
    $\mathcal{A}'' \leftarrow \mathcal{A}$
    $temp \leftarrow temp_0$
    while $temp > temp_f$ **do**
        for $i \leftarrow 0$ **to** $L - 1$ **do**
            if pr **then** $\mathcal{A}' \leftarrow \mathcal{NF}_1(\mathcal{A})$
            else $\mathcal{A}' \leftarrow \mathcal{NF}_2(\mathcal{A})$
            $\mathcal{A} \leftarrow \mathcal{A}'$ with a probability $\min\{1, e^{-\frac{\tau(\mathcal{A}', \dots) - \tau(\mathcal{A}, \dots)}{temp}}\}$
            if $\tau(\mathcal{A}, \dots) < \tau(\mathcal{A}'', \dots)$ **then**
                $\mathcal{A}'' \leftarrow \mathcal{A}$
                if $\tau(\mathcal{A}'', \dots) = 0$ **then return** $\mathcal{A}''$
            end
        end
        $temp \leftarrow temp \cdot \alpha$
        if *there was an improvement in* $\mathcal{A}''$ **then** $CE \leftarrow 1$ **else** $CE{+}{+}$
        if $CE == ff$ **then return** $\mathcal{A}''$
    end
    return $\mathcal{A}''$
end

2.5 Implementation Note

The proposed algorithms were coded using C language and compiled with GCC 4.3.5 with -O3 optimization flag, and run in cores of the type AMD® 8435 (2.6 Ghz).

3 Results

To measure the effectiveness of MPO, the NIST repository of CA [10] was processed. NIST repository of CA consists of 21,964 CA with $v \in \{2, \dots, 6\}$ and $t \in \{2, \dots, 6\}$. For each instance we report: average percentage reduction of rows (Υ), average time in minutes (Γ), and number of instances (I).

Table 1: Results of MPO algorithm after processing entire repository [10]. Information is organized in triplets containing: average percentage reduction of rows (Υ); average time in minutes (Γ); and number of instances (I) (Continues in table 2).

$v \backslash t$	2			3			4			5		
	Υ	Γ (m)	I	Υ	Γ (m)	I	Υ	Γ (m)	I	Υ	Γ (m)	I
2	16.21	3.27	1998	2.95	670.96	1997	1.17	19809.28	90	2.09	6650.83	186
3	6.16	9.25	1998	1.42	1753.65	1997	1.06	21141.30	496	2.35	10851.70	111
4	5.20	9.01	1998	0.57	2140.89	1997	0.89	8161.35	304	2.47	16584.87	76
5	3.82	23.95	1998	0.35	2280.55	1968	0.99	9531.80	204	2.60	11895.09	56
6	3.37	64.55	1998	0.37	5097.92	1303	1.07	13944.75	159	2.93	12399.77	41
avg	6.95	20.47	1998	1.19	2236.92	1852.4	1.08	21384.18	250.6	2.35	16769.35	94

Table 2: (Continued from table 1) Results of MPO algorithm after processing entire repository [10]. Information is organized in triplets containing: average percentage reduction of rows (Υ); average time in minutes (Γ); and number of instances (I).

$v \backslash t$	6		
	Υ	Γ (m)	I
2	3.83	4045.90	80
3	4.08	8526.52	45
4	4.20	8715.07	30
5	5.16	9690.55	19
6	-	-	-
avg	4.10	11647.32	43.5

Table 1 and table 2 summarize the results of processing NIST repository [10]. Information is organized in triplets containing: Υ, Γ, and I. It is shown that many instances were optimized, resulting in the construction of 349 state of the art upper bounds for CA by using MPO algorithm. The comparison between the MPO new upper bounds and the IPOG-F bounds is shown in tables 3, 4 and 5. Results are shown in figures 3, 4, 5, 6, and 7 where instances are grouped by combinatorial interaction coverage degree (t).

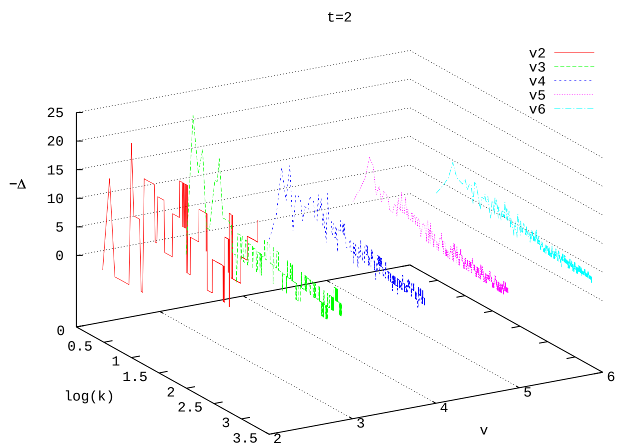

Figure 3: Results of MPO after processing instances of repository [10] with $t = 2$. ($-\Delta = N - N'$)

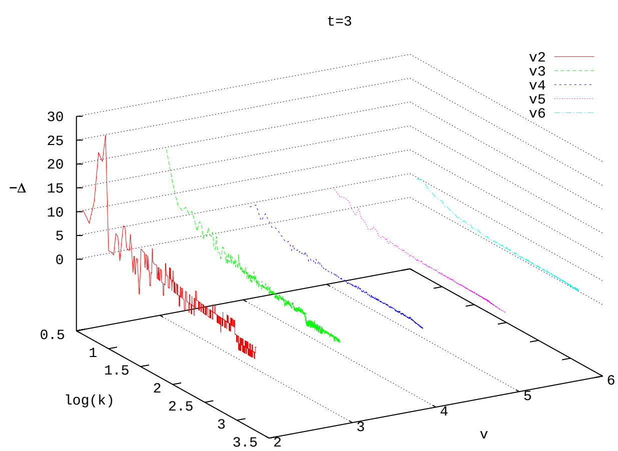

Figure 4: Results of MPO after processing instances of repository [10] with $t = 3$. ($-\Delta = N - N'$)

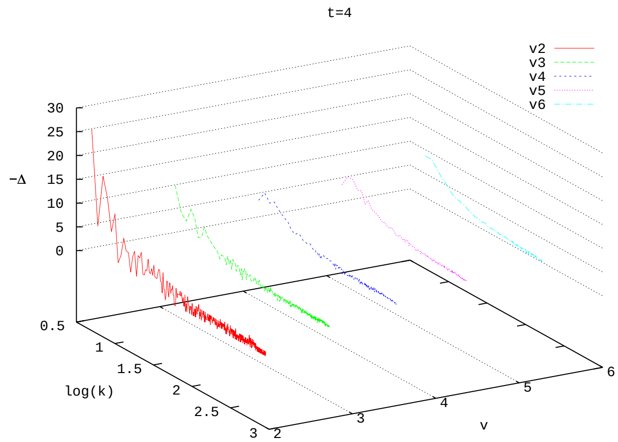

Figure 5: Results of MPO after processing instances of repository [10] with $t = 4$. $(-\Delta = N - N')$

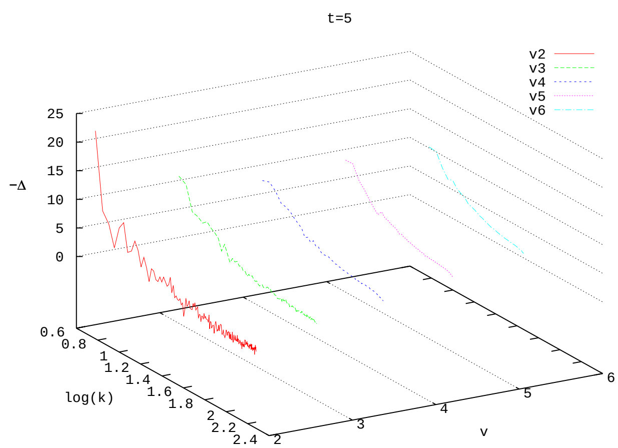

Figure 6: Results of MPO after processing instances of repository [10] with $t = 5$. $(-\Delta = N - N')$

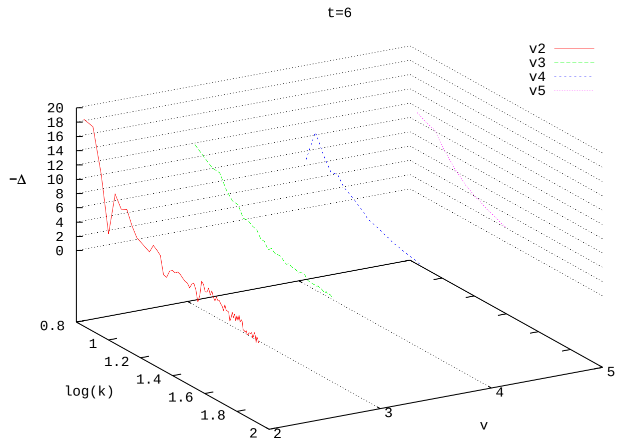

Figure 7: Results of MPO after processing instances of repository [10] with $t = 6$. ($-\Delta = N - N'$)

4 Conclusions

In this paper we have presented a Metaheuristic Post-Optimization (MPO) approach to reduce the cardinality of Covering Arrays. MPO was implemented using three components: a redundancy detector, a row reducer, and a missing-combination reducer. The redundancy detector has the mission of detecting elements in CA that could be changed without affecting degree of coverage of CA. The row reducer takes advantage of redundant elements of CA to reduce number of rows. When the removal of a row produces missing combinations, then control is given to the missing-combination reducer. The missing-combination reducer is implemented with simulated annealing (the metaheuristic component of MPO) and tries to make the number of missing combinations equal to zero. Even though all three components are key to the success of MPO, we believe metaheuristic component is the most important part of MPO, given that through its use it is possible to reduce iteratively number of rows of CA.

We have conducted big-scale experimentation through instantiation of MPO with the whole NIST repository of CA. NIST repository consists of

Table 3: New best upper bounds constructed with MPO algorithm. Part 1 of 3.

id	k	*v3t4* IPOG-F [9]	MPO
1	315	968	964
2	316	969	964
3	317	969	963
4	318	971	965
5	319	971	963
6	320	971	963
7	321	974	965
8	322	974	966
9	323	975	970
10	324	976	965
11	325	976	970
12	326	976	974
13	327	977	973
14	328	978	969
15	329	979	975
16	330	979	976
17	331	981	977
18	332	981	973
19	333	983	975
20	335	983	979
21	336	983	978
22	337	984	977
23	338	985	977
24	339	986	977
25	340	987	978
26	341	987	979
27	342	987	981
28	343	990	985
29	345	991	984
30	346	992	985
31	347	992	985
32	348	992	987
33	349	992	986
34	351	993	991
35	353	994	987
36	360	999	994
37	361	1001	995
38	362	1001	996
39	364	1002	997
40	368	1007	1002
41	370	1008	1001
42	373	1009	1003
43	375	1009	1005
44	377	1011	1003
45	379	1013	1007
46	386	1017	1009
47	405	1027	1024
48	439	1046	1045
49	447	1052	1050

id	k	*v6t4* IPOG-F [9]	MPO
50	49	9323	9212
51	50	9393	9294
52	51	9466	9397
53	52	9550	9463
54	53	9623	9540
55	55	9762	9673
56	56	9828	9742
57	57	9900	9813
58	58	9964	9869
59	59	10032	9948
60	60	10097	10013
61	61	10163	10067
62	62	10219	10142
63	63	10282	10198
64	64	10347	10250
65	65	10398	10328
66	66	10463	10368
67	67	10520	10441
68	68	10578	10478
69	69	10633	10572
70	70	10693	10599
71	71	10745	10676
72	72	10798	10744
73	73	10850	10758
74	74	10909	10821
75	75	10958	10882
76	76	11012	10959
77	77	11057	10992
78	78	11110	11017
79	79	11158	11100
80	80	11203	11121
81	81	11253	11187
82	82	11303	11219
83	83	11353	11282
84	84	11397	11319
85	85	11441	11384
86	86	11484	11407
87	87	11533	11478
88	88	11577	11504
89	89	11625	11581
90	90	11666	11591
91	91	11710	11630
92	92	11753	11671
93	93	11790	11729
94	94	11833	11762
95	95	11874	11800
96	96	11913	11884
97	97	11956	11918
98	98	11997	11924
99	99	12038	11949
100	100	12085	12003
101	101	12120	12036
102	102	12148	12140
103	103	12194	12120
104	104	12231	12185
105	105	12267	12196
106	106	12306	12240
107	107	12343	12268
108	109	12408	12388
109	110	12445	12380
110	112	12517	12449
111	116	12651	12593
112	118	12716	12698
113	120	12785	12734
114	121	12816	12757
115	128	13036	13007
116	129	13062	13056
117	133	13192	13146
118	149	13634	13606

id	k	*v3t5* IPOG-F [9]	MPO
119	35	1867	1826
120	36	1895	1850
121	37	1920	1882
122	38	1947	1909
123	39	1974	1933
124	40	1997	1949
125	41	2023	1975

Table 4: New best upper bounds constructed with MPO algorithm. Part 2 of 3.

v3t5			
id	k	IPOG-F [9]	MPO
126	42	2046	2002
127	43	2070	2022
128	44	2091	2050
129	45	2112	2071
130	46	2130	2086
131	47	2150	2112
132	48	2174	2134
133	49	2191	2154
134	50	2213	2182
135	51	2231	2198
136	52	2251	2217
137	53	2269	2232
138	54	2289	2246
139	55	2309	2277
140	56	2327	2281
141	57	2342	2304
142	58	2358	2316
143	59	2373	2335
144	60	2394	2347
145	61	2408	2370
146	62	2425	2387
147	63	2440	2397
148	64	2451	2413
149	65	2468	2439
150	66	2482	2447
151	67	2498	2459
152	68	2512	2477
153	69	2527	2487
154	70	2542	2504
155	71	2555	2518
156	72	2573	2531
157	73	2584	2546
158	74	2597	2564
159	75	2609	2588
160	76	2625	2593
161	77	2639	2608
162	78	2648	2615
163	79	2661	2625
164	80	2673	2635
165	81	2686	2651
166	82	2700	2668
167	83	2710	2677
168	84	2725	2684

v3t5			
id	k	IPOG-F [9]	MPO
169	85	2739	2700
170	86	2749	2706
171	87	2762	2728
172	88	2770	2736
173	89	2783	2747
174	90	2792	2761
175	91	2805	2772
176	92	2815	2783
177	93	2825	2798
178	94	2836	2797
179	95	2847	2813
180	96	2857	2819
181	97	2868	2834
182	98	2877	2838
183	99	2885	2854
184	100	2895	2862
185	101	2909	2871
186	102	2920	2878
187	103	2928	2898
188	104	2938	2901
189	105	2945	2908
190	107	2962	2928

v4t5			
id	k	IPOG-F [9]	MPO
191	26	6957	6775
192	27	7116	6937
193	28	7267	7088
194	29	7414	7247
195	30	7555	7379
196	31	7691	7527
197	32	7816	7649
198	33	7939	7782
199	34	8064	7907
200	35	8183	8023
201	36	8301	8137
202	37	8420	8259
203	38	8530	8376
204	39	8629	8477
205	40	8737	8583
206	41	8847	8693
207	42	8945	8791
208	43	9035	8890
209	44	9137	8979

v4t5			
id	k	IPOG-F [9]	MPO
210	45	9227	9104
211	46	9320	9176
212	47	9406	9262
213	48	9501	9357
214	49	9588	9453
215	50	9673	9520
216	51	9755	9621
217	52	9835	9682
218	53	9922	9769
219	54	9998	9849
220	55	10079	9927
221	56	10155	10007
222	57	10232	10082
223	59	10379	10231
224	60	10454	10302
225	62	10590	10442
226	63	10650	10509

v5t5			
id	k	IPOG-F [9]	MPO
227	21	18779	18260
228	22	114775	111818
229	23	118587	115802
230	24	122201	119500
231	25	125683	123108

v6t5			
id	k	IPOG-F [9]	MPO
232	17	40334	38976
233	18	42102	40820
234	19	43833	42554
235	20	45425	44224
236	21	46970	45784
237	22	48479	47352
238	23	49924	48838
239	24	51287	50180
240	25	52604	51505
241	26	53850	52814
242	27	55069	54032
243	28	56225	55275
244	29	57363	56353
245	30	58468	57503
246	31	59529	58576
247	32	60570	59612
248	33	61562	60608

Table 5: New best upper bounds constructed with MPO algorithm. Part 3 of 3.

$v6t5$			
id	k	IPOG-F [9]	MPO
249	34	62527	61612
250	35	63471	62557
251	36	64399	63519

$v2t6$			
id	k	IPOG-F [9]	MPO
252	41	572	547
253	42	579	550
254	43	590	565
255	44	594	565
256	45	603	578
257	46	611	588
258	47	617	590
259	48	625	600
260	49	630	604
261	50	636	612
262	51	643	620
263	52	650	630
264	53	656	630
265	54	662	640
266	55	667	645
267	56	672	650
268	57	677	663
269	58	683	665
270	59	689	665
271	60	695	675
272	61	699	675
273	62	703	685
274	63	709	685
275	64	715	695
276	65	721	695
277	66	725	705
278	67	728	705
279	68	732	710
280	69	738	724
281	70	743	729
282	71	747	734
283	72	751	736
284	73	755	743
285	74	761	749
286	75	766	751
287	76	770	755
288	77	773	758
289	78	777	760

$v2t6$			
id	k	IPOG-F [9]	MPO
290	79	782	770
291	80	785	771
292	81	791	772
293	82	794	778
294	83	796	787
295	84	800	784
296	85	804	790
297	86	809	799

$v3t6$			
id	k	IPOG-F [9]	MPO
298	26	5709	5544
299	27	5853	5667
300	28	6003	5827
301	29	6150	5969
302	30	6281	6103
303	31	6413	6245
304	32	6535	6348
305	33	6656	6461
306	34	6772	6583
307	35	6877	6715
308	36	6989	6832
309	37	7092	6932
310	38	7194	7036
311	39	7293	7131
312	40	7391	7233
313	41	7490	7315
314	42	7574	7411
315	43	7672	7506
316	44	7757	7600
317	45	7845	7702
318	46	7938	7766
319	47	8013	7856
320	50	8256	8108
321	51	8333	8179

$v4t6$			
id	k	IPOG-F [9]	MPO
322	19	26392	25430
323	20	27534	26564
324	21	28625	27676
325	22	29640	28735
326	23	30636	29720
327	24	31591	30724
328	25	32501	31654

$v4t6$			
id	k	IPOG-F [9]	MPO
329	26	33369	32513
330	27	34187	33356
331	28	35006	34198
332	29	35791	34971
333	30	36570	35778
334	31	37305	36515
335	32	38015	37255

$v5t6$			
id	k	IPOG-F [9]	MPO
336	11	56615	52471
337	12	63620	59622
338	13	70190	66275
339	14	76390	72680
340	15	82139	78480
341	16	87559	84102
342	18	97605	94263
343	19	102208	98994
344	20	106642	103514
345	21	110842	107773
346	22	114775	111818
347	23	118587	115802
348	24	122201	119500
349	25	125683	123108

$21,964$ CA, and while improving almost all CA in repository, the most remarkable result is that we have set 349 new upper bounds for these CA.

Acknowledgements

The authors acknowledge "Xiuhcoatl"-CGSTIC of CINVESTAV and ABACUS-CINVESTAV, CONACYT grant EDOMEX-2011-COI-165873, for providing access to high performance computing. This research was partially funded by the project: CONACyT 238469 - Métodos Exactos para Construir Covering Arrays. Part of this paper was written during first author's research stay at NIST in 2014-2015.

Disclaimer

Any mention of commercial products in this paper is for information only; it does not imply recommendation or endorsement by NIST.

References

[1] E.H.L. Aarts and J.K. Lenstra. *Local search in combinatorial optimization.* Princeton Univ Pr, 2003. ISBN: 0691115222.

[2] H. Avila-George, J. Torres-Jimenez, V. Hernández, and L. Gonzalez-Hernandez. Simulated annealing for constructing mixed covering arrays. In *Proceedings of the 9th International Symposium on Distributed Computing and Artificial Intelligence - DCAI*, volume 151 of *Advances in Intelligent and Soft Computing*, pages 657–664, Salamanca, Spain, from 28th to 30th March, 2012. Springer Berlin / Heidelberg.

[3] Himmer Avila-George, Jose Torres-Jimenez, Loreto Gonzalez-Hernandez, and Hernndez Vicente. Metaheuristic approach for constructing functional test-suites. *IET Software*, 7(2):104 – 117, 2013.

[4] K.Z. Bell. *Optimizing effectiveness and efficiency of software testing: A hybrid approach.* PhD thesis, North Carolina State University, 2006.

[5] D. Bertsimas and J. Tsitsiklis. Simulated annealing. *Statistical Science*, 8(1):10–15, 1993.

[6] M.B. Cohen, C.J. Colbourn, and A.C.H. Ling. Augmenting simulated annealing to build interaction test suites. In *Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on*, pages 394 – 405, November 2003.

[7] M.B. Cohen, P.B. Gibbons, W.B. Mugridge, and C.J. Colbourn. Constructing test suites for interaction testing. In *Proceedings of the 25th International Conference on Software Engineering*, pages 38 – 48, Portland, Oregon, USA, May 2003.

[8] Myra B Cohen. *Designing test suites for software interaction testing*. PhD thesis, University of Auckland, 2004.

[9] M. Forbes, J. Lawrence, Y. Lei, R.N. Kacker, and D.R. Kuhn. 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.

[10] M. Forbes, J. Lawrence, Y. Lei, R.N. Kacker, and D.R. Kuhn. Repository of CAs. *last access September*, 2014.

[11] A. Hedayat, N.J.A. Sloane, and J. Stufken. *Orthogonal arrays: theory and applications*. Springer Verlag, 1999. ISBN: 0-387-98766-5.

[12] A. Jose-Garcia, H. Romero-Monsivais, C. Hernandez-Morales, A. Rodriguez-Cristerna, I. Rivera-Islas, and J. Torres-Jimenez. A simulated annealing algorithm for the problem of minimal addition chains. In *Progress in Artificial Intelligence*, volume 7026 of *Lecture Notes in Computer Science*, pages 311–325. Springer Berlin / Heidelberg, 2011.

[13] S. Kirkpatrick, C.D. Gelatt Jr, and M.P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[14] D.R. Kuhn, D.R. Wallace, and A.M. Gallo Jr. Software fault interactions and implications for software testing. *IEEE Transactions on Software Engineering*, 30(6):418–421, 2004.

[15] J Lam. *An efficient simulated annealing schedule*. PhD thesis, Yale University New Haven CT, 1988.

[16] J.L. Lions and et al. Ariane 5 flight 501 failure, accesed 2014-09. Technical report, http://www.di.unito.it/ damiani/ariane5rep.htm, 09 2014.

[17] G.J. Myers, C. Sandler, T. Badgett, and T.M. Thomas. *The art of software testing*. John Wiley & Sons Inc, 2 edition, 2004. ISBN: 0471469122.

[18] P. Nayeri, C. Colbourn, and G. Konjevod. Randomized postoptimization of covering arrays. In *Combinatorial Algorithms*, volume 5874 of *Lecture Notes in Computer Science*, pages 408–419. Springer Berlin / Heidelberg, 2009.

[19] N. Rangel-Valdez, J. Torres-Jiménez, J. Bracho-Rıos, and P. Quiz-Ramos. Problem and algorithm fine-tuning - a case of study using bridge club and simulated annealing. In A. Dourado, A. C. Rosa, and K. Madani, editors, *IJCCI*, IJCCI 2009 - Proceedings of the International Joint Conference on Computational Intelligence, Funchal, Madeira, Portugal, October 5-7, 2009, pages 302–305. INSTICC Press, 2009.

[20] A. Rodriguez-Cristerna, J. Torres-Jimnez, I. Rivera-Islas, C. Hernandez-Morales, H. Romero-Monsivais, and A. Jose-Garcia. A mutation-selection algorithm for the problem of minimum brauer chains. In Ildar Batyrshin and Grigori Sidorov, editors, *Advances in Soft Computing*, volume 7095 of *Lecture Notes in Computer Science*, pages 107–118. Springer Berlin / Heidelberg, 2011.

[21] Arturo Rodriguez-Cristerna and Jose Torres-Jimenez. A simulated annealing with variable neighborhood search approach to construct mixed covering arrays. *Electronic Notes in Discrete Mathematics*, 39(0):249 – 256, 2012. EURO Mini Conference.

[22] Arturo Rodriguez-Cristerna and Jose Torres-Jimenez. A genetic algorithm for the problem of minimal brauer chains for large exponents. In *Soft Computing Applications in Optimization, Control, and Recognition*, pages 27–51. Springer, 2013.

[23] G. Tassey. The economic impacts of inadequate infrastructure for software testing. Technical report, Research Triangle Institute, National Institute of Standards and Technology, U.S. Deparment of Commerce, 05 2002.

[24] J. Torres-Jimenez and E. Rodriguez-Tello. New bounds for binary covering arrays using simulated annealing. *Information Sciences*, 185(1):137–152, February 2012.