Constraint Models for the Covering Test Problem*

Brahim Hnich[1], Steven D. Prestwich[1], Evgeny Selensky[2], and Barbara M. Smith[1]

[1] Cork Constraint Computation Centre
University College, Cork, Ireland
{brahim, s.prestwich, b.smith}@4c.ucc.ie,
[2] Vidus Limited, Ipswich, U.K.
eselensky@vidus.com

Abstract. Covering arrays can be applied to the testing of software, hardware and advanced materials, and to the effects of hormone interaction on gene expression. In this paper we develop constraint programming models of the problem of finding an optimal covering array. Our models exploit global constraints, multiple viewpoints and symmetry-breaking constraints. We show that compound variables, representing tuples of variables in our original model, allow the constraints of this problem to be represented more easily and hence propagate better. With our best integrated model, we are able to either prove the optimality of existing bounds or find new optimal solutions, for arrays of moderate size. Local search on a SAT-encoding of the model is able to find improved solutions and bounds for larger problems.

1 Introduction

Software and hardware testing play an important role in the process of product development. For instance, software testing may consume up to half of the overall software development cost [21]. Furthermore, even for simple software or hardware products, exhaustive testing is infeasible because the number of possible test cases is typically prohibitively large. For example, suppose we have a machine with 10 switches that have to be set, each with two positions. We wish to test the machine before shipping. Since there are 2^{10} possible combinations, it becomes impractical to test them all. We might instead be satisfied with a test suite in which every subset of, say, three switches gets exercised in all 2^3 possible ways. In such a case, the question becomes: what is the smallest number of test vectors that we need? This problem is an instance of the *t-covering array* problem.

* This work has received support from Science Foundation Ireland under Grant 00/PI.1/C075 and from Bausch & Lomb Ireland and Enterprise Ireland. The first author is also supported by an ILOG Software license grant. We wish to thank Radoslaw Szymanek for useful comments on an earlier draft of the paper

A covering array $CA(t, k, g)$ of size b is an $b \times k$ array consisting of b vectors of length k with entries from $\{0, 1, \ldots, g-1\}$ (g is the size of the alphabet) such that every one of the g^t possible vectors of size t occurs at least once in every possible selection of t elements from the vectors. The parameter t is referred to as the *covering strength*. The objective is to find the minimum b for which a $CA(t, k, g)$ of size k exists.

As an example, consider generating test vectors for all triples of 5 binary parameters that test all combinations of 3 parameters ($t = 3$, $k = 5$, $g = 2$), i.e. finding a $CA(3, 5, 2)$. The matrix in Figure 1 shows an optimal solution to this covering test problem, with 10 test vectors. We highlight the different combinations of 0 and 1 in the first three columns, to show that all possible combinations occur; this property holds for any subset of three columns.

0	**0**	**0**	0	0
0	0	0	1	1
0	**0**	**1**	0	1
0	**1**	**0**	0	1
0	**1**	**1**	1	0
1	**0**	**0**	0	1
1	**0**	**1**	1	0
1	**1**	**0**	1	0
1	**1**	**1**	0	0
1	1	1	1	1

Fig. 1. A $CA(3, 5, 2)$ of minimum size. All 10 subsets of three columns contain all possible combinations of 0 and 1.

Covering arrays have been applied in many areas; the following are examples given by Colbourn [10]:

Software interaction testing: Software components may produce system faults due to unexpected interactions. Ideally, one should test all possible combinations of components, but there may be prohibitively many combinations. Instead, pairwise or t-wise testing can test a fixed level of interaction which finds a large number of faults in practice.

Hardware testing: In a circuit, input signals interact through arithmetic and logical operations to yield a desired output vector. However, errors may still occur. As in software interaction testing, this hardware testing problem can be addressed using covering arrays.

Testing of advanced materials: Materials are sometimes combined to yield improved properties such as strength, flexibility and melting point. However, certain combinations are toxic or explosive and must be avoided. Covering arrays can aid in designing experiments.

Interactions regulating gene expressions: Hormones impact the expression of particular genes, and may interact with each other. Although not all possi-

ble combinations of hormones can be examined, it is the interactions between small numbers of hormones that are of interest. Again, covering arrays are a very suitable modelling tool for such a problem.

The problem of minimising the number of test cases in a t-wise covering test suite for k parameters with domains of size n was, according to [29], first studied by Rényi [25]. Constructions for optimal covering arrays $CA(2, k, 2)$ were given in the 1970s by Rényi, Katona, and Kleitman and Spencer; see [29] for references. However, when $g > 2$ the problem of finding an optimal $CA(2, k, g)$ is NP-complete [28].

In this paper we develop constraint models of this problem, in its most general form. We show that with a constraint programming approach we are able either to prove optimality of existing bounds or to find new optimal values, for problems of relatively moderate size. When the size of the problem increases — in terms of either alphabet size g or covering strength t — our models' performance degrades, but we are able to find improved (though not provably optimal) bounds for larger problems by applying a local search algorithm to a SAT-encoding of the constraint model.

The rest of the paper is organised as follows. In section 2, we describe the covering test problem and give an overview of related work. In section 3 we detail the proposed constraint models; section 4 discusses the symmetry in the CP models and how it can be dealt with. Section 5 presents experimental results. Sections 6 and 7 present a local search approach based on a SAT-encoding of the problem, with experimental results. We show how the models could be extended to handle more general cases in section 8. Finally, we conclude in section 9 and outline our future directions.

2 Related Work

The covering test problem is a direct application of the problem of covering arrays arising in hardware and software testing: the following definitions are based on Hartman and Raskin [16][3].

Definition 1. *A covering array $CA(t, k, g)$ of size b and strength t, is a $b \times k$ array $A = (a_{ij})$ over $Z_g = \{0, 1, 2, \ldots, g-1\}$ with the property that for any t distinct columns $1 \leq c_1 \leq c_2 \leq \ldots \leq c_t \leq k$, and any member $(x_1, x_2, \ldots, x_t)$ of Z_g^t there exists at least one row r such that $x_i = a_{rc_i}$ for all $1 \leq i \leq t$.*

Definition 2. *The covering array number $CAN(t, k, g)$ is the smallest b for which a $CA(t, k, g)$ of size b exists.*

As mentioned in the last section, optimal covering arrays can be constructed when $t = g = 2$, but in general finding optimal covering arrays is NP-complete. Except for some special cases and small values of the parameters, researchers

[3] We have reversed the roles of the rows and columns in Hartman and Raskin's definition of a covering array.

have aimed at finding small covering arrays rather than provably optimal arrays. Here, we discuss some recent examples of the main approaches that have been used.

Chateauneuf and Kreher [4] survey known results for covering arrays with $t = 3$ and introduce algebraic techniques for their construction. They claim good results for their techniques in comparison with a commercially-available method, AETG, described below. Williams [32] describes a method, TConfig, for constructing covering arrays (for $t = 2$) from smaller 'building blocks', which is fast and was found to give better results than IPO (described below) except for heterogeneous alphabets, i.e. when g is not uniform for all parameters. Hartman and Raskin [16] describe their CTS (Combinatorial Test Services) package which aims to find small covering arrays, using a variety of constructive methods and choosing the best result. They also consider a number of related problems, such as maximising the number k of parameters with domains of size g in a t-wise covering test suite with a fixed number b of test cases, and finding a test suite giving maximum t-wise coverage from b tests. Meagher and Stevens [20] describe a constructive method using group theory which improves on the previously best known solutions for several instances.

Another group of techniques use a greedy strategy to construct covering arrays. Lei and Tai [19] describe a method for constructing pair-wise test suites based on the IPO (In-Parameter-Order) strategy. Given a pair-wise test set for the first two parameters, the remaining parameters are added one at a time and the test suite extended by adding new tests if necessary. Polynomial algorithms for extending a test suite are given; it is claimed that this is a practical approach to generating test suites for large numbers of parameters. AETG [7, 6] is another greedy approach which adds test vectors one at a time, considering many possibilities and choosing the one that covers the largest number of so-far-uncovered parameter combinations. Tung and Aldiwan's Test Case Generator [31] is a similar greedy approach, but makes choices deterministically rather than randomly. The Deterministic Density Algorithm [9, 2] generates only one candidate test vector to add to the suite, but aims to generate a vector that covers more of the uncovered pairs.

Many of the best known results for covering arrays have been found using local search methods; for instance, Nurmela [22] uses tabu search to find small covering arrays and gives some new upper bounds, e.g. for $CAN(3, 12, 2)$. Cohen *et al.* [8] apply hill-climbing and simulated annealing to finding covering arrays with fixed and heterogeneous alphabets, and also consider a variant in which the strength t can be increased for specified subsets of the parameters. However, finding good bounds using such methods can be very time-consuming, especially in comparison to the greedy methods.

To the best of our knowledge, this area has not previously been studied from a constraint perspective. Our first attempt to fill this gap is reported in [17]. In this paper, we present the further development of the CP models and show that constraint-based approaches can compete with existing methods.

3 Constraint-Based Approaches

In this section we explore Constraint Programming models of the covering test problem that exploit features such as global constraints and multiple viewpoints. Many scheduling, assignment, routing and other decision problems can be solved efficiently by CP models consisting of matrices of decision variables (*matrix models* [13]). We can model the problem of generating test vectors using matrices, in different ways. In what follows, we usually assume that we have a binary alphabet $Z_2 = \{0, 1\}$, for clarity, i.e. that $g = 2$. We use the covering array $CA(3, 5, 2)$ of Figure 1 as a running example.

3.1 A naïve matrix model

An obvious CP model of the covering test problem has a $b \times k$ matrix of integer variables, x_{ri}, for $1 \leq r \leq b$ and $1 \leq i \leq k$, such that $x_{ri} = m$ if the value of parameter i in test vector r is m. However, it is hard to express the *coverage* constraints, i.e. that every subset of t parameters must be combined in all possible g^t ways. For every subset of t parameters in each row, we introduce a Boolean variable for each combination that is set to true whenever these t parameters cover that particular combination, by means of *reification constraints*. For example, when $t = 3$, the constraints on these Boolean variables are:

$$x_{rijlmnp} = (x_{ri} = m \ \& \ x_{rj} = n \ \& \ x_{rk} = p) \quad \begin{array}{l} \forall\, r, i, j, l, m, n, p; \\ 1 \leq r \leq b; \\ 1 \leq i < j < l \leq k; \\ 0 \leq m, n, p < g \end{array}$$

We then impose the constraint that each combination of parameter values should occur in at least one test vector for each combination of (in the example) 3 parameters, using a sum constraint over the auxiliary Boolean variables:

$$\sum_r x_{rijlmnp} \geq 1 \quad \forall\, i, j, l, m, n, p; \quad 1 \leq i < j < l \leq k; 0 \leq m, n, p < g$$

Unfortunately, imposing the coverage constraints in this way introduces a huge number of auxiliary variables and reification constraints. Furthermore, propagation of these constraints is inefficient and ineffective. We therefore need a different model where the coverage constraints can be easily expressed and propagated efficiently.

3.2 An alternative matrix model

In our $CA(3, 5, 2)$ example, there are $\binom{5}{3} = 10$ triples of the parameters:

$$T = \{(1,2,3),(1,2,4),(1,2,5),(1,3,4),(1,3,5),(1,4,5),\\ (2,3,4),(2,3,5),(2,4,5),(3,4,5)\}$$

We can exploit an alternative viewpoint of the problem to concisely express the covering constraints. We again use a matrix of integer variables. Each of the b rows in this matrix represents a possible setting of the parameters, as before. However, there are now $\binom{k}{t}$ columns, each representing one of the possible $t-$combinations (i.e. one of the triples in T, in our example). Hence, in the new model, each variable represents a tuple of t variables in the naïve model. We shall refer to the variables of the new model as *compound* variables. For instance, the compound variable $y_{r(i,j,l)}$ represents the tuple of variables (x_{ri}, x_{rj}, x_{rk}) in the original model. The domain of each variable is $\{0,\ldots,2^t-1\}$, or $\{0,\ldots,7\}$ in this example, each value representing a tuple of three values in the original matrix, e.g. $y_{r(i,j,l)} = 7$ represents $x_{ri} = 1, x_{rj} = 1, x_{rk} = 1$. Figure 2 shows the covering array of Figure 1 represented as a solution to the alternative model.

$(1,2,3)$	$(1,2,4)$	$(1,2,5)$	$(1,3,4)$	$(1,3,5)$	$(1,4,5)$	$(2,3,4)$	$(2,3,5)$	$(2,4,5)$	$(3,4,5)$
0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	3	1	1	3	3
1	0	1	2	3	1	2	3	1	5
2	2	3	0	1	1	4	5	5	1
3	3	2	3	2	2	7	6	6	6
4	4	5	4	5	5	0	1	1	1
5	5	4	7	6	6	3	2	2	6
6	7	6	5	4	6	5	4	6	2
7	6	6	6	6	4	6	6	4	4
7	7	7	7	7	7	7	7	7	7

Fig. 2. The same solution as in Figure 1 presented as a matrix of values of the compound variables in the alternative model. Each column represents a triple of parameters, listed at the head of the column.

Coverage constraints. In the alternative matrix model, we can easily express the coverage constraints with the help of *global cardinality constraints* [24]. Each such constraint specifies that every number in the range 0 to $2^t - 1$ should be present *at least* once and *at most* $b - 2^t + 1$ times in the b test vectors in the column corresponding to the t-tuple.

For every t-tuple, i.e. every column of the alternative matrix, we post one such constraint globally over the b test vectors, giving rise to $\binom{k}{t}$ constraints. This ensures that we cover all possible values of any t parameters.

Intersection constraints. The variables of the alternative model represent tuples of values in the covering array and we have to ensure that the values

assigned to two compound variables are consistent, in terms of the values they imply for the covering array. For instance, because the variables in the first and the second column of the alternative matrix model both represent positions 1 and 2 in the test vectors, the parameter values (0 or 1) in these positions should be the same. With the alternative model, we introduce the burden of expressing such *intersection* constraints. So for every row r and every two columns c_1 and c_2, if the two columns share some positions then we state a binary constraint between the variables (r, c_1) and (r, c_2) in the alternative matrix. For instance, for each row r the binary constraint between the compound variables $y_{r,(1,2,3)}$ and $y_{r,(1,2,4)}$ in columns 1 and 2 can be expressed *extensionally* by listing the pairs of values that are allowed for these variables, as follows:

$$\{(0,0),(0,1),(1,0),(1,1),(2,2),(2,3),(3,2),(3,3),(4,4),(4,5),(5,4),(5,5),\\(6,6),(6,7),(7,6),(7,7)\}$$

For instance, $y_{r,(1,2,3)} = 4$ represents $x_{r1} = 1$, $x_{r2} = 0$, $x_{r3} = 0$ while $y_{r,(1,2,4)} = 5$ represents $x_{r1} = 1$, $x_{r2} = 0$, $x_{r4} = 1$, so that the pair of values (4,5) should be allowed by the intersection constraint.

For each compound variable, there are $t\binom{k-t}{t-1}$ other compound variables in the same row that have one of the original variables in common with it; for instance, $y_{r,(1,2,3)}$ has one variable in common with each of $y_{r,(1,4,5)}, y_{r,(2,4,5)}, y_{r,(3,4,5)}$. There are $\binom{t}{2}\binom{k-t}{t-2}$ other compound variables in the same row that have two original variables in common with it; for instance, $y_{r,(1,2,3)}$ has two original variables in common with six other compound variables.

It appears at first sight that we need to define an intersection constraint in each case, giving 9 such constraints for each compound variable, in the example, and 45 constraints for each row (since each has been counted twice). However, many of these constraints are redundant. The compound variables of the alternative model are similar to the dual variables used in the dual graph translation of a non-binary constraint satisfaction problem to one with only binary constraints [1, 11]; in that case, intersection constraints between dual variables are similarly needed to ensure that the values of the original variables implied by values assigned to the dual variables are the same for all dual variables. An intersection constraint between two dual variables is redundant if there is a chain of intersection constraints linking the two such that the original variables that they share are also shared by every dual variable in the chain. [30] shows that if arc consistency is maintained on the intersection constraints after assignments are made, redundant intersection constraints can be removed without affecting the search for solutions.

In the present case, the constraints between pairs of compound variables that have only one original variable in common, or in general, fewer than $t-1$ original variables in common, can be deleted. Further, if a set of compound variables share $t-1$ original variables, the clique of intersection constraints linking them can be reduced to just a path: Figure 3 illustrates this.

When $t = 3$, there are $\binom{k}{2}$ pairs of parameters. For each pair, $k-3$ intersection constraints are needed (for each row of the matrix). For instance, when k=5, only

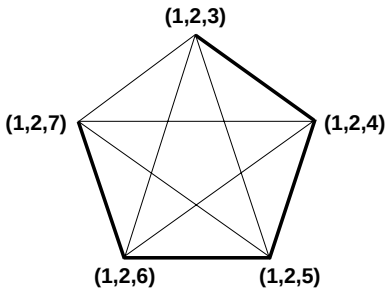

Fig. 3. A clique of triples of original variables, with variables 1 and 2 in common, from the problem with $k = 7$, $t = 3$. Each row of the alternative matrix has a compound variable corresponding to each node, with a binary intersection constraint for each edge. Only four constraints are needed (thick lines mark a possible set); all the others can safely be deleted.

20 constraints are needed, out of the 45 possible constraints. For larger values of k, the difference is more marked: in general, when $t = 3$, the total number of possible intersection constraints in each row is $\binom{k}{3}3(k-2)(k-3)/4$.

3.3 An Integrated Model

In the naïve matrix model, it is difficult to express the coverage constraints in such a way that we can reason efficiently and effectively about them. In the alternative matrix model, on the other hand, we can use global cardinality constraints for which efficient propagation algorithms exist [24]. However, we still have a large number of intersection constraints, even after redundant constraints are removed.

We propose to integrate the two models by using the variables of both, linked by channelling constraints, in the manner proposed by Cheng *et al.* [5]. The disadvantages of integration are the increased number of variables and the additional channelling constraints to be processed; however, the channelling constraints make all the intersection constraints between the compound variables redundant.

The channelling constraints associate each compound variable in the alternative matrix with the t corresponding variables in the original matrix. The association between the values of the compound variables and those of the component original variables is defined in section 3.2. For instance, if $t = 3$ and the alphabet is binary then the constraint between the compound variable $y_{r,(i,j,k)}$ from the alternative matrix and its t corresponding variables in the naïve model x_{ri}, x_{rj}, x_{rk} can be expressed extensionally as follows:

$$(y_{r,(i,j,k)}, x_{ri}, x_{rj}, x_{rk}) \in \{(0,0,0,0), (1,0,0,1), (2,0,1,0), (3,0,1,1) \\ (4,1,0,0), (5,1,0,1), (6,1,1,0), (7,1,1,1)\}$$

or intensionally as:

$$y_{r,(i,j,k)} = 4x_{ri} + 2x_{rj} + x_{rk}$$

For any t-covering we have $\binom{k}{t} \times b$ channelling constraints and the arity of each constraint is $t+1$. If generalized arc consistency is maintained on the channelling constraints, then assigning a value to a compound variable will assign the correct value to each of its constituent variables, and conversely assigning a value to any of the original variables will reduce the domain of any compound variable that the original variable is a component of. Consequently, the role of the intersection constraints in the alternative model is superseded by the channelling constraints in the integrated model. Since there is only one channelling constraint for each compound variable, using channelling constraints rather than intersection constraints reduces the number of constraints, even after redundant intersection constraints have been removed. On the other hand, the channelling constraints have arity $t+1$, whereas the intersection constraints are binary. We shall show in section 5 whether the channelling constraints are more efficient than the intersection constraints, in practice. We discuss the integrated model, and further modifications to it, in the context of our experimental results.

4 Symmetry

Given a covering array $CA(t, k, g)$ of size b, permuting the rows and/or columns gives an equivalent covering array. The rows represent a set of test vectors, and their order is immaterial. Permuting the columns does not affect whether or not every subset of t columns contains every possible vector of length t. Since the variables of the naïve model correspond directly to the elements of the array, the rows and columns of the matrix can also be permuted. Symmetry in a CP model is likely to slow down the search for solutions, since symmetrically-equivalent assignments are needlessly explored.

In matrix models, it is common that permuting both rows and columns transforms a solution into another solution and a non-solution into another non-solution [12]. Row and column symmetry in a matrix model can be reduced by ordering the rows and the columns lexicographically using *lexicographic ordering constraints* [14]. By posing such an ordering constraint between consecutive rows (columns), we break all row (column) symmetry [12]. Whilst it is easy to break all symmetry in one dimension of the matrix, breaking symmetry in both dimensions is harder, as the rows and columns intersect. After constraining the rows to be lexicographically ordered we cannot freely permute the columns, thus the columns are no longer symmetric. Nevertheless, given a matrix with row and column symmetry, each symmetry equivalence class has *at least* one element where both the rows *and* columns are lexicographically ordered. Unfortunately, more than one element where both the rows and columns are lexicographically ordered may exist [12], so lexicographic ordering does not eliminate all row and column symmetry. The implementation of the lexicographic ordering constraint is linear in the size of the vector and maintains *generalized arc consistency*.

In the alternative model, permuting the columns of the matrix of variables in a solution does not in general give another solution. However, the effect of lexicographically ordering the columns of the naïve matrix can be at least partially achieved by lexicographically ordering the columns corresponding to the tuples (1,2,3), (1,2,4), ..., (1,2,k). Again, this can be safely combined with lexicographically ordering the rows.

A final, less obvious, source of symmetry is that the values assigned to any parameter in a set of test vectors can be permuted, without affecting whether or not the set of test vectors is a covering array.[4] In terms of the naïve model, the values within any column of the matrix can be permuted; for instance, with a binary alphabet, the 0s and 1s can be swapped in any column without affecting whether or not the constraints are satisfied. Given the symmetry-breaking constraints already introduced, to deal with this symmetry we need to introduce further constraints that do not conflict with them, since otherwise we would risk losing solutions. There are several possibilities; for instance, we could impose constraints on the number of occurrences of each value in each column. If n_{is} is the number of occurrences of the value s in column i ($1 \leq i \leq k$, $0 \leq s \leq g-1$), then we could add the constraints $n_{i0} \leq n_{i1} \leq \ldots \leq n_{i,g-1}$, for $1 \leq i \leq k$. Clearly, these constraints would not be affected by permuting the rows and columns of the matrix, and hence are compatible with the lexicographic ordering constraints. Alternatively, given a binary alphabet, we can force the first row of the matrix to be all 0s (or equivalently the last row to be all 1s). This constraint gives better propagation than constraining the number of occurrences of each value. It clearly does not interfere with permuting the columns of the matrix, and it is compatible with lexicographic ordering of the rows because it forces the first row of the matrix to be lexicographically smallest (or the last row to be lexicographically largest), whatever the values assigned to the other rows. For $g > 2$, we can combine these ideas; for instance, we can set the first row of the matrix to be all 0s and order the number of occurrences of the values 1 to $g-1$ in each column.

In the alternative model, we can similarly set the first or last row of the matrix (constraining every value in the first row to be 0 or, in the binary case, constraining every value in the last row to be $2^t - 1$). However, we cannot easily impose an equivalent to the constraints on the number of occurrences of each value in the assignments to an individual parameter.

Because the integrated model has both sets of variables, we can break the symmetry of the models in the most convenient and effective way. (We cannot break the same symmetry twice, by constraints on both sets of variables, since this would risk losing solutions.) Hence, we can break the column symmetry by ordering the columns of the original matrix; the row symmetry by ordering the rows of either the original or the alternative matrix; and the value symmetry in each column by constraining the original matrix.

Since the different forms of symmetry can be combined with each other, the size of the symmetry group is $k! \times b! \times (g!)^k$. For a symmetry group of this

[4] Note that this symmetry was not identified in the earlier paper on this work [17].

size, adding symmetry-breaking constraints to reduce the symmetry was the only practical method available. Many of these transformations may have the same effect on any given assignment; even so, the number of possible symmetric equivalents must clearly be very large and it is essential to consider as few of them as possible; as we show later, the symmetry-breaking constraints are vital to solving the CP models.

5 Experiments

We used the different models to solve the covering array problem for different values of the alphabet size, g, the coverage strength t, and the number of parameters, k. As well as, in some cases, finding new solutions to the problem, this allowed us to evaluate the models. First we report on experiments using a Pentium M 1.7GHz PC running ILOG Solver 6.0.[5]

In our experiments we used instances of the covering test problem with coverage strengths t of 3 and 4 over a binary alphabet $Z_2 = \{0, 1\}$. In each experiment we vary the number k of parameters. Initial experiments with the naïve model showed that it was very inefficient and always outperformed by the other models. For this reason, we excluded it from further analysis and do not present results for this model.

In the integrated model, we can use either the original variables or the compound variables as search variables. We found that assigning values to the compound variables was far better; this is understandable, since the coverage constraints are expressed on the compound variables, and no propagation can take place until sufficient compound variables have been assigned. Using the original variables as the search variables delays this propagation. Hence, in the experiments reported below that use the integrated model, the search variables are the compound variables.

In the experiments with the alternative and integrated models, we tried a variety of labelling strategies. In every case, the values of each compound variable were assigned in ascending order. Given a two-dimensional matrix model, two obvious overall strategies to consider are labelling by rows and labelling by columns. In this case, labelling by rows, i.e. labelling all the compound variables corresponding to one row of the matrix before labelling the variables corresponding to the next row, proved to be much worse than labelling by columns. This is not surprising since the main constraints in the problem are the coverage constraints on the columns, which will only propagate when most of the compound variables in any column have been assigned. Hence, the labelling strategy chose a column and assigned the variables corresponding to that column (in some order) before going on to the next column. The columns corresponding to the tuples (1,2,3), (1,2,4), (1,2,5),..., (1,2,k) were labelled, in that order; clearly, the matrix is then completely specified. We tried labelling sets of non-overlapping columns (as far as possible given the value of k), for instance, those corresponding to

[5] http://www.ilog.com/products/optimization.

the tuples (1,2,3) and (4,5,6) when $k = 6$: this was inspired by the labelling of non-overlapping 'supercell' variables in [30], which in that case was a successful labelling strategy. In this case, however, it performed very poorly.

We found that lexicographic ordering (*lex*) within each column gave the same results as smallest-domain ordering (*sdf*). Given the order in which the columns are labelled, after the first column of compound variables has been assigned, the search strategy in effect reduces to assigning the original variables, column by column. With binary domains, *sdf* will behave no differently from *lex*, since a variable either has two values, or is assigned; hence, we use lexicographic ordering within the columns.

With the alternative model, we can add symmetry-breaking ordering constraints on the rows and columns of compound variables, as described in section 4. With the integrated model, we can break row symmetry either on the original or on the alternative matrix (not both). Our experiments showed that both choices led to the same number of backtracks (given the other choices made) but that ordering the rows of the original matrix rather than the alternative matrix reduced the runtime slightly.

With the integrated model, the channelling constraints can be expressed either extensionally, by listing the allowed tuples, or intensionally as a linear constraint, such as $y_{r,(i,j,k)} = 4x_{ri} + 2x_{rj} + x_{rk}$, as described in section 3.3. Solver, by default, enforces generalized arc consistency on extensionally-defined constraints and bounds consistency on linear constraints. With a binary alphabet, achieving bounds consistency on linear channelling constraints is the same as achieving generalized arc consistency, because each value in the domain of the variables is a bound. Since enforcing bounds consistency is much faster than enforcing generalized arc consistency, the runtime is significantly reduced by using the linear constraints rather than the extensional constraints. However, if the alphabet were non-binary, bounds consistency could miss some propagation, and faster constraint processing might then have to be balanced against possibly increased search effort. We have given the results for both extensional and linear channelling constraints to indicate the difference in run-time. From these results, it seems likely that linear channelling constraints would be the better choice overall for non-binary alphabets, even at the expense of increased search; limited experiments with $g = 3$ have confirmed this.

We found in our experiments that ILOG Solver reported the same number of backtracks for both the alternative and integrated models, indicating that they are exploring the same search tree. However, the runtime differs considerably between models. Table 1 displays the results of the experiments in more detail. In all the tables we use bold face to show the best known value and a * to highlight the values that we have proved to be optimal, by showing that there is no solution with a smaller value of b. Our results also show that the integration of the different models is beneficial despite the increase in the number of variables. Even with extensional channelling constraints, the integrated model is much faster than the alternative model. When $k = 12$, for instance, the integrated model has 220 channelling constraints for each row of the matrix: it is evidently

					Alternative Model				Integrated Model	
		Upper bound		No. of back-	All intersection constraints		Minimal set		Extensional constraints	Linear constraints
k	b	in [16]	soluble	tracks	no.	time	no.	time	time	time
4	**8***	**8**	+	0	6	0	6	0	0	0
5	8	12	–	1	45	0.01	20	0.01	0.01	0.01
5	9	12	–	9	45	0.01	20	0.01	0.01	0.01
5	**10***	12	+	5	45	0.01	20	0.01	0.01	0.01
6	10	12	–	93	180	0.33	45	0.06	0.04	0.02
6	11	12	–	820	180	1.82	45	0.31	0.19	0.09
6	**12***	**12**	+	53	180	0.16	45	0.04	0.03	0.02
7	**12***	13	+	124	525	1.32	84	0.13	0.06	0.05
8	**12***	13	+	124	1,260	3.67	140	0.23	0.11	0.06
9	**12***	18	+	467	2,646	31.04	216	1.54	0.58	0.25
10	**12***	18	+	479	5,040	62.08	315	2.50	1.02	0.35
11	**12***	18	+	493	8,910	112.12	440	3.77	1.71	0.71
12	12	18	–	8,476	14,850	> 300	594	48.52	29.96	5.76
12	13	18	–	316,791	14,850	> 300	594	> 300	> 300	269.58

Table 1. Finding $b_{min} = CAN(3, k, 2)$ for different parameters k using the alternative and integrated models. The alternative model has either all the intersection constraints or a minimal set; the number of constraints for each row of the matrix is shown in each case. In the integrated model, the channelling constraints between each compound variable and its constituent variables are either extensionally defined (and generalized arc consistency is maintained) or linear (and bounds consistency is maintained). The run-time is given in seconds, on a Pentium M 1.7GHz PC, running ILOG Solver 6.0. The run-time limit is 5 minutes.

much faster to enforce generalized arc consistency on these than on the 594 non-redundant intersection constraints of the alternative model, even though the channelling constraints are in this case 4-ary whereas the intersection constraints are binary.

Note also that our results use the symmetry-breaking constraints in all tested models. In fact, the symmetry-breaking constraints play a vital part. For example, to prove that there is no covering array $CA(3, 6, 2)$ of size 10 can be done with 93 backtracks and 0.02 sec., with the integrated model, as shown in Table 1; without the symmetry breaking the same problem takes 975,024 backtracks and 63 sec. with the same search strategy.

Finally, our approach improved on several of the results given in [16] for $t = 3$ and $k \leq 11$ and proved optimality. We ran a further set of experiments to find covering arrays $CA(4, k, 2)$ for varying k. We observe in Table 2 that the best integrated model could find $CAN(4, k, 2)$ for $k \leq 6$ in 1 hour, and the improvements of the bounds that we obtained are significantly larger than the improvements we got on $CAN(3, k, 2)$. Overall, with the presented approach we can find provably optimal covering test suites for those instances which induce a moderate number of variables in our models. This translates to getting

k	b	Upper bound in [16]	soluble	No. of backtracks	time
5	**16***	24	+	0	0.01
6	16	28	–	1	0.01
6	17	28	–	12	0.02
6	18	28	–	146	0.06
6	19	28	–	1,863	0.40
6	20	28	–	20,381	2.99
6	**21***	28	+	160	0.11
7	21	38	–	184,661	35.47
7	22	38	–	1,419,407	247.08
7	23	38	–	9,518,449	1504.86

Table 2. Finding covering arrays $CA(4, k, 2)$ of size b, or proving that there is none, for different parameters k using the integrated model. The runtime limit is 1 hour.

$CAN(3, k, 2)$ for up to $k = 11$ parameters (around 2000 variables) within a CPU time limit of 5 minutes. However, as problem size becomes larger the required amount of search proves computationally prohibitive, and for larger problems we turn to local search methods.

6 A Model for Local Search

Constraint solvers typically alternate variable assignment with constraint propagation; when propagation leads to an empty variable domain, backtracking occurs. An alternative way of finding solutions to constraint problems is local search. Usually starting from a randomly-chosen assignment of all variables, single variables (or sometimes more than one) are selected and reassigned to a different value, each reassignment being a *local move*. The choice of variable and value is made heuristically, with no attempt to maintain completeness of search. This is in contrast to backtrack search, which is complete and can therefore find all solutions, or prove that no solutions exist. The advantage of local search is that it can sometimes solve much larger problems than backtrack search. We decided to evaluate local search on our problem, and chose SAT as a framework in which to experiment. SAT problems may be viewed as CSPs with binary domains and non-binary constraints, and many effective local search algorithms have been designed for SAT. After experiments with different algorithms we chose a new variant of the Walksat algorithm [26], described below.

We could simply SAT-encode our best CP model using one of the well-known standard approaches, but the best model for local search is not necessarily the best model for backtrack search [23]. In fact our SAT model is not identical to any of the previous matrix models, for reasons given below. As before we suppose a $b \times k$ matrix M of integers in Z_g. For each row i, column j and value x define a Boolean variable m_{ijx} which is true if and only if x occurs in position (i, j). We

also suppose an alternative $b \times \binom{k}{t}$ matrix A of integers in Z_{g^t}. For each row i, column j' and value y define a Boolean variable $a_{ij'y}$. The following constraints are defined for all $1 \leq i \leq b$, $1 \leq j \leq k$, $1 \leq j' \leq \binom{k}{t}$, x,$x' \in Z_g$ and y,$y' \in Z_{g^t}$. Each M and A position must take exactly one symbol:

$$\bigvee_x m_{ijx} \tag{1}$$

$$\bar{m}_{ijx} \vee \bar{m}_{ijx'} \tag{2}$$

$$\bigvee_y a_{ij'y} \tag{3}$$

$$\bar{a}_{ij'y} \vee \bar{a}_{ij'y'} \tag{4}$$

where $x < x'$ in (2) and $y < y'$ in (4). The coverage constraints are:

$$\bigvee_{j'} a_{ij'y} \tag{5}$$

To channel between the two matrices we infer the values of the t entries in M for the corresponding A entries:

$$\bar{a}_{ij'y} \vee m_{ijx} \tag{6}$$

for all i, j, j', x, y such that $M_{ij} = x$ and $A_{ij'} = y$ are consistent. We refer to our SAT model as the *weakened matrix model* because it omits several constraints, as follows. Firstly the upper bound on the coverage constraints is hard to express in SAT. This is an implied constraint, and though implied clauses sometimes aid local search [3, 18] they are not a necessary part of the model. Secondly, symmetry breaking constraints can have a negative effect on local search performance [23]. Omitting them aids local search by increasing the number of SAT solutions, and also by reducing the size of the model and thus improving the flip rate (number of local moves per second). We therefore omitted upper bound and symmetry breaking constraints from our encoding.

The third difference is perhaps less obvious. When applying local search to a SAT-encoded constraint satisfaction problem (CSP) it is common to omit clauses ensuring that each CSP variable is assigned no more than one domain value [27], again improving performance. A CSP solution can still be extracted from a SAT solution by taking any one of the assigned domain values for each CSP variable. Here we may omit clauses (1,3,4). Note that we can still extract a CSP solution from any SAT solution: by clauses (5) in any SAT solution each combination of symbols occurs in at least one row of A for each combination of t columns; by clauses (6) each such occurrence induces the corresponding entries in M; and by clauses (2) no more than one value is possible in each M position. In fact the omitted clauses (1,3,4) are implied by clauses (2,5,6), and experiments suggest that omitting them makes little difference to the search effort. It reduces the size of the encoding but not its space complexity, which is dominated by the channelling constraints and is $O(\binom{k}{t}btg^t)$ literals.

7 Local Search Experiments

We use a new SAT local search algorithm, which works as follows. Starting from a random initial truth assignment to all Boolean variables, it repeatedly selects a *violated clause* (one in which all literals are false) and *flips* the truth value of a heuristically-selected variable (if it is true then set it to false, and vice-versa). Such a flip is guaranteed to satisfy the selected clause, but may cause others to become unsatisfied. The key ingredient in such algorithms is the variable selection heuristic. In our algorithm, with a given probability p we randomly select a variable, where p is a *noise parameter* for the algorithm (this is called a *random walk move* in the literature). Otherwise, with probability $1 - p$, we select the variable with smallest *score*. The score for a variable v is defined as the change in the total number of unsatisfied clauses that would result from flipping v, divided by the total number of flips on all variables other than v. Ties are broken by selecting the variable that was least-recently flipped. However, if the clause contains several variables such that flipping them would not create new unsatisfied clauses, then instead we randomly select one of these variables; in the literature this is sometimes called a *freebie move*. The aim of this scoring function is to minimise the number of unsatisfied clauses, while preferring variables that have been flipped fewest times in the search so far. This new variant of Walksat is a simple hybrid of two well-known variants that has not, to the best of our knowledge, been described in the literature: it is HWSAT [15] augmented with the freebie moves of SKC [26]. We shall call it HWFSAT, and in experiments on covering arrays it gave better results than other variants tested.

Our results were found using HWFSAT with the noise parameter p set to 0.2, 0.3 or 0.4. In each case we chose the most promising value, after monitoring the progress of the algorithm under each value. We ran it on a 733 MHz Pentium III, using decreasing values of b until no solution was found after an hour. The results for various values of t, k, g are shown in Tables 3 and 4 with the best result for each instance shown in **bold**, and unavailable results denoted by "—". We compare HWFSAT results with those from several recent papers: HW denotes HWFSAT; HR, Hartmann and Raskin [16]; CK, Chateaneuf and Kreher [4] using a mathematical construction; MS, Meagher and Stevens [20] using group theory; and NU, Nurmela [22] using Tabu search.

HWFSAT was able to reproduce the improved bounds found by ILOG Solver, and to further improve some bounds. On some instances we could not match the best results of the other approaches, and constructive methods are able to provide bounds for much larger instances than we can handle. But SAT-based local search can find solutions that are competitive with the best known results on many instances, and in two cases we improve on the best known results, as far as we can ascertain: upper bounds of 40 for $CAN(3, 7, 3)$ and 24 for $CAN(4, 10, 2)$. Because of the earlier finding that there is no $CA(4, 7, 2)$ with value 23, shown in Table 2, the solution with value 24 found by HWFSAT for $CA(4, 10, 2)$ is optimal, and 24 is also the optimal value for $k = 7, 8$ and 9. Examples of these results are shown in Figures 4 and 5, with rows and columns transposed for space reasons. Our $CAN(4, k, g)$ results also beat those of Hartman & Raskin.

Though we do not expect the SAT approach to scale to much larger problems because of increasing SAT model sizes, a local search algorithm using a higher-level constraint model could avoid this problem.

0 1 1 0 1 0 0 0 1 1 1 0 0 1 1 0 1 0 1 0 0 0 1 1
1 0 1 0 0 1 1 0 1 0 0 0 0 1 1 1 1 0 1 1 1 0 0 0
0 1 1 1 0 0 1 0 0 1 1 0 1 0 1 0 0 1 1 1 1 0 0 0
1 0 0 1 0 1 0 0 0 1 0 0 0 1 1 0 0 1 1 0 1 1 1 1
0 1 1 0 0 1 0 1 1 1 0 1 0 1 0 0 0 1 0 1 1 0 0 1
1 0 1 1 0 0 1 1 0 1 1 1 0 1 0 0 1 0 0 0 1 0 1 0
1 0 1 0 1 1 1 0 1 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0
0 1 1 1 0 1 0 0 0 0 0 1 1 0 0 1 1 0 1 0 1 0 1 1
0 1 0 1 1 1 1 0 0 1 0 1 0 1 0 0 1 0 1 1 0 1 0 0
0 1 0 0 0 0 1 0 1 0 1 1 0 1 0 1 0 1 1 0 1 1 1 0

Fig. 4. An optimal covering array $CA(4, 10, 2)$ of size 24 (rows and columns transposed)

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

Fig. 5. A solution showing that $CAN(3, 7, 3) \leq 40$ (rows and columns transposed)

8 Extensions

For reasons of clarity, we presented our models assuming a fixed binary alphabet and uniform coverage. However, our models can easily be extended to model different practical extensions:

- **Larger alphabet.** Although the results presented in section 5 are only for problems with a binary alphabet, our models do already handle non-binary alphabets. The results in Tables 3 and 4 show that the local search model can also handle this extension.
- **Heterogeneous alphabets.** The model can easily be extended to allow heterogeneous alphabets. The domains of the variables as well as the channelling constraints need to be slightly changed to reflect this extension, but the essence of the models remains the same.

k	g	HW	HR	CK	NU
3	3	**9**	**9**	**9**	—
3	4	**16**	**16**	**16**	—
3	5	**25**	**25**	**25**	—
3	6	**36**	**36**	**36**	—
3	7	**49**	**49**	**49**	—
4	3	**9**	**9**	**9**	—
4	4	**16**	**16**	**16**	—
4	5	**25**	**25**	**25**	—
4	6	**37**	48	**37**	**37**
4	7	**49**	**49**	**49**	—
5	2	**6**	**6**	**6**	—
5	3	**11**	15	**11**	—
5	4	**16**	**16**	**16**	—
5	5	**25**	**25**	**25**	—
5	6	**39**	48	**39**	—
5	7	52	**49**	**49**	—

k	g	HW	HR	CK	NU
6	3	**12**	15	**12**	—
6	4	**19**	24	**19**	**19**
6	5	**25**	**25**	**25**	—
6	6	42	48	**41**	—
6	7	58	**49**	**49**	—
7	3	**12**	15	**12**	—
7	4	**21**	28	**21**	**21**
7	5	**29**	45	**29**	**29**
7	6	45	48	**42**	—
7	7	61	**49**	**49**	—
8	3	14	15	**13**	—
8	4	**23**	28	**23**	**23**
8	5	34	45	**33**	—
8	6	48	48	**42**	—
8	7	63	**49**	**49**	—

k	g	HW	HR	CK	MS	NU
9	3	**13**	15	**13**	—	—
9	4	**24**	28	**24**	—	—
9	5	**35**	45	**35**	—	**35**
9	6	51	62	48	**46**	48
9	7	66	**63**	**63**	—	—
10	3	**14**	15	**14**	—	**14**
10	4	25	28	**24**	—	—
10	5	38	45	**37**	—	**37**
10	6	53	78	52	**51**	—
10	7	71	79	63	**61**	63
11	3	**15**	**15**	**15**	—	—
11	4	**25**	28	**25**	—	—
11	5	39	45	38	—	**38**
11	6	**55**	78	**55**	—	—
11	7	73	91	73	**67**	—

Table 3. Results for $CA(2, k, g)$

k	g	HW	HR	CK
9	2	**12**	18	**12**
10	2	**12**	18	**12**
11	2	**12**	18	**12**
12	2	**15**	18	**15**
13	2	**16**	19	**16**
14	2	17	19	**16**
15	2	18	19	**17**
16	2	18	19	**17**
17	2	**18**	24	**18**
18	2	20	24	**18**
5	3	**33**	45	**33**
6	3	**33**	45	**33**
7	3	**40**	45	45
8	3	46	**45**	**45**
9	3	**51**	75	**51**

k	g	HW	HR
7	2	**24**	38
8	2	**24**	42
9	2	**24**	50
10	2	**24**	50
5	3	**81**	135

Table 4. Results for $CA(3, k, g)$ and $CA(4, k, g)$

– **Partial coverage.** To allow for partial coverage, we simply exclude from the global cardinality constraints those values that represent the combinations that need not appear in a solution.
– **Side constraints.** Covering array problems can come with side constraints such as fixed columns or forbidden configurations [16]. CP is convenient for solving problems with such constraints, which can simply be added to the model.

The last three extensions (heterogeneous alphabets, partial coverage and side constraints) would reduce the symmetry inherent in the problem, and the symmetry-breaking constraints would need to be adjusted accordingly.

9 Conclusion

We have presented constraint models of a core problem in combinatorial software testing: the covering test problem. We consider four matrix models for the problem:

– **The naïve matrix model.** This model compactly represents the problem. However, it is difficult to express the coverage constraints in such a way that we can efficiently reason about them.
– **The alternative matrix model.** This model uses compound variables, each representing a tuple of variables in the naïve model. We can thereby overcome the disadvantages of the previous model by the use of powerful global cardinality constraints. However, this comes at the cost of expressing binary intersection constraints between the compound variables.
– **The integrated matrix model.** This model combines the complementary strengths of both models. The coverage constraints are stated using the global cardinality constraints while the intersection constraints become redundant with the channelling constraints linking the original variables and the compound variables. The overhead of this integrated model is the increased number of variables and the non-binary channelling constraints.
– **The weakened matrix model.** This is a modification of the integrated matrix model, and designed for use with a SAT local search algorithm. It omits several constraints with the aim of increasing the number of SAT solutions and reducing runtime overheads.

The problems are highly symmetric, and it is important when using complete backtracking search to deal with the symmetry effectively; we have shown that this can be done.

We show that for moderate problem sizes with a CP approach one can find provably optimal solutions, which improve on the published results. We further showed that a local search algorithm on a SAT-encoding of the problem can find improved solutions for somewhat larger instances. We proved optimality for one instance ($CA(4, 10, 2)$) by a combination of local search to find the optimal solution and complete search using the CP approach to prove that there is no

smaller $CA(4, 7, 2)$, and therefore no smaller $CA(4, 10, 2)$ either. These results show the applicability of constraint-based techniques to the problem, at least for instances up to a certain size. This approach may find application to less pure versions of the problem with side constraints, such as those found in some industrial applications. The easy handling of side constraints (simply by adding them to the model) is one of the advantages of CP.

In future work we will aim to further improve the presented results. One possible direction for improvement could be exploring the effects of different value ordering heuristics on backtrack search. Another direction is to design a dedicated local search algorithm for the problem; this would greatly reduce model sizes, which currently forms a bottleneck on the size of problems that we are able to solve.

The alternative and integrated models show the usefulness of compound variables in expressing complex constraints so that they can propagate effectively. Although similar variables have been used in the well-known dual graph translation of a CSP with non-binary constraints into one with only binary constraints, this work shows that they have wider application in making non-binary constraints easier to express, without necessarily eliminating them. Further work will explore other applications of this modelling pattern.

References

1. F. Bacchus and P. van Beek. On the Conversion Between Non-Binary and Binary Constraint Satisfaction Problems. In *Proceedings of the 15th National Conference on Artificial Intelligence, AAAI 98*, pages 311–318, 1998.
2. R. Bryce, C. Colbourn, and M. Cohen. A Framework of Greedy Methods for Constructing Interaction Tests. In *The 27th International Conference on Software Engineering (ICSE 2005)*, pages 146–155, 2005.
3. B. Cha and K. Iwama. Adding New Clauses for Faster Local Search. In *Proceedings of the 13th National Conference on Artificial Intelligence, AAAI 96*, volume 1, pages 332–337, 1996.
4. M. Chateauneuf and D. Kreher. On the state of strength-three covering arrays. *Journal of Combinatorial Designs*, 10(4):217–238, 2002.
5. B. M. W. Cheng, K. M. F. Choi, J. H. M. Lee, and J. C. K. Wu. Increasing Constraint Propagation by Redundant Modeling: an Experience Report. *Constraints*, 4:167–192, 1999.
6. D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG System: An Approach to Testing Based on Combinatorial Design. *IEEE Trans. Softw. Eng.*, 23(7):437–444, 1997.
7. D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton. The Combinatorial Design Approach to Automatic Test Generation. *IEEE Software*, 13(5):83–88, 1996.
8. M. B. Cohen, P. B. Gibbons, W. B. Mugridge, and C. J. Colbourn. Constructing test suites for interaction testing. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 38–48. IEEE Computer Society, 2003.
9. C. Colbourn, M. Cohen, and R. Turban. A Deterministic Density Algorithm for Pair wise Interaction Coverage. In *The IASTED International Conference on Software Engineering*, pages 242–252, 2004.

10. C. J. Colbourn. Combinatorial aspects of covering arrays. *Le Matematiche (Catania)*, to appear.
11. R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.
12. P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetry in matrix models. In P. van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP-2002)*, LNCS 2470, pages 462–476. Springer, 2002.
13. P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Matrix modelling: Exploiting common patterns in constraint programming. In A. Frisch, editor, *Proceedings of the International Workshop on Reformulating Constraint Satisfaction Problems*, pages 27–41, 2002.
14. A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Global constraints for lexicographic orderings. In P. van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP-2002*, LNCS 2470, pages 93–108. Springer, 2002.
15. I. P. Gent and T. Walsh. Unsatisfied Variables in Local Search. In J. Hallam, editor, *Hybrid Problems, Hybrid Solutions*, pages 73–85. IOS Press, 1995.
16. A. Hartman and L. Raskin. Problems and algorithms for covering arrays. *Discrete Mathematics*, 284(1-3):149–156, 2004.
17. B. Hnich, S. D. Prestwich, and E. Selensky. Constraint-Based Approaches to the Covering Test Problem. In B. Faltings, A. Petcu, F. Fages, and F. Rossi, editors, *Recent Advances in Constraints, Joint ERCIM/CoLogNet International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2004*, LNCS 3419, pages 172–186. Springer, 2005.
18. K. Kask and R. Dechter. GSAT and Local Consistency. In C. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI 95*, pages 616–622, 1995.
19. Y. Lei and K. Tai. In-Parameter-Order: a Test Generation Strategy for Pairwise Testing. In *Third IEEE International High-Assurance Systems Engineering Symposium*, pages 254–161, 1998.
20. K. Meagher and B. Stevens. Group construction of covering arrays. *Journal of Combinatorial Designs*, 13(1):70–77, 2005.
21. N.Kobayashi. *Design and Evaluation of Automatic Test Generation Strategies for Functional Testing of Software*. PhD thesis, Osaka University, 2002.
22. K. Nurmela. Upper Bounds for Covering Arrays by Tabu Search. *Discrete Applied Mathematics*, 138:143–152, 2004.
23. S. Prestwich. Negative Effects of Modeling Techniques on Search Performance. *Annals of Operations Research*, 118:137–150, 2003.
24. J. Régin. Generalized arc consistency for global cardinality constraints. In *Proceedings of the 13th National Conference on Artificial Intelligence, AAAI 96*, pages 209–215. AAAI Press/The MIT Press, 1996.
25. A. Rényi. *Foundations of Probability*. Wiley, 1971.
26. B. Selman, H. A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the 12th National Conference on Artificial Intelligence, AAAI 94*, volume 1, pages 337–343, 1994.
27. B. Selman, H. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the 10th National Conference on Artificial Intelligence, AAAI 92*, pages 440–446, 1992.
28. G. Seroussi and N. Bshouty. Vector Sets for Exhaustive Testing of Logic Circuits. *IEEE Trans. Info. Theory*, IT-34:513–522, 1988.

29. N. J. A. Sloane. Covering Arrays and Intersecting Codes. *J. Combinatorial Designs*, 1:51–63, 1993.
30. B. M. Smith. A Dual Graph Representation of a Problem in 'Life'. In P. van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP-2002*, LNCS 2470, pages 402–414. Springer, 2002.
31. Y.-W. Tung and W. Aldiwan. Automating test case generation for the new generation missionsoftware system. In *Proceedings IEEE Aerospace Conference*, volume 1, pages 431–437, 2000.
32. A. W. Williams. Determination of Test Configurations for Pair-Wise Interaction Coverage. In *Proceedings of the 13th International Conference on the Testing of Communicating Systems (TestCom 2000)*, pages 59–74, 2000.