Generating Event-Sequence Test Cases by Answer Set Programming with the Incidence Matrix

Mutsunori Banbara[1], Naoyuki Tamura[1], and Katsumi Inoue[2]

1 Information Science and Technology Center, Kobe University
1-1 Rokko-dai, Nada-ku, Kobe, Hyogo 657-8501, Japan
{banbara,tamura}@kobe-u.ac.jp

2 National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
inoue@nii.ac.jp

Abstract

The effective use of ASP solvers is essential for enhancing efficiency and scalability. The incidence matrix is a simple representation used in Constraint Programming (CP) and Integer Linear Programming for modeling combinatorial problems. Generating test cases for event-sequence testing is to find a sequence covering array (SCA). In this paper, we consider the problem of finding optimal sequence covering arrays by ASP and CP. Our approach is based on an effective combination of ASP solvers and the incidence matrix. We first present three CP models from different viewpoints of sequence covering arrays: the naïve matrix model, the event-position matrix model, and the incidence matrix model. Particularly, in the incidence matrix model, an SCA can be represented by a $(0, 1)$-matrix called the incidence matrix of the array in which the coverage constraints of the given SCA can be concisely expressed. We then present an ASP program of the incidence matrix model. It is compact and faithfully reflects the original constraints of the incidence matrix model. In our experiments, we were able to significantly improve the previously known bounds for many arrays of strength three. Moreover, we succeeded either in finding optimal solutions or in improving known bounds for some arrays of strength four.

1998 ACM Subject Classification D.1.6 Logic Programming; D.2.5 Testing and Debugging

Keywords and phrases Event-Sequence Testing, Answer Set Programming, Matrix Model, Constraint Programming, Propositional Satisfiability (SAT)

Digital Object Identifier 10.4230/LIPIcs.ICLP.2012.86

1 Introduction

Recent development of Answer Set Programming (ASP) [3, 15, 21] suggests a successful direction to extend logic programming to be more expressive and more efficient. ASP provides a rich modeling language and can be well suited for modeling combinatorial problems in Computer Science and Artificial Intelligence: multi-agent systems, systems biology, planning, scheduling, semantic web, and Constraint Satisfaction Problems (CSPs). Remarkable improvements in the efficiency of ASP solvers have been made over the last decade, through the adoption of advanced techniques of Constraint Programming (CP) and Propositional Satisfiability (SAT). Such improvements encourage researchers to solve hard combinatorial problems by using ASP.

Combinatorial testing is an effective black-box testing method to detect elusive failures of hardware/software. The basic idea is based on the observations that most failures are caused by interactions of multiple components. The number of test cases is therefore much smaller than exhaustive testing. Generating test cases for combinatorial testing is to find a *Covering Array* (CA) in Combinatorial Designs [2, 4, 5, 6, 7, 8, 9, 17, 18, 20, 22, 23, 27, 28]. However, these CA-based combinatorial testing methods can not be directly applied to detect failures that are caused by a particular *event sequence*, an ordering of multiple events to be processed.

Event-sequence testing is a combinatorial testing method focusing on event-driven hardware/software. Suppose we want to test a system with 10 events. We have $10! = 3,628,800$ test cases for exhaustive testing. Instead, we might be satisfied with test cases that exercise all possible 3-sequences of 10 events (strength three event-sequence testing). Naively, we need ${}_{10}P_3 = 8 \times 9 \times 10 = 720$ test cases. We can reduce to less than 240 since one test case covers at least three 3-sequences. The question is "what is the smallest number of test cases that we need now?". It comes down to an instance of the problem of finding optimal *Sequence Covering Array* (SCA) proposed by Kuhn et al [19]. A sequence covering array provides a set of test cases, where each row of the array can be regarded as an event sequence for an individual test case. Fig.1 shows an optimal sequence covering array of 11 rows, an answer of the question above.

ASP solvers have an important role in the latest ASP technology. The effective use of them is essential for enhancing efficiency and scalability. The *incidence matrix* is a simple representation used in CP and integer linear programming for modeling combinatorial problems such as balanced incomplete block designs [9]. Our approach is based on an effective combination of ASP solvers and the incidence matrix.

In this paper, we consider the problem of finding optimal sequence covering arrays by ASP and CP. We first present three CP models from different viewpoints of sequence covering arrays: *naïve matrix model*, *event-position matrix model*, and *incidence matrix model*. Particularly, in the incidence matrix model, an SCA can be represented by a $(0,1)$-matrix called the incidence matrix of the array in which the coverage constraints of the given SCA can be concisely expressed. We then present an ASP program of the incidence matrix model. It is compact and faithfully reflects the original constraints of the incidence matrix model. For example, it requires only 8 rules for the arrays of strength three. From the perspective of ASP, Erdem et al. recently proposed an ASP-based approach for event-sequence testing [11], and have shown that it enables a tester to rapidly specify problems and to experiment with different formulations at a purely declarative level.

In our experiments, we were able to significantly improve the previously known bounds obtained by a greedy algorithm [19] and an ASP-based approach [11] for many arrays of strength three with small to large sizes of events. Moreover, we succeeded either in finding optimal solutions or in improving known bounds for some arrays of strength four.

2 Sequence Covering Arrays and Related Work

▶ **Definition 1.** A *sequence covering array* $SCA(n; S, t)$ is an $n \times |S|$ (n rows and $|S|$ columns) array $A = (a_{ij})$ over a finite set S of symbols with the property that

- each row of A is a permutation of S, and
- for each t-sequence $\sigma = (e_1, e_2, \ldots, e_t)$ over S, there exists at least one row r with column indices $1 \le c_1 < c_2 < \cdots < c_t \le |S|$ such that $e_i = a_{rc_i}$ for all $1 \le i \le t$.

The parameter n is the size of the array, S is the set of events, and t is the *strength* of the array. Then trivial case when $t = 2$ is excluded from further consideration.

▶ **Definition 2.** The *sequence covering array number* $SCAN(S, t)$ is the smallest n for which an $SCA(n; S, t)$ exists.

▶ **Definition 3.** A sequence covering array $SCA(n; S, t)$ is *optimal* if $SCAN(S, t) = n$.

$\boxed{\boldsymbol{a}}$	$\boxed{\boldsymbol{b}}$	$\boxed{\boldsymbol{c}}$	d	e	f	g	h	i	j
f	$\boxed{\boldsymbol{a}}$	i	g	j	h	e	$\boxed{\boldsymbol{c}}$	d	$\boxed{\boldsymbol{b}}$
h	d	i	$\boxed{\boldsymbol{b}}$	e	$\boxed{\boldsymbol{a}}$	j	g	f	$\boxed{\boldsymbol{c}}$
i	$\boxed{\boldsymbol{c}}$	j	d	$\boxed{\boldsymbol{b}}$	$\boxed{\boldsymbol{a}}$	h	f	e	g
g	d	f	e	b	a	h	j	c	i
d	j	h	$\boxed{\boldsymbol{c}}$	g	e	$\boxed{\boldsymbol{a}}$	i	f	$\boxed{\boldsymbol{b}}$
g	c	b	j	i	e	h	a	f	d
h	j	e	b	f	i	g	a	d	c
i	h	f	c	b	g	d	a	e	j
e	f	j	d	g	i	$\boxed{\boldsymbol{b}}$	$\boxed{\boldsymbol{c}}$	$\boxed{\boldsymbol{a}}$	h
e	d	c	j	i	f	h	g	a	b

- Each event is represented as an alphabet instead of an integer.
- Each row represents an event sequence.
- We highlight the different 3-sequences over $\{a, b, c\}$ to show all possible 3-sequences (six permutations) occur at least once.
- This property holds for all 3-sequences over $\{a, b, c, d, e, f, g, h, i, j\}$.

Figure 1 An optimal sequence covering array $SCA(11; 10, 3)$.

▶ **Notation 4.** Let s be an integer. $SCA(n; s, t)$ and $SCAN(s, t)$ are intended to denote, respectively, $SCA(n; \{1, \ldots, s\}, t)$ and $SCAN(\{1, \ldots, s\}, t)$.

Fig. 1 shows an example of $SCA(11; 10, 3)$, a sequence covering array of strength $t = 3$ with $s = 10$ events. It is an optimal sequence covering array which has $n = 11$ rows.

In this paper, we define two kinds of problems to make our approach more understandable. For a given tuple $\langle n, s, t\rangle$, *SCA decision problem* is the problem to decide whether an $SCA(n; s, t)$ exists or not, and find it if exists. For a given pair $\langle s, t\rangle$, *SCA optimization problem* is the problem to find an optimal covering array $SCA(n; s, t)$. Oetsch et al. have recently proved that the *Generalised Event Sequence Testing* (GEST) problem is NP-complete [1]. Most *SCA* decision problems studied in this paper are special cases of GEST.

Kuhn et al. proposed a greedy algorithm for solving the *SCA* optimization problems [19]. The practical effectiveness, especially scalability, of their algorithm has been shown by the fact that they succeeded in finding upper bounds for the arrays of strength $3 \leq t \leq 4$ with $s \leq 80$ events. We refer to their algorithm [19] as Kuhn's encoding.

Erdem et al. proposed ASP encodings and an ASP-based greedy algorithm for solving the *SCA* optimization problems [11]. They have found and proved optimal solutions for the arrays of strength $t = 3$ with $5 \leq s \leq 8$ events through their exact ASP encodings. Moreover, their ASP-based greedy algorithm that synergistically integrates ASP with a greedy method is designed to improve the scalability issue of the ASP encodings. We refer to their encodings [11] as Erdem's encoding. When we need to distinguish between their exact ASP encodings and greedy algorithm, we refer to the former as Erdem's exact encoding and the latter as Erdem's greedy encoding.

3 Constraint Programming Models

We propose three different CP models for solving the *SCA* decision problems: the naïve matrix model, the event-position matrix model, and the incidence matrix model. We assume throughout that we have an $SCA(n; s, 3)$, a sequence covering array of strength $t = 3$, for the sake of clarity. Note that our models can be extended in a straightforward way to the case of any strength $t \geq 3$. We also use a sequence covering array $SCA(6; \{a, b, c, d\}, 3)$ of Fig. 2 as a running example.

[1] Oetsch et al. personal communication

1	2	3	4
a	d	b	c
d	c	b	a
c	d	a	b
a	c	b	d
b	d	a	c
b	c	a	d

Figure 2 A sequence covering array $SCA(6; \{a, b, c, d\}, 3)$.

a	b	c	d
1	3	4	2
4	3	2	1
3	4	1	2
1	3	2	4
3	1	4	2
3	1	2	4

Figure 3 The event-position matrix of $SCA(6; \{a, b, c, d\}, 3)$ shown in Fig. 2.

3.1 Naïve Matrix Model

For a given SCA decision problem for $SCA(n; s, 3)$, the most direct model would be using an $n \times s$ (n rows and s columns) matrix of integer variables $m_{r,i}$ $(1 \leq r \leq n, 1 \leq i \leq s)$. The domain of each variable is $\{1, 2, \ldots, s\}$. This matrix identifies a sequence covering array itself. We also use the auxiliary binary variables $a_{r,(i,j,k),(p,q,u)}$ with $1 \leq r \leq n$, $1 \leq i < j < k \leq s$, $1 \leq p, q, u \leq s$, $p \neq q$, $p \neq u$, and $q \neq u$. The variable $a_{r,(i,j,k),(p,q,u)}$ is intended to denote $m_{r,i} = p$, $m_{r,j} = q$, and $m_{r,k} = u$ in the matrix.

A *global constraint* is a constraint that can specify a relation between an arbitrary number of variables [26]. In the naïve matrix model, we use the alldifferent constraint that is one of the best known and most studied global constraint in CP. The constraint alldifferent$(X_1, X_2, \ldots, X_\ell)$ ensures that the values assigned to the variables $X_1, X_2, \ldots, X_\ell$ must be pairwise distinct.

The constraints for $SCA(n; s, 3)$ are defined as follows.

- Permutation constraints:
$$\mathsf{alldifferent}(m_{r,1}, m_{r,2}, \ldots, m_{r,s}) \tag{1}$$
- Channeling constraints:
$$a_{r,(i,j,k),(p,q,u)} = 1 \Leftrightarrow (m_{r,i} = p) \wedge (m_{r,j} = q) \wedge (m_{r,k} = u) \tag{2}$$
- Coverage constraints:
$$\sum_{\substack{1 \leq r \leq n \\ 1 \leq i < j < k \leq s}} a_{r,(i,j,k),(p,q,u)} \geq 1 \tag{3}$$

where $1 \leq r \leq n$, $1 \leq i < j < k \leq s$, $1 \leq p, q, u \leq s$, $p \neq q$, $p \neq u$, and $q \neq u$.

The permutation constraints can be easily expressed by using alldifferent constraints of (1). That is, for every row, one alldifferent is enforced to ensure that every event in the range 1 to s occurs exactly once. The constraints (2) express the channeling constraints. The constraints (3) express the coverage constraints such that every 3-sequence of the events $\{1, \ldots, s\}$ occurs at least once in the matrix.

Note that the constraints of leftward arrows in (2) can be omitted. Even if they may be omitted, we can still get a solution. For any solution, the constraints (3) ensure that every 3-sequence of the events occurs at least once. For each such an occurrence, the corresponding entries (i.e. a 3-tuple of variables) of the matrix are derived from the constraints (2). The condition that each row is a permutation of the events is ensured by the constraints (1).

The drawback of this model is not only the number of instances required for the coverage constraints (3), but also the number of variables contained within each cardinality constraint in (3). We need in total ${}_sP_3$ cardinality constraints, and each of them contains $n\binom{s}{3}$ variables. To avoid this problem, we propose another matrix model, called the event-position matrix model.

	a	a	b	b	c	c	a	a	b	b	d	d	a	a	c	c	d	d	b	b	c	c	d	d
	b	c	a	c	a	b	b	d	a	d	a	b	c	d	a	d	a	c	c	d	b	d	b	c
	c	b	c	a	b	a	d	b	d	a	b	a	d	c	d	a	c	a	d	c	d	b	c	b
a d b c	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0
d c b a	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1
c d a b	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0
a c b d	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0
b d a c	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0
b c a d	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0

Figure 4 The incidence matrix of $SCA(6; \{a, b, c, d\}, 3)$ shown in Fig. 2.

3.2 Event-Position Matrix Model

We give another view of sequence covering arrays. For a given sequence covering array $A = (a_{ij})$, we can represent it by the *event-position matrix* of the array. The event-position matrix $B = (b_{ie})$ of A is defined so that $b_{ie} = j$ if $a_{ij} = e$. That is, the rows are the same as before but the columns are labeled with the distinct events, and each entry represents the position of its corresponding event. Fig. 3 shows the event-position matrix of $SCA(6; \{a, b, c, d\}, 3)$ shown in Fig. 2.

For a given SCA decision problem for $SCA(n; s, 3)$, in the event-position matrix model, we use again an $n \times s$ matrix of integer variables $x_{r,i}$ $(1 \leq r \leq n, 1 \leq i \leq s)$. It identifies an event-position matrix instead of a sequence covering array. The domain of each variable is $\{1, 2, \ldots, s\}$. We also use the auxiliary binary variables $y_{r,(i,j,k)}$ with $1 \leq r \leq n$, $1 \leq i, j, k \leq s$, $i \neq j$, $i \neq k$, and $j \neq k$. The variable $y_{r,(i,j,k)}$ is intended to denote $x_{r,i} < x_{r,j} < x_{r,k}$ in the event-position matrix.

The constraints for $SCA(n; s, 3)$ are defined as follows.

- Permutation constraints:

$$\mathsf{alldifferent}(x_{r,1}, x_{r,2}, \ldots, x_{r,s}) \tag{4}$$

- Channeling constraints:

$$y_{r,(i,j,k)} = 1 \Leftrightarrow (x_{r,i} < x_{r,j}) \wedge (x_{r,i} < x_{r,k}) \wedge (x_{r,j} < x_{r,k}) \tag{5}$$

- Coverage constraints:

$$\sum_{r} y_{r,(i,j,k)} \geq 1 \tag{6}$$

where $1 \leq r \leq n$, $1 \leq i, j, k \leq s$, $i \neq j$, $i \neq k$, and $j \neq k$.

The constraints (4) is the same as (1) of the previous model except that each argument represents the position of the event. The constraints (5) express the channeling constraints. The coverage constraints can be concisely expressed by the constraints (6). That is, for every 3-sequence (i, j, k) of the events, one cardinality constraint is enforced to ensure that there is at least one row r that satisfies the condition $x_{r,i} < x_{r,j} < x_{r,k}$. This means that we cover all possible 3-sequence.

The comparisons $x_{r,i} < x_{r,k}$ in (5) are clearly redundant and can be omitted, but we leave them because of efficiency improvements. The constraints of leftward arrows in (5) can be also omitted for the same reason as before.

3.3 Incidence Matrix Model

We now give yet another view of sequence covering arrays. For a given sequence covering array, we can represent it by the *incidence matrix* of the array. Each row is labeled with one row (i.e. an event sequence) of the array. Each column is labeled with one of all possible t-sequences of the events. The incidence matrix $C = (c_{ij})$ of $SCA(n; s, t)$ is a $(0, 1)$-matrix with n rows and ${}_sP_t$ columns such that $c_{ij} = 1$ if the t-sequence j is a sub-sequence of the event sequence i, and $c_{ij} = 0$ otherwise.

Fig. 4 shows the incidence matrix of $SCA(6; \{a, b, c, d\}, 3)$ shown in Fig. 2. Each row is labeled with one row of the $SCA(6; \{a, b, c, d\}, 3)$. Each of ${}_4P_3 = 24$ columns is labeled with one of all possible 3-sequences of the events $\{a, b, c, d\}$. The labels of the columns are written vertically. For example, the entry in the first row and first column is a 1 since "$a\ b\ c$" is a sub-sequence of "$a\ d\ b\ c$".

In contrast, on the incidence matrix, let us consider the constraints that must be satisfied for $SCA(6; \{a, b, c, d\}, 3)$. Each column has at least one 1 (coverage constraints). From a viewpoint of 3-combinations of the events $\{a, b, c, d\}$, there are $6 \times \binom{4}{3} = 24$ sub-matrices with one row and six columns. Each sub-matrix sharing the same three events in the columns has exactly one 1. Furthermore, for each row, such occurrences of 1's are consistent with each other in terms of the ordering of the events.

For a given SCA decision problem for $SCA(n; s, 3)$, in the incidence matrix model, we use an $n \times {}_sP_3$ matrix of binary variables $y_{r,(i,j,k)}$ with $1 \le r \le n$, $1 \le i, j, k \le s$, $i \ne j$, $i \ne k$, and $j \ne k$. We can express the permutation constraints by using only the $y_{r,(i,j,k)}$ variables, but it requires a large number of constraints that are very costly to deal with. To avoid this problem, we introduce the auxiliary binary variables $pr_{r,(i,j)}$ with $1 \le r \le n$, $1 \le i, j \le s$, and $i \ne j$. The variable $pr_{r,(i,j)}$ is intended to denote the event i precedes the event j in the row r.

The constraints for $SCA(n; s, 3)$ are defined as follows.

- Permutation constraints:
$$\big((pr_{r,(i,j)} = 1) \land (pr_{r,(j,k)} = 1)\big) \Rightarrow pr_{r,(i,k)} = 1 \tag{7}$$
$$\neg(pr_{r,(i,j)} = 1) \lor \neg(pr_{r,(j,i)} = 1) \tag{8}$$
$$(pr_{r,(i,j)} = 1) \lor (pr_{r,(j,i)} = 1) \tag{9}$$
- Channeling constraints:
$$y_{r,(i,j,k)} = 1 \Leftrightarrow (pr_{r,(i,j)} = 1) \land (pr_{r,(i,k)} = 1) \land (pr_{r,(j,k)} = 1) \tag{10}$$
- Coverage constraints:
$$\sum_r y_{r,(i,j,k)} \ge 1 \tag{11}$$

where $1 \le r \le n$, $1 \le i, j, k \le s$, $i \ne j$, $i \ne k$, and $j \ne k$.

The permutation constraints can be expressed by enforcing total ordering on the events: (7) for transitivity, (8) for asymmetry, and (9) for comparability (totality). Note that the constraints (7) can be replaced with the following arithmetic constraints (12), and the constraints (8) and (9) with (13).

$$pr_{r,(i,j)} + pr_{r,(j,k)} - pr_{r,(i,k)} \le 1 \tag{12}$$
$$pr_{r,(i,j)} + pr_{r,(j,i)} = 1 \tag{13}$$

The channeling constraints are expressed by the constraints (10) that are slightly modified to adjust the pr variables compared with (5). The coverage constraints (11) are the same as (6). The equations $pr_{r,(i,k)} = 1$ in (10) and the constraints of leftward arrows in (10) can be omitted for the same reason as before.

Table 1 Benchmark results of different CP models for $SCA(n; s, t)$.

n	s	t	Result	Incidence	Incidence	E-Position	E-Position	E-Position
					(lex-row)		(snake lex)	(double lex)
6	5	3	UNSAT	1.320	< 0.000	5.457	0.008	0.011
$\mathbf{7}^*$	5	3	SAT	< 0.000	< 0.000	0.005	0.010	0.010
7	6	3	UNSAT	1327.350	0.110	$T.O$	0.383	0.588
$\mathbf{8}^*$	6	3	SAT	0.005	0.006	0.012	0.016	0.022
7	7	3	UNSAT	1442.410	0.180	$T.O$	1.921	5.509
$\mathbf{8}^*$	7	3	SAT	0.008	0.015	0.032	0.077	0.027
7	8	3	UNSAT	$T.O$	0.390	$T.O$	8.280	28.870
$\mathbf{8}^*$	8	3	SAT	0.070	0.094	7.870	2.815	18.160
9	9	3	SAT	0.075	0.242	6.139	6.070	64.570
9	10	3	SAT	11.896	5.890	982.580	1188.240	$T.O$
10	11	3	SAT	0.047	0.052	59.670	30.216	67.031
10	12	3	SAT	0.046	0.456	774.338	117.258	$T.O$
10	13	3	SAT	0.980	0.371	$T.O$	$T.O$	$T.O$
10	14	3	SAT	5.546	25.880	$T.O$	$T.O$	$T.O$
10	15	3	SAT	541.480	443.012	$T.O$	$T.O$	$T.O$
11	16	3	SAT	89.580	107.334	$T.O$	$T.O$	$T.O$
11	17	3	SAT	62.560	$T.O$	$T.O$	$T.O$	$T.O$
12	18	3	SAT	3.603	3.830	$T.O$	$T.O$	$T.O$
12	19	3	SAT	2.851	18.840	$T.O$	$T.O$	$T.O$
12	20	3	SAT	22.500	180.256	$T.O$	$T.O$	$T.O$
12	21	3	SAT	1353.810	824.680	$T.O$	$T.O$	$T.O$
13	22	3	SAT	29.660	9.783	$T.O$	$T.O$	$T.O$
13	23	3	SAT	$T.O$	898.820	$T.O$	$T.O$	$T.O$
14	24	3	SAT	4.838	13.962	$T.O$	$T.O$	$T.O$
14	25	3	SAT	25.600	7.763	$T.O$	$T.O$	$T.O$
14	26	3	SAT	67.850	8.864	$T.O$	$T.O$	$T.O$
14	27	3	SAT	1126.390	251.660	$T.O$	$T.O$	$T.O$
14	28	3	SAT	$T.O$	641.320	$T.O$	$T.O$	$T.O$
15	29	3	SAT	127.470	18.955	$T.O$	$T.O$	$T.O$
15	30	3	SAT	673.210	190.200	$T.O$	$T.O$	$T.O$
17	40	3	SAT	771.990	$T.O$	$M.O$	$M.O$	$M.O$
23	5	4	UNSAT	$T.O$	0.046	$T.O$	3.554	4.980
$\mathbf{24}^*$	5	4	SAT	0.100	0.081	94.488	1.150	0.690
23	6	4	UNSAT	$T.O$	0.260	$T.O$	$T.O$	$T.O$
$\mathbf{24}^*$	6	4	SAT	0.230	0.460	376.184	$T.O$	$T.O$
38	7	4	SAT	$T.O$	40.390	$T.O$	$T.O$	$T.O$
47	8	4	SAT	$T.O$	688.400	$T.O$	$T.O$	$T.O$
52	9	4	SAT	$T.O$	51.950	$T.O$	$T.O$	$T.O$
58	10	4	SAT	341.420	659.830	$T.O$	$T.O$	$T.O$
65	11	4	SAT	$T.O$	159.330	$T.O$	$T.O$	$T.O$
69	12	4	SAT	$T.O$	243.590	$T.O$	$T.O$	$T.O$

4 Experiments

To evaluate the effectiveness of our CP models, we solve SCA optimization problems (35 problems in total) of strength $3 \leq t \leq 4$ with small to moderate sizes of events. For each problem, we solve multiple SCA decision problems of $SCA(n; s, t)$ with varying the value of n. Such decision problems contain both satisfiable and unsatisfiable problems and their optimal solutions exist on the boundaries.

For each SCA decision problem, we represent it by using our models with and without breaking the symmetries. More precisely, we apply the *lexicographic ordering constraints* for breaking the row symmetry in the naïve matrix model and the incidence matrix model. In the event-position matrix model, for breaking the row and column symmetry, we apply the *double lex* [12] and the *snake lex* [16] separately. In addition, we constrain every entry in the first row to be "1 2 ... s" for the naïve matrix model and the event-position matrix model with double lex. We note that applying these constraints for breaking the symmetries does not lose any solutions. For every model, we omit the constraints of leftward arrows in the channeling constraints.

For solving every model of each decision problem, we use a SAT-based CSP solver Sugar [2], an award-winning system in GLOBAL category (including global constraints such as alldifferent) of the 2008 and 2009 International CSP Solver Competitions. Sugar solves a finite linear CSP by encoding it into SAT and then solving the SAT-encoded problem by using an external SAT solver at the back-end. The SAT encoding that Sugar adopted is called the *order encoding* [24, 25]. It is efficient in the sense that unit propagation keeps the *bounds consistency* in original CSPs. We use MiniSat 2.2.0 (core) [10], Glucose 2 [1], and clasp 2.0.2 [13, 14] as back-end SAT solvers. The first two are efficient CDCL SAT solvers. The last one clasp is not only a state-of-the-art ASP solver, but also an efficient SAT solver. In particular, Glucose and clasp are award-winning solvers in the 2011 SAT Competition.

Table 1 shows the best CPU time in seconds of three SAT solvers for solving $SCA(n; s, t)$. We only shows our best lower and/or upper bounds of n for each SCA optimization problem. We use the symbol "$*$" to indicate that the value of n is optimal. The column "Result" indicates whether it is satisfiable (`SAT`) or unsatisfiable (`UNSAT`). The columns "Incidence" and "E-Position" indicate the incidence matrix model and the event-position matrix model respectively. Note that we exclude the results of the naïve matrix model from Table 1 since it was quite inefficient. All times were collected on Mac OS X with Intel Xeon 3.2GHz and 16GB memory. We set a timeout ("$T.O$") including the encoding time of Sugar to 1800 seconds for each SCA decision problem. The "$M.O$" indicates a memory error of SAT solvers.

We observe in Table 1 that the incidence matrix based models ("Incidence" and "Incidence with lex-row") are faster and much more scalable to the number of events than the event-position matrix based models ("E-Position", "E-Position with snake lex" and "E-Position with double lex"). "Incidence with lex-row" solved 39 SCA decision problems out of 41, rather than 31 of "Incidence", 14 of "E-Position with snake lex", 12 of "E-Position with double lex", and 11 of "E-Position". The main difference between two incidence matrix based models is that "Incidence with lex-row" were able to give solutions for 7 arrays of strength $t = 4$ not solved in timeout by "Incidence".

Our models reproduced and re-proved 4 previously known optimal solutions. Moreover, we found optimal solutions for $SCAN(5, 4)$ and $SCAN(6, 4)$. We also improved on previously known upper bounds [11, 19] for the arrays of strength $t = 3$ with $18 \leq s \leq 40$ events and strength $t = 4$ with $5 \leq s \leq 12$ events except $s = 10$.

5 An ASP Program of the Incidence Matrix Model

We present an ASP program of our best incidence matrix model. It is compact and faithfully reflects the original constraints of the incidence matrix model. Our program has $\binom{t}{2} + 5$ rules for the SCA decision problem of $SCA(n; s, t)$. For example, Fig. 5 shows the ASP program `sca3.lp` for $SCA(n; s, 3)$, which has only $\binom{3}{2} + 5 = 8$ rules. Note that this program can be extended in a straightforward way to the case of any strength $t \geq 3$. We use the syntax supported by the solver clasp and the grounder gringo [13, 14].

In Fig. 5, the first two rules `row(1..n)` and `col(1..s)` express that the row indices are integers in the range `1` to `n`, and the events are integers in the range `1` to `s`. The constants `n` and `s` are replaced with given values by a grounder. The third rule corresponds to the coverage constraints (11) where the predicate `y(R,I,J,K)` expresses the binary variable $y_{r,(i,j,k)}$. To express the coverage constraints, it uses special constructs called *cardinality expressions* of the form $\ell\{a_1, \ldots, a_k\}u$ where each a_i is an atom and ℓ and u are non-negative

[2] `http://bach.istc.kobe-u.ac.jp/sugar/`

```
% SCA(n;s,3)
row(1..n). col(1..s).

% coverage constraints
1{ y(R,I,J,K) : row(R) } :- col(I;J;K), I!=J, I!=K, J!=K.

% channeling constraints
pr(R,I,J) :- y(R,I,J,K).
pr(R,I,K) :- y(R,I,J,K).
pr(R,J,K) :- y(R,I,J,K).

% asymmetry & comparability constraints
1{ pr(R,I,J), pr(R,J,I) }1 :- row(R), col(I;J), I<J.

% transitivity constraints
pr(R,I,K) :- pr(R,I,J), pr(R,J,K), row(R), col(I;J;K), I!=J, I!=K, J!=K.
```

Figure 5 `sca3.lp`: An ASP program for $SCA(n;s,3)$.

integers denoting the lower bound and the upper bound of the cardinality expression. The third rule first generates a candidate for the incidence matrix, and then constrains a lower bound on the number of atoms is 1 for each column (i.e. each 3-sequence of the events). From the fourth to the sixth rule, the predicate `pr(R,I,J)` expresses the auxiliary binary variable $pr_{r,(i,j)}$. These three rules correspond to the constraints of rightward arrows in the channeling constraints (10). The seventh rule again uses cardinality expressions to express the asymmetry and comparability constraints (13). The transitivity constraints (7) are expressed by the last rule. The command "`gringo sca3.lp -c n=`n `-c s=`s `| clasp`" gives an answer set of an $SCA(n;s,3)$ decision problem. We can get a solution of the original problem by decoding the resulting answer set.

6 Comparison

We compare our ASP program with different approaches. We use Kuhn's benchmark set that consists of 62 SCA optimization problems for $SCAN(s,t)$ of strength $3 \leq t \leq 4$ with $5 \leq s \leq 80$ events. We execute our ASP program by using clasp 2.0.4 and gringo 2.0.5 to solve multiple SCA decision problems of $SCA(n;s,t)$ with varying the value of n for each optimization problem. All times were measured on Mac OS X with Intel Xeon 2.66GHz and 24GB memory. We set a timeout for clasp to 3600 seconds for each $SCA(n;s,t)$.

Table 2 shows the comparison results of different approaches on the best known upper bounds of $SCAN(s,t)$. Our comparison includes our ASP program with clasp, our CP models with Sugar, Erdem encoding [11], and Kuhn encoding [19]. We note that Erdem encoding is closely related to the event-position matrix model of our CP models. We highlight the best value of different approaches for each $SCAN(s,t)$. The symbol "-" is used to indicate that the result is not available in either our experiments or published literature.

In the case of strength $t = 3$, our ASP program with clasp were able to produce significantly improved bounds compared with those in Erdem greedy encoding and Kuhn encoding. The more events are considered, the more significant are the improvements. For example, when $s = 80$ events, it produced an array of $n = 24$ rows compared with 38 of Erdem and 42 of Kuhn. On average, it improved every bound of Erdem greedy encoding and Kuhn encoding by 10 and 9 rows respectively. Compared with Erdem exact encoding, our ASP program can be more scalable. In the case of strength $t = 4$, although not able to match Erdem greedy encoding for $SCAN(10,4)$ and $SCAN(20,4)$, our ASP program were able to improve every bound of Kuhn encoding for the arrays with $s \leq 23$ events by 19 rows on average.

Table 2 Comparison of different approaches on the best known upper bounds of $SCAN(s,t)$.

s	Our ASP with clasp		Our CP with Sugar		Erdem exact encoding [11]		Erdem greedy encoding [11]		Kuhn encoding [19]	
	$t=3$	$t=4$	$t=3$	$t=4$	$t=3$	$t=4$	$t=3$	$t=4$	$t=3$	$t=4$
5	**7**	**24**	**7**	**24**	**7**	–	–	–	8	29
6	**8**	**24**	**8**	**24**	**8**	–	–	–	10	38
7	**8**	40	**8**	**38**	**8**	–	–	–	12	50
8	**8**	**44**	**8**	47	**8**	–	–	–	12	56
9	**9**	53	**9**	**52**	**9**	–	–	–	14	68
10	**9**	59	**9**	58	**9**	–	11	**55**	14	72
11	**10**	**65**	**10**	**65**	**10**	–	–	–	14	78
12	**10**	73	**10**	**69**	**10**	–	–	–	16	86
13	**10**	**77**	**10**	–	**10**	–	–	–	16	92
14	**10**	**81**	**10**	–	**10**	–	–	–	16	100
15	**10**	**84**	**10**	–	**10**	–	–	–	18	108
16	**11**	**89**	**11**	–	**11**	–	–	–	18	112
17	**11**	**91**	**11**	–	**11**	–	–	–	20	118
18	**12**	**97**	**12**	–	–	–	–	–	20	122
19	**12**	**100**	**12**	–	–	–	–	–	22	128
20	**12**	105	**12**	–	–	–	19	**104**	22	134
21	**12**	**104**	**12**	–	–	–	–	–	22	134
22	**13**	**111**	**13**	–	–	–	–	–	22	140
23	14	**112**	**13**	–	–	–	–	–	24	146
24	**14**	–	**14**	–	–	–	–	–	24	**146**
25	**14**	–	**14**	–	–	–	–	–	24	**152**
26	**14**	–	**14**	–	–	–	–	–	24	**158**
27	**14**	–	**14**	–	–	–	–	–	26	**160**
28	**14**	–	**14**	–	–	–	–	–	26	**162**
29	**15**	–	**15**	–	–	–	–	–	26	**166**
30	**15**	–	**15**	–	–	–	23	**149**	26	166
40	**17**	–	**17**	–	–	–	27	**181**	32	198
50	**19**	–	–	–	–	–	31	–	34	**214**
60	**21**	–	–	–	–	–	34	–	38	**238**
70	**22**	–	–	–	–	–	36	–	40	**250**
80	**24**	–	–	–	–	–	38	–	42	**264**

7 Conclusion

In this paper, we considered the problem of finding optimal sequence covering arrays by ASP and CP. We presented three CP models from different viewpoints of sequence covering arrays. Among them, the incidence matrix model is efficient in the sense that an SCA can be represented by the incidence matrix of the array in which the coverage constraints of the given SCA can be concisely expressed. We presented a new ASP program that is compact and faithfully reflects the incidence matrix model. To evaluate the effectiveness of our ASP program, we solved Kuhn's benchmark set that consists of 62 SCA optimization problems for $SCAN(s,t)$ of strength $3 \leq t \leq 4$ with $5 \leq s \leq 80$ events. We were able to significantly improve the previously known bounds for many arrays, as shown in Table 2. Moreover, we found optimal solutions for $SCAN(5,4)$ and $SCAN(6,4)$. However, we were still not able to find any solutions for $SCAN(s,4)$ with $24 \leq s \leq 80$ events because of expensive grounding, which shows a limitation of our approach at present. To overcome this problem, hybrid approaches to SCA, like Erdem greedy encoding, can be promising.

Our approach is based on an effective combination of ASP solvers and the incidence matrix. It can be applied to a wide range of combinatorial search problems such as balanced incomplete block designs [9] and SAT-based standard combinatorial testing [2].

References

1 Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 399–404, 2009.

2 Mutsunori Banbara, Haruki Matsunaka, Naoyuki Tamura, and Katsumi Inoue. Generating combinatorial test cases by efficient SAT encodings suitable for CDCL SAT solvers. In *Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-17), LNCS 6397*, pages 112–126, 2010.

3 Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

4 D.A. Bulutoglu and F. Margot. Classification of orthogonal arrays by integer programming. *Journal of Statistical Planning and Inference*, 138:654–666, 2008.

5 M. A. Chateauneuf and Donald L. Kreher. On the state of strength-three covering arrays. *Journal of Combinatorial Designs*, 10(4):217–238, 2002.

6 David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, and Gardner C. Patton. The AETG system: An approach to testing based on combinatiorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444, 1997.

7 Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Transactions on Software Engineering*, 34(5):633–650, 2008.

8 Myra B. Cohen, Peter B. Gibbons, Warwick B. Mugridge, and Charles J. Colbourn. Constructing test suites for interaction testing. In *Proceedings of the 25th International Conference on Software Engineering (ICSE 2003)*, pages 38–48, 2003.

9 Charles J. Colbourn and Jeffrey H. Dinitz. *Handbook of Combinatorial Designs*. Chapman & Hall/CRC, 2007.

10 Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003), LNCS 2919*, pages 502–518, 2003.

11 Esra Erdem, Katsumi Inoue, Johannes Oetsch, Jorg Puhrer, Hans Tompits, and Cemal Yilmaz. Answer-set programming as a new approach to event-sequence testing. In Teemu Kanstrén, editor, *Proceedings of the 3rd International Conference on Advances in System Testing and Validation Lifecycle (VALID 2011)*, pages 25–34. Xpert Publishing Services, 2011.

12 Pierre Flener, Alan M. Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, Justin Pearson, and Toby Walsh. Breaking row and column symmetries in matrix models. In *Proceedings of the 8th International Joint Conference on Principles and Practice of Constraint Programming (CP 2002), LNCS 2470*, pages 462–476, 2002.

13 Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. Conflict-driven answer set solving. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 386–392. MIT Press, 2007.

14 Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. The conflict-driven answer set solver clasp: Progress report. In Esra Erdem, Fangzhen Lin, and Torsten Schaub, editors, *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2009), LNCS 5753*, pages 509–514. Springer, 2009.

15 Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pages 1070–1080. MIT Press, 1988.

16 Andrew Grayland, Ian Miguel, and Colva M. Roney-Dougal. Snake lex: An alternative to double lex. In *Proceedings of the 15th International Joint Conference on Principles and Practice of Constraint Programming (CP 2009), LNCS 5732*, pages 391–399, 2009.

17 Alan Hartman and Leonid Raskin. Problems and algorithms for covering arrays. *Discrete Mathematics*, 284(1–3):149–156, 2004.

18 Brahim Hnich, Steven David Prestwich, Evgeny Selensky, and Barbara M. Smith. Constraint models for the covering test problem. *Constraints*, 11(2-3):199–219, 2006.

19 D. Richard Kuhn, James M. Higdon, James F. Lawrence, Raghu N. Kacker, and Yu Lei. Combinatorial methods for event sequence testing. *submitted for publication*, 2010. Available at `http://csrc.nist.gov/groups/SNS/acts/documents/event-seq101008.pdf`.

20 Yu Lei and Kuo-Chung Tai. In-parameter-order: A test generation strategy for pairwise testing. In *Proceedings of 3rd IEEE International Symposium on High-Assurance Systems Engineering (HASE 1998)*, pages 254–261, 1998.

21 Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Mathematics and Artificial Intelligence*, 25(3–4):241–273, 1999.

22 Kari J. Nurmela. Upper bounds for covering arrays by tabu search. *Discrete Applied Mathematics*, 138(1-2):143–152, 2004.

23 Toshiaki Shiba, Tatsuhiro Tsuchiya, and Tohru Kikuno. Using artificial life techniques to generate test cases for combinatorial testing. In *Proceedings of 28th International Computer Software and Applications Conference (COMPSAC 2004)*, pages 72–77, 2004.

24 Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear CSP into SAT. In *Proceedings of the 12th International Joint Conference on Principles and Practice of Constraint Programming (CP 2006), LNCS 4204*, pages 590–603, 2006.

25 Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear CSP into SAT. *Constraints*, 14(2):254–272, 2009.

26 Willem Jan van Hoeve and Irit Katriel. Global constraints. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, pages 169–208. Elsevier, 2006.

27 Alan W. Williams. Determination of test configurations for pair-wise interaction coverage. In *Proceedings of 13th International Conference on Testing Communicating Systems (TestCom 2000)*, pages 59–74, 2000.

28 Hantao Zhang. Combinatorial designs by SAT solvers. In *Handbook of Satisfiability*, pages 533–568. IOS Press, 2009.