# Strength Two Covering Arrays Construction Using a SAT Representation

Daniel Lopez-Escogido, Jose Torres-Jimenez,
Eduardo Rodriguez-Tello, and Nelson Rangel-Valdez

Laboratorio de Tecnologías de Información, CINVESTAV-Tamaulipas,
Carretera Nacional Cd. Victoria-Monterrey Km 6, Cd. Victoria
Tamaulipas, México, CP. 87276, Tel.: (52 834) 316 6600
{dlopeze,jtj,ertello,nrangel}@cinvestav.mx

**Abstract.** According to the NIST report of 2002 there is a great potential to reduce the cost, and to increase the quality of the software developed in USA through the creation of automated tools that help in the software testing process. One alternative to improve the software testing process is the creation of tools that generate testing cases in an automatic way. Through the construction of Covering Arrays (CA) it is possible to obtain a minimum set of test cases with the maximum possibility of testing all the functionality of the developed software.

In this paper an approach to construct CA using the propositional satisfiability problem (SAT) is presented. The approach starts with the transformation of a CA instance into a non-conjunctive normal form (non-CNF) SAT instance. Then the SAT instance is solved through a non-CNF SAT solver, and finally the SAT solution is transformed into a CA solution. The main contributions of this work are: an efficient non-CNF SAT solver able to equals or improves previously reported results, and a simplified SAT representation to solve the CA problem.

## 1 Introduction

In the software industry the development and production of high-quality software at reasonable cost is a critical issue. A common source of faults in software components is the unexpected interaction between their input parameters. Software testing is used widely to assure the quality of the software that will be offered to the market. The National Institute for Standards and Technology (NIST) [19] reports that software defects cost to the USA economy close to $60 billion per year. They suggest that approximately $22 billion can be saved through a more effective software testing process. Indeed, there is a need for advanced software testing techniques that offer a good cost-benefit ratio.

Software testing process can be divided into different phases [7]: testing small pieces of code written by one programmer, integration and testing of several subsystems, and system testing (the testing of the combinations of subsystems). There are methods to test software such as structural testing or white box testing [14], referred as the type of testing in which the test cases are designed according

A. Gelbukh and E.F. Morales (Eds.): MICAI 2008, LNAI 5317, pp. 44–53, 2008.

to the architectural knowledge of the software under test. Another method is the functional testing or black box testing [14], which refers to the type of testing where only the functional knowledge of the software is known.

When a software component contains many input parameters, the risk of faults is magnified. A software development enterprise would want to exercise all the possible combinations of input parameters in each software component. However, the number of possible input parameters configurations can grow exponentially, so it becomes impractical for the enterprise to accomplish it due to its constraints of time and money. Reducing the number of test cases could produce significant savings, but it is necessary to guarantee some level of quality of the tests.

The Covering Arrays (CA) are combinatorial objects, that recently have been applied to do functional tests to software components. The use of CA allows to test all the interactions, from a given size, among input parameters using the minimum number of test cases.

According to Burr and Young [1], 80% to 90% of the functionality of a software component is exercised if the tests cover all the possible combinations of its inputs parameters in pairs. If all two-way interactions among input parameters in a software component are tested, the set of tests could be reduced. This paper introduces an approach to construct Covering Arrays using the propositional satisfiability problem (SAT).

The rest of this paper is organized as follows. Section 2 presents the CA construction problem. Section 3 gives a formal definition of SAT problem, while Section 4 introduces a model to transform a CA construction problem instance into a SAT instance. The implementation details of a Simulating Annealing algorithm used to solve the resulting SAT instances are presented in Section 5. The experiments carried out with this approach are detailed in Section 6. Finnally, the conclusions of this research work are preented in Section 7.

## 2 Covering Arrays

The CA have been used to test systems that involve several input parameters, such as software components, networks, and circuit systems [4,5,22,24,25]. They have been also used to elaborate experimental designs in areas like agriculture, medicine and manufacturing.

A CA with $k$ columns, with $v$ levels or values for each column, with an interaction size $t$, and $N$ rows is defined using the following notation: $CA(N; k, v, t)$. A CA is a matrix $M$ of size $N \times k$, where each element $m_{i,j}$ can take only one value from the set $S = \{0, 1, 2, ..., v-1\}$. The parameter $v$ is called the alphabet of the CA and $t$ is called the interaction size or strength. The parameter $N$ is called the size of the CA. Given $t$, $k$ and $v$ the Covering Array Number, denoted by $N = CAN(k, v, t)$, is the minimum $N$ for which there exists a $CA(N; k, v, t)$. A CA of size $N = CAN(k, v, t)$ is called optimal. The main goal in CA construction is to minimize the $N$ value given $k$, $v$ and $t$. A CA of strength two has the following property: every subset of two columns $j$, $l$ $(0 \leq j < l < k)$ contains all

the elements of $\{0, 1, 2, ..., v-1\}^2$ at least once. This paper deals with the CA of strength two.

There are several approaches to construct CA; In [23] the author reported a greedy algorithm, the CA is constructed adding new rows to the matrix $M$ one at a time according to a special heuristic. Chateauneuf and Kreher [3] use algebraic techniques for their construction. Hartmann and Raskin [12] describe a software package called Combinatorial Test Services (CTS), which finds CA with the smallest number $N$ using a variety of constructive methods and choosing the best result. Nurmela [18] developed a Tabu Search algorithm to find CA which improved some previously reported upper bounds. The problem of finding optimal CA has solution for only some special cases, for example when the alphabet is binary and the strength is two [16], but in general, the problem is NP-complete [17].

An example of an optimal CA with 4 columns ($k = 4$), 3 possible values ($v = 3$), and strength $t = 2$, is shown in Table 1.

**Table 1.** A CA(9;4,3,2). Every subset of two columns $j$, $l$ $(0 \leq j < l < k)$ contains all the elements of $\{0, 1, 2\}^2$.

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 2 | 0 | 2 | 2 |
| 0 | 2 | 2 | 1 |
| 2 | 1 | 0 | 1 |
| 0 | 1 | 1 | 2 |
| 2 | 2 | 1 | 0 |
| 1 | 1 | 2 | 0 |
| 1 | 2 | 0 | 2 |

In the following section the propositional satisfiability problem that will be used to model the CA instances is described.

## 3 The Propositional Satisfiability Problem

The propositional satisfiability problem has many applications in the areas of computer science, artificial intelligence, hardware design, automatic electronic design and verification, among others. The SAT problem consists in finding a truth assignment to a set of boolean variables of a formula $F$ such that the formula $F$ is *true* (*i.e.* the formula $F$ is satisfied).

The SAT problem is an NP-complete problem [6]. The SAT problem involves finding an assignment to a set of boolean variables $x_i, x_2, ..., x_n$ that satisfies a set of constraints represented by a well-formed boolean formula in CNF format $F : B^n \rightarrow B$, $B = \{0, 1\}$, *i.e.*, a conjunction of clauses, each of which is a disjunction of variables. If such assignment exists the instance is called $satisfiable$ and $unsatisfiable$ otherwise.

The SAT problem has been used as a language to encode several kinds of hard problems, and in many cases solving a problem in the SAT domain is easier than solving the original problem. In this work SAT is used to model CA construction problem instances. The following section presents this transformation model.

## 4 Transformation Model

The approach presented in this paper produces a non-CNF formula to model instances of CA of strength two. The model for a $CA(N; k, v, 2)$ instance requires for each element $m_{i,j}$ of the matrix $M$ associated with the CA instance the use of $v$ boolean variables $m_{i,j,x}$ $(0 \leq x < v)$. The element $m_{i,j}$ of the matrix $M$ takes the value $x$ iff the boolean variable $m_{i,j,x}$ is *true*. The clauses required are presented in the Equations 1, 2 and 3.

$$\kappa_1 = \bigwedge_{i=0}^{N-1} \Big( \bigvee_{\forall j,x | 0 \leq j < k, 0 \leq x < v} m_{i,j,x} \Big) \tag{1}$$

$$\kappa_2 = \bigwedge_{i=0}^{N-1} \Big( \bigvee_{\forall j,x,y | 0 \leq j < k, 0 \leq x < y < v} (\overline{m_{i,j,x}} \vee \overline{m_{i,j,y}}) \Big) \tag{2}$$

$$\kappa_3 = \bigwedge_{\forall x,y | 0 \leq x \leq y < v} \Big( \bigwedge_{\forall j,l | 0 \leq j < l < k} \Big( \bigvee_{\forall i | 0 \leq i < N} (m_{i,j,x} \wedge m_{i,l,y}) \Big) \Big) \tag{3}$$

The Equation 1 defines clauses that guarantee that each element in the CA matrix $M$ takes at least a value from $\{0, 1, ..., v-1\}$; it generates $N \times k$ clauses, and the size of every clause is $v$. The clauses defined by the Equation 2 specify that each element in the CA matrix $M$ takes only one value; the total number of clauses produced by this equation are $N \times k \times \binom{v}{2}$, each one of size 2. The clauses shown in Equations 1 and 2 are in CNF.

The third set of clauses, shown in the Equation 3, guarantee that the CA is constructed correctly according to its definition, *i.e.* that the interactions among the columns is met. These clauses are organized in a non-CNF. The SAT formula that represents a CA instance will include the three set of clauses previously described, as shown in the Equation 4.

$$F = \kappa_1 \wedge \kappa_2 \wedge \kappa_3 \tag{4}$$

The SAT model for CA(4;2,2,2) is given in Equations 5, 6 and 7.

$$\begin{aligned} \kappa_1 = {} & (m_{0,0,0} \vee m_{0,0,1}) \wedge (m_{0,1,0} \vee m_{0,1,1}) \wedge \\ & (m_{1,0,0} \vee m_{1,0,1}) \wedge (m_{1,1,0} \vee m_{1,1,1}) \wedge \\ & (m_{2,0,0} \vee m_{2,0,1}) \wedge (m_{2,1,0} \vee m_{2,1,1}) \wedge \\ & (m_{3,0,0} \vee m_{3,0,1}) \wedge (m_{3,1,0} \vee m_{3,1,1}) \end{aligned} \tag{5}$$

$$\begin{aligned} \kappa_2 = {} & (\overline{m_{0,0,0}} \vee \overline{m_{0,0,1}}) \wedge (\overline{m_{0,1,0}} \vee \overline{m_{0,1,1}}) \wedge \\ & (\overline{m_{1,0,0}} \vee \overline{m_{1,0,1}}) \wedge (\overline{m_{1,1,0}} \vee \overline{m_{1,1,1}}) \wedge \\ & (\overline{m_{2,0,0}} \vee \overline{m_{2,0,1}}) \wedge (\overline{m_{2,1,0}} \vee \overline{m_{2,1,1}}) \wedge \\ & (\overline{m_{3,0,0}} \vee \overline{m_{3,0,1}}) \wedge (\overline{m_{3,1,0}} \vee \overline{m_{3,1,1}}) \end{aligned} \tag{6}$$

$$\begin{aligned}\kappa_3 = &((m_{0,0,0} \wedge m_{0,1,0}) \vee (m_{1,0,0} \wedge m_{1,1,0}) \vee \\ &(m_{2,0,0} \wedge m_{2,1,0}) \vee (m_{3,0,0} \wedge m_{3,1,0})) \wedge \\ &((m_{0,0,0} \wedge m_{0,1,1}) \vee (m_{1,0,0} \wedge m_{1,1,1}) \vee \\ &(m_{2,0,0} \wedge m_{2,1,1}) \vee (m_{3,0,0} \wedge m_{3,1,1})) \wedge \\ &((m_{0,0,1} \wedge m_{0,1,0}) \vee (m_{1,0,1} \wedge m_{1,1,0}) \vee \\ &(m_{2,0,1} \wedge m_{2,1,0}) \vee (m_{3,0,1} \wedge m_{3,1,0})) \wedge \\ &((m_{0,0,1} \wedge m_{0,1,1}) \vee (m_{1,0,1} \wedge m_{1,1,1}) \vee \\ &(m_{2,0,1} \wedge m_{2,1,1}) \vee (m_{3,0,1} \wedge m_{3,1,1}))\end{aligned} \tag{7}$$

In the literature Hnich *et al.* [13] reported a model that transforms CA instances into SAT instances. The model reported in [13] uses $N \times k \times v + N \times \binom{k}{2} \times v^2$ boolean variables and $2N \times k \times v + v^2 \times N \times \binom{k}{2} + 2N \times v^2 \times \binom{k}{2}$ literals. The non-CNF model described in this paper only requires $N \times k \times v$ boolean variables, and $2N \times (k + k \times \binom{v}{2} + v^2 \times \binom{k}{2})$ literals. If both models are compared then it will be evident that the search space in the model reported in [13] is greater than that produced by the model proposed here.

On the other hand, no model that uses $N \times k \times v$ boolean variables to transform CA into a CNF SAT instance was found in the literature. An alternative is to apply the distributive law to Equation 3 but it will result in an exponential growth in the number of literals. In particular, the number of literals produced will be $v^2 \times \binom{k}{2} \times 2^N$.

In order to solve the non-CNF SAT model, a Simulated Annealing metaheuristic was used, implementation details are described in the next section.

## 5 A Simulated Annealing Implementation

The SAT problem is an issue widely studied, and many researchers have developed many approaches to solve different instances of this problem. There are two main approaches to construct solvers for SAT instances, they are complete SAT solvers and incomplete SAT solvers. The complete SAT solvers usually are based on the Davis-Putnam procedure [8,9], which employs a systematic backtracking search to explore the solution space, looking for a satisfying assignment. The main inconvenient with this procedure is that the execution time can grow exponentially according to the problem size (the number of boolean variables).

The incomplete SAT solvers employ mainly local search procedures, these solvers may find a solution in less time than the complete SAT solvers, but there is not guarantee that a solution can be found even if it exists. Local search algorithms have shown to be very effective for solving SAT instances [21], specially when the size of the instance does not allow to use an exact algorithm [10,11,20].

In this work a local search implementation based on a classic Simulated Annealing (SA) [2,15] was used to construct an incomplete SAT solver for the proposed model. This solver avoids the use of the set of clauses shown in Equations 1 and 2 by constructing a random initial solution, that ensures the correct assignment of values to elements of the matrix $M$. The correct assignment of values implies that each element $m_{i,j}$ of the matrix $M$ will take only one value.

It is formally defined as follows: let $m_{i,j,x}$ be the set of boolean variables created by model from a CA instance then if $m_{i,j,x} \leftarrow true$ then $m_{i,j,y} \leftarrow false, \forall x \neq y$.

The evaluation function for the SA returns the number of unsatisfied clauses. In order to find a solution for the CA instance the number of unsatisfied clauses must be zero. The solution will be a matrix $M$ where the value of each element $m_{i,j}$ will be $x$ iff the boolean variable $m_{i,j,x} = true$.

In the algorithm a set of $p$ solutions is created from the current solution by randomly selecting two variables from it and exchanging their values. The neighboring function then chose as the new solution the neighbor, the solution with the best value of the evaluation function.

After that, the algorithm will form a new neighbor by exchanging values between respect to the matrix $M$.

In the next section the experiments carried out with this SA algorithm are presented.

## 6 Experiments and Results

The model introduced in Section 4 was used to model different strength two CA instances, and for solving them. This section presents the main results obtained. In a preliminary experiment a complete SAT solver for this representation was implemented. It only used the set of clauses 3; and it employed the Davis-Putnam procedure [8,9]. The complete solver showed good performance in small cases (see Table 2), but in CA instances with more than four columns and alphabets greater than five the execution time grew up exponentially.

Due to the excessive consumed time consumed by this complete SAT solver, a second experiment was designed. This experiment used an incomplete SAT solver based on the SA approach described in Section 5. The parameters used for the SA were: an initial temperature $c_0 = 1$, a final temperature $c_f = 1 \times 10^{-9}$, a cooling factor $\alpha = 0.99$, and a Markov chain length $L = N \times k \times v$. The number of solutions used for the neighboring function is $p = 20 \times v$.

Since the optimum value of $N$ is unknown we defined a lower bound and an upper bound, then a binary search technique was used to calculate a new value for $N$. The process is repeated with this new value of $N$. This procedure ends when it is not possible to lower the value of $N$ anymore. Elementary counting arguments show that the minimal size of a CA of strength two is $v^2$, then $v^2$ was used as the lower bound. As the upper bound we selected $p^2$, were $p$ is the

**Table 2.** Strength two Covering Arrays constructed using a complete SAT solver

| $k$ | $v$ | $N$ | **time[sec]** |
|---|---|---|---|
| 3 | 3 | 9 | 0.04 |
| 4 | 3 | 9 | 0.04 |
| 3 | 4 | 16 | 0.47 |
| 4 | 4 | 16 | 1.73 |
| 5 | 4 | 16 | 3.61 |
| 3 | 5 | 25 | 518.38 |

.

**Table 3.** Results for $CA(k, v, 2)$. This table shows the results obtained by the non-CNF solver implemented in this work. Its results are compared with those reported in the literature. HW, HR, CK, NU denotes [13,12,3,18] respectively (results in italics are not the optimal value, these are the best upper bounds reported).

| | | $N$ | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $k$ | $v$ | **non-CNF** | **HW** | **HR** | **CK** | **NU** | Optimal | time[sec] |
| 3 | 3 | 9 | 9 | 9 | 9 | - | 9 | 0.002 |
| 3 | 4 | 16 | 16 | 16 | 16 | - | 16 | 0.027 |
| 3 | 5 | 25 | 25 | 25 | 25 | - | 25 | 0.53 |
| 3 | 6 | 36 | 36 | 36 | 36 | - | 36 | 1.1 |
| 3 | 7 | 49 | 49 | 49 | 49 | - | 49 | 6.93 |
| 4 | 3 | 9 | 9 | 9 | 9 | - | 9 | 0.007 |
| 4 | 4 | 16 | 16 | 16 | 16 | - | 16 | 0.034 |
| 4 | 5 | 25 | 25 | 25 | 25 | - | 25 | 0.95 |
| 4 | 6 | 37 | 37 | 48 | 37 | 37 | *37* | 32.1 |
| 4 | 7 | 49 | 49 | 49 | 49 | - | 49 | 3589.2 |
| 5 | 3 | 11 | 11 | 15 | 11 | 11 | *11* | 0.6 |
| 5 | 4 | 16 | 16 | 16 | 16 | - | 16 | 1.3 |
| 5 | 5 | 25 | 25 | 25 | 25 | - | 25 | 5.27 |
| 5 | 6 | 39 | 39 | 48 | 39 | 39 | *39* | 5621.42 |
| 5 | 7 | 52 | 52 | 49 | 49 | - | 49 | 7237.39 |
| 6 | 3 | 12 | 12 | 15 | 12 | - | *12* | 0.99 |
| 6 | 4 | 19 | 19 | 24 | 19 | 19 | *19* | 338.24 |
| 6 | 5 | 25 | 25 | 25 | 25 | - | 25 | 2360.52 |
| 6 | 6 | 42 | 42 | 48 | 41 | 41 | *41* | 4070.16 |
| 6 | 7 | **57** | 58 | 49 | 49 | - | 49 | 4622.93 |
| 7 | 3 | 12 | 12 | 15 | 12 | - | *12* | 121.83 |
| 7 | 4 | 21 | 21 | 28 | 21 | 21 | *21* | 782.12 |
| 7 | 5 | 29 | 29 | 45 | 29 | 29 | *29* | 3125.23 |
| 7 | 6 | **44** | 45 | 48 | 42 | - | *42* | 3223.93 |
| 7 | 7 | **60** | 61 | 49 | 49 | - | 49 | 7085.96 |
| 8 | 3 | 14 | 14 | 15 | 13 | - | *13* | 0.63 |
| 8 | 4 | 23 | 23 | 28 | 23 | 23 | *23* | 10.23 |
| 8 | 5 | 34 | 34 | 45 | 33 | 33 | *33* | 6552.51 |
| 8 | 6 | **47** | 48 | 48 | 42 | 42 | *42* | 6842.18 |
| 8 | 7 | 63 | 63 | 49 | 49 | - | 49 | 7152.83 |

smallest prime greater than $k$ and $v$, the logic behind this is that it is possible to construct a $CA(p^2; k, v, 2)$ by using a $CA(p^2, p+1, p, 2)$ [12].

Both, the complete and incomplete SAT solvers were coded in C and compiled with gcc using -O3 and -lm as parameters for the compiler. They were run into a Pentium 4 at 2.8 GHz with 1 GB of RAM under a Linux operating system.

The incomplete SAT solver was compared with four approaches reported in the literature: Nurmela (NU) solved CA using tabu search [18], Chateauneuf and Kreher (CK) used mathematical constructions [3], Hartman and Raskin (HR)

[12] used direct and recursive methods and Hnich *et al.* [13] (HW) using a CNF SAT solver.

The results obtained in the experimentation are shown in Table 3. The columns 1 and 2 indicate the number of columns and the CA alphabet for each instance. Column 3 shows the results obtained using the approach presented in this paper. When the font is bold indicates that the results obtained by the approach presented in this paper is better than the reported in [13]. The columns 4, 5, 6 and 7 present the results reported by [13,12,3,18], respectively. Column 8 depicts the best reported results for each instance; when the number appears in italics corresponds to the best known solution, otherwise it is an optimal value. The column 9 shows the CPU time in seconds spent by our approach to solve each instance.

## 7 Conclusions

In this work a model to transform CA of strength two into the SAT problem instances was presented. The SAT instance produced is in a non-CNF form and is smaller, in the number of literals and boolean variables, than its equivalent in CNF form previously reported in [13].

An incomplete SAT solver based on SA was implemented. This solver used local search to construct the CA for a benchmark composed of 30 instances.

It was experimentally shown that it is possible to construct in a competitive way CA using a non-CNF SAT formula that represents the problem. Furthermore, it was shown that the performance of the incomplete SAT solver is good for reasonable size CA instances.

The approach presented in this paper enable us to equal and even improve some results produced by a previously similar approach reported.

## Acknowledgments

This research was partially funded by the following projects: CONACyT 58554 Cálculo de Covering Arrays, 51623 Fondo Mixto CONACyT y Gobierno del Estado de Tamaulipas, CONACyT 74521 Repatriación.

## References

1. Burr, K., Young, W.: Combinatorial test techniques: Table-based automation, test generation, and code coverage. In: Proceedings of the International Conference on Software Testing Analysis and Review, San Diego, USA, pp. 503–513 (October 1998)
2. Cerny, V.: A thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal Optimization Theory and Applications 45, 41–51 (1985)

3. Chateauneuf, M., Kreher, D.L.: On the state of strength-three covering arrays. Journal of Combinatorial Designs 10(4), 217–238 (2002)
4. Cohen, D.M., Parelius, J., Dalal, S.R., Patton, G.C.: The combinatorial design approach to automatic test generation. IEEE Sofware 13(5), 83–88 (1996)
5. Cohen, D.M., Dalal, S.R., Fredman, M.L., Patton, G.C.: The aetg system: an approach to testing based on combinatorial design. Transactions on Software Engineering 23(7), 437–444 (1997)
6. Cook, S.A.: The complexity of theorem-proving procedures. In: STOC 1971: Proceedings of the third annual ACM symposium on Theory of computing, pp. 151–158. ACM, New York (1971)
7. Dalal, S.R., Mallows, C.L.: Factor-covering designs for testing software. Technometrics 40(3), 234–243 (1998)
8. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications. ACM 5(7), 394–397 (1962)
9. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM 7(3), 201–215 (1960)
10. Gu, J.: Efficient local search for very large-scale satisfiability problems. SIGART Bull. 3(1), 8–12 (1992)
11. Hansen, P., Jaumard, B.: Algorithms for the maximum satisfiability problem. Computing 44(4), 279–303 (1990)
12. Hartman, A., Raskin, L.: Problems and algorithms for covering arrays. Discrete Math. 284(1-3), 149–156 (2004)
13. Hnich, B., Prestwich, S.D., Selensky, E., Smith, B.M.: Constraint models for the covering test problem. Constraints 11(2-3), 199–219 (2006)
14. Jorgensen, P.C.: Software Testing: A Craftsman's Approach. CRC Press, New York (2002)
15. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 4598, 671–680 (1983)
16. Kleitmain, D.J., Spencer, J.: Families of k-independent sets. Discrete Math. 6, 255–262 (1973)
17. Lei, Y., Tai, K.C.: In-parameter-order: A test generation strategy for pairwise testing. In: HASE 1998: The 3rd IEEE International Symposium on High-Assurance Systems Engineering, Washington, DC, USA, pp. 254–261. IEEE Computer Society, Los Alamitos (1998)
18. Nurmela, K.J.: Upper bounds for covering arrays by tabu search. Discrete Applied Math. 138(1-2), 143–152 (2004)
19. National Institute of Standards and Technology. The economic impacts of inadequate infrastructure for software testing (2002), `www.nist.gov/director/prog-ofc/report02-3.pdf`
20. Selman, B., Kautz, H.A., Cohen, B.: Local search strategies for satisfiability testing. In: Proceedings of the Second DIMACS Challange on Cliques, Coloring, and Satisfiability (1993)
21. Selman, B., Levesque, H., Mitchell, D.: A new method for solving hard satisifiability problems. In: AAAI 1992, San Jose, CA, pp. 440–446 (July 1992)
22. Sloane, N.J.A.: Covering arrays and intersecting codes. Journal of Combinatorial Designs 1(1), 51–63 (1993)
23. Turban, R.C.: Algorithms for covering arrays. PhD thesis, Tempe, AZ, USA, Adviser-Charles Colbourn (2006)

24. Williams, A.W., Probert, R.L.: A practical strategy for testing pair-wise coverage of network interfaces. In: ISSRE 1996: Proceedings of the Seventh International Symposium on Software Reliability Engineering (ISSRE 1996), Washington, DC, USA, pp. 246–256. IEEE Computer Society, Los Alamitos (1996)
25. Williams, A.W.: Determination of test configurations for pair-wise interaction coverage. In: TestCom 2000: Proceedings of the IFIP TC6/WG6.1 13th International Conference on Testing Communicating Systems, Deventer, The Netherlands, pp. 59–74. Kluwer, B.V. (2000)