2H5-OS-03b-5

# 組合せテストケース生成問題に対する制約解集合プログラミングの適用

Applying Constraint Answer Set Programming to Combinatorial Test Case Generation Problems

兼行大将 *1 Hiromasa Kaneyuki
番原睦則 *2 Mutsunori Banbara
宋剛秀 *2 Takehide Soh
田村直之 *2 Naoyuki Tamura
井上克巳 *3 Katsumi Inoue

*1神戸大学大学院システム情報学研究科 Graduate School of System Informatics, Kobe University
*2神戸大学情報基盤センター Information Science and Technology Center, Kobe University
*3国立情報学研究所 National Institute of Informatics

Generating test cases for combinatorial testing is to find a covering array in Combinatorial Designs. In this paper, we consider the problem of finding covering arrays by Constraint Answer Set Programming (CASP). CASP is an extension of Answer Set Programming (ASP) with Constraint Programming. We present a new constraint modelling and study the effectiveness of CASP for the problem. Our modeling is designed to reduce the number of auxiliary variables compared with Hnich's ones. In our CASP encoding, the coverage constraints of covering array is expressed by the integrity constraints of ASP. We show that CASP can express the problem compactly, and can be flexible enough to incorporate several techniques such as search heuristics and symmetry breaking for constructing covering arrays efficiently.

## 1. はじめに

解集合プログラミング (Answer Set Programming; ASP) [Gelfond 88, 井上 08] は，一階述語論理に基づく表現力の高いモデリング言語と安定モデル意味論に基づいた解集合を求める ASP システムからなる．計算手続きはトップダウン方式の反駁手続きではなく，ボトムアップ方式によるモデル生成が基礎となる．記号を整数等で置換することなく直接扱うことができ，記号制約を簡潔に記述できる．近年，SAT 技術を応用した高速 ASP システムが実現され，人工知能分野の様々な問題への実用的応用が急速に拡大している．

制約解集合プログラミング (Constraint Answer Set Programming; CASP) は，ASP 言語に加え，制約プログラミング (Constraint Programming; CP) [Rossi 06] 言語のベースとなる制約充足問題 (Constraint Satisfaction Problems, CSP) を記述できるように言語拡張されている．これにより，記号制約に加え，有限ドメイン上の算術制約も簡潔に記述できる．CASP は ASP を表現力と効率性の両面から拡張する試みであり，2010 年以降活発に研究が進められている [Drescher 10, Ostrowski 12, Balduccini 13].

**組合せテスト** (Combinatorial Testing) はソフトウェア・ハードウェアのテスト手法の一つである．この手法の特長は，欠陥の多くは少数のパラメータの組合せによって発生するという観測を元に，テストケースの増大を回避し，現実的かつ効果的なテストを行える点である．組合せテストのテストケース生成問題は，組合せデザイン分野の**カバリング配列** (Covering Array, $CA$) を生成する問題に帰着できる [Colbourn 07, Zhang 09]. $CA$ 生成問題に対する主なアプローチとしては，群論等を用いた数学的手法，貪欲法，局所探索法，整数計画法，CP，SAT などがある．しかし，CASP を用いたアプローチは存在しない．

本稿では，$CA$ 生成問題に対する新しい制約モデル (CSP 表現) を提案し，CASP を用いたアプローチの有効性を検証する．提案する制約モデルは，既存モデル [Hnich 06] と比較して，導入する補助変数の数を少なく抑えることができる．また，その CASP 符号化は，$CA$ のカバレッジ制約を ASP の一貫性制約 (integrity constraint) を用いて簡潔に記述できる．

提案する制約モデルの有効性を評価するために，CASP システムを用いて，小中規模の $CA$ 生成問題に対する実行実験を行った．その結果，提案モデルは，既存モデルと比較して，より多くの問題をより高速に解くことができた．これにより，提案モデルが CASP に適していることが確認できた．

$CA$ 生成問題に対して CASP を用いる利点としては，(a) 問題を簡潔に記述できる表現力，(b) これまで他のアプローチで提案されたヒューリスティックス，対称性除去などの手法を取込むことができる柔軟性，(c) 実用的な組合せテストケース生成のために必要な水準の異なる因子・禁則処理 [Hartman 04] への拡張容易性が挙げられる．本稿では (a) を中心に，(b) についても具体例を用いてその利点を明らかにする．

連絡先: 兼行大将．神戸大学大学院システム情報学研究科．神戸市灘区六甲台町 1-1. 078-803-5365. kaneyuki@stu.kobe-u.ac.jp

## 2. カバリング配列

カバリング配列 $CA(b;t,k,v)$ とは，$b \times k$ 配列 ($b$ 行 $k$ 列) であり，各要素は $\{0,1,2,\ldots,v-1\}$ のいずれかの値を取る．そして，どの $t$ 個の列 ($1 \leq t \leq k$) についても，全部で $v^t$ 通りあるそれらの値の組合せすべてが少なくとも一つ出現する．ここでは，$t$ を強さ，$k$ を因子，$v$ を水準，$b$ を大きさと呼ぶことにする．以下の定義は，[Hnich 06] に基づいている．

**定義 1** カバリング配列 $CA(b;t,k,v)$ とは，以下の性質を満たす $b \times k$ 配列 $A = (a_{ij})$ である．

- $a_{ij} \in \mathbb{Z}_v = \{0,1,2,\ldots,v-1\}$
- 任意の異なる $t$ 個の列 $1 \leq c_1 < c_2 < \ldots < c_t \leq k$，および任意の値の組 $(x_1,x_2,\ldots,x_t) \in {\mathbb{Z}_v}^t$ に対して，$x_i = a_{rc_i} (\forall i; 1 \leq i \leq t)$ を満たす行 $r$ が少なくとも一つ存在する．これを**カバレッジ制約**と呼ぶ．

**定義 2** カバリング配列数 $CAN(t,k,v)$ とは，$CA(b;t,k,v)$ が存在する最小の $b$ である．

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

図 1: $CA(11;2,5,3)$

図 1 に，強さ $t=2$，因子 $k=5$，水準 $v=3$，大きさ $b=11$ のカバリング配列 $CA(11;2,5,3)$ の例を示す．最初の 2 列について，異なる値の組合せをボールド体で記している．全部で $3^2$ 通りある値の組合せすべてが，少なくとも一つ出現していることがわかる．他のどの 2 列の組み合わせについても同様の性質を満たす．組合せテストの観点から見た場合，各行が一つのテストケースに対応する．また，$CAN(2,5,3)=11$ であることが知られており，この例は，強さ $t=2$，因子 $k=5$，水準 $v=3$ に対する最小のカバリング配列である．

$CA$ **生成問題**とは，与えられた $t,k,v,b$ に対して，カバリング配列 $CA(b;t,k,v)$ が存在するかどうかを判定し，存在する場合は構成する問題である．Hnich らは，この $CA$ 生成問題に対して 2 つの制約モデル (CSP 表現) を提案している：基本モデルと結合モデル [Hnich 06]．本稿では，一般性を失うことなく，最も研究されている強さ $t=2$ のカバリング配列 $CA(b;2,k,v)$ について説明する．

基本モデルでは，$CA(b;2,k,v)$ の各要素を表す整数変数 $x_{r,i}(1\le r\le b, 1\le i\le k)$ を導入する．そのドメインは $x_{r,i}\in\{0,1,2,\ldots,v-1\}$ である．加えて，補助ブール変数 $covered_{r,(i,j)(d_1,d_2)}$ $(1\le r\le b, 1\le i<j\le k, 0\le d_1,d_2\le v-1)$ を導入する．このとき，カバレッジ制約は以下のように表される．

$$\bigvee_{1\le r\le b} covered_{r,(i,j)(d_1,d_2)} \tag{1}$$

$$covered_{r,(i,j)(d_1,d_2)} \Leftrightarrow x_{r,i}=d_1 \wedge x_{r,j}=d_2 \tag{2}$$

ただし，$1\le r\le b, 1\le i<j\le k, 0\le d_1,d_2\le v-1$．式 (1) は，任意の相異なる列の組 $(i,j)$，任意の値の組 $(d_1,d_2)$ に対して，$x_{r,i}=d_1$ かつ $x_{r,j}=d_2$ となる行 $r$ が少なくとも 1 つ存在することを表している．なお，式 (2) の左矢印は省略可能である [Banbara 10]．

結合モデルは，基本モデルで導入した整数変数 $x_{r,i}$ に加え，新しく整数変数 $y_{r,(i,j)}$ $(1\le r\le b, 1\le i<j\le k)$ を導入する．この $y_{r,(i,j)}$ は，カバリング配列の要素を表す変数の組 $(x_{r,i},x_{r,j})$ と対応しており，そのドメインは $y_{r,(i,j)}\in\{0,1,2,\ldots,v^2-1\}$ である．例えば，図 1 の $CA(11;2,5,3)$ の場合，$(x_{r,i},x_{r,j},y_{r,(i,j)}) \in \{(0,0,0),(0,1,1),(0,2,2),(1,0,3),(1,1,4),(1,2,5),(2,0,6),(2,1,7),(2,2,8)\}$ となる．カバレッジ制約は以下のように表される．

$$\bigvee_{1\le r\le b} y_{r,(i,j)}=w \tag{3}$$

$$y_{r,(i,j)}=v\cdot x_{ri}+x_{rj} \tag{4}$$

ただし，$1\le r\le b, 1\le i<j\le k, 0\le w\le v^2-1$．結合モデルは，カバレッジ制約が変数 $y_{r,(i,j)}$ 上の制約として表される点が基本モデルと異なる．

これらの制約モデルの有効性は，SAT ソルバー用いたアプローチにより，いくつかのカバリング配列数の上限の更新，既知の上限の最適性証明がなされたことから示されている [Hnich 06, Banbara 10]．

## 3. 提案する制約モデル

提案モデルは，Hnich らの制約モデルと同様，$CA(b;2,k,v)$ の各要素を表す整数変数 $x_{ri}\in\{0,1,2\ldots,v-1\}$ を用いる $(1\le r\le b, 1\le i\le k)$．加えて，補助ブール変数 $covered_{(i,j)(d_1,d_2)}$ $(1\le i<j\le k, 0\le d_1,d_2\le v-1)$ を導入する．このとき，カバレッジ制約は以下のように表される．

$$\bigwedge_{\substack{1\le i<j\le k\\ 0\le d_1,d_2\le v-1}} covered_{(i,j)(d_1,d_2)} \tag{5}$$

$$covered_{(i,j)(d_1,d_2)} \Leftrightarrow \bigvee_{1\le r\le b} (x_{r,i}=d_1 \wedge x_{r,j}=d_2) \tag{6}$$

ただし，$1\le i<j\le k, 0\le d_1,d_2\le v-1$．式 (6) より，補助ブール変数 $covered_{(i,j)(d_1,d_2)}$ は，相異なる列の組 $(i,j)$ に値の組 $(d_1,d_2)$ が出現するとき真となる．式 (5) は，任意の相異なる列の組 $(i,j)$，任意の値の組 $(d_1,d_2)$ に対して $covered_{(i,j)(d_1,d_2)}$ が真となる，すなわち，$x_{r,i}=d_1$ かつ $x_{r,j}=d_2$ となる行 $r$ が少なくとも 1 つ存在することを表している．提案モデルで導入される補助ブール変数は $\binom{k}{2}v^2$ 個であり，基本モデルの $b\binom{k}{2}v^2$ 個と比較して少なく抑えられる．

## 4. CASP 符号化

ASP 言語は Prolog 言語の拡張である一般拡張選言プログラムをベースとしている．詳細については文献 [井上 08] を参照していただきたい．本稿では，説明の簡略化のため，そのサブクラスである**標準論理プログラム** (Normal Logic Program; NLP) について簡単に説明する．NLP は以下の形式のルールの集合として構成される．

$$L_1 \texttt{ :- } L_2\texttt{,}\ldots\texttt{,}L_m\texttt{,not } L_{m+1}\texttt{,}\ldots\texttt{,not } L_n\texttt{.}$$

このルールは，「$L_2,\ldots,L_m$ がすべて成り立ち，$L_{m+1},\ldots,L_n$ のそれぞれが成り立たないならば，$L_1$ が成り立つ」と読む．ここで $n\ge m\ge 0$ であり，$L_i$ は正または負のリテラルで，`not` は**デフォルトの否定**，“`,`” は連言を表す．このルールにおいて “`:-`” の左部をルールの**ヘッド**，右側をルールの**ボディ**と呼ぶ．ボディが空のルールを**ファクト**と呼び，ファクトは “`:-`” を省略してよい．ヘッドが空のルールを**一貫性制約**と呼ぶ．Prolog 言語は，NLP のうちボディに `not` を含まない $(n=m)$ ものに相当する．

さらに，近年の ASP システムでは，組合せ問題を ASP で解くために便利な**アグリゲート** (aggregate) と呼ばれる表記法がいくつか用意されている．例えば，“$\texttt{1 }\{L_1\texttt{,}\ldots\texttt{,}L_k\}\texttt{ :- }Body\texttt{.}$” は，**個数制約**と呼ばれる代表的なアグリゲートの一例である．これは $Body$ が成り立つとき，$L_1,\ldots,L_k$ のうち 1 個以上のリテラルが成り立つことを表している．

近年広く普及している ASP システム gringo-4/clasp は，ASP 言語に加え，CASP への言語拡張が部分的に実装されている．CASP 中の CSP 部分は，順序符号化 [Tamura 09] によって ASP に符号化される．gringo-4/clasp では，有限領域上の整数変数を “`$`” 付きのリテラル，算術上の制約を “`$+`” (加算) “`$*`” (乗算) “`$=`” (等号) “`$<=`” (比較) などを用いて記述することができる．

```
1  row(1..b). col(1..k). dom(0..v-1).
2
3  % 整数変数
4  0 $<= $x(R,I) $<= v-1 :- row(R), col(I).
5
6  % カバレッジ制約
7  1 { covered(R,(I,J),(D1,D2)) : row(R) } :-
8          dom(D1;D2), col(I;J), I<J.
9
10 $x(R,I) $= D1 :- covered(R,(I,_),(D1,_)).
11 $x(R,J) $= D2 :- covered(R,(_,J),(_,D2)).
```

図 2: 基本モデルの CASP 符号化：$CA(\mathtt{b};2,\mathtt{k},\mathtt{v})$

```
1  row(1..b). col(1..k). dom(0..v-1).
2
3  % 整数変数
4  0 $<= $x(R,I) $<= v-1 :- row(R), col(I).
5  0 $<= $y(R,I,J) $<= v**2-1 :-
6          row(R), col(I;J), I<J.
7
8  % カバレッジ制約
9  1 { $y(R,I,J) $= GT : row(R) } :-
10         col(I;J), I<J, GT = 0..v**2-1.
11
12 $y(R,I,J) $= v $* $x(R,I) $+ $x(R,J) :-
13         row(R), col(I;J), I<J.
```

図 3: 結合モデルの CASP 符号化：$CA(\mathtt{b};2,\mathtt{k},\mathtt{v})$

図 2，図 3，図 4 に基本モデル，結合モデル，提案モデルの CASP 符号化を示す．すべての制約モデルが，6〜7 個のルールで簡潔に記述されてることがわかる．これら 3 つの符号化の大きな違いは，基本モデルと結合モデルでは，カバレッジ制約が個数制約で表現されているのに対し，提案モデルでは一貫性制約で表現されている点である．

提案モデルの CASP 符号化 (図 4) のみ説明する．1 行目は，与えられた $CA(\mathtt{b};2,\mathtt{k},\mathtt{v})$ に対して，行，列，値を表すリテラル `row(1..b)`, `col(1..k)`, `dom(0..v-1)` を導入している．ここで “..” は複数のリテラルを表す略記法であり，`row(1..b)` は，`row(1)`, `row(2)`,⋯, `row(b)` を表す．4 行目は，$CA$ の各要素を表す整数変数 `$x(R,I)` を導入している．9〜11 行目のルールは，任意の行 `R`，相異なる列の組 `(I,J)`，値の組 `(D1,D2)` に対して，$CA$ の `(R,I)` 成分が `D1` かつ `(R,J)` 成分が `D2` ならば `covered((I,J),(D1,D2))` を生成し，7〜8 行目の一貫性制約で，カバレッジ制約を満たすかどうかチェックしている．

```
1  row(1..b). col(1..k). dom(0..v-1).
2
3  % 整数変数
4  0 $<= $x(R,I) $<= v-1 :- row(R), col(I).
5
6  % カバレッジ制約
7  :- not covered((I,J),(D1,D2)),
8          dom(D1;D2), col(I;J), I<J.
9  covered((I,J),(D1,D2)) :-
10         $x(R,I) $= D1, $x(R,J) $= D2,
11         row(R), dom(D1;D2), col(I;J), I<J.
```

図 4: 提案モデルの CASP 符号化：$CA(\mathtt{b};2,\mathtt{k},\mathtt{v})$

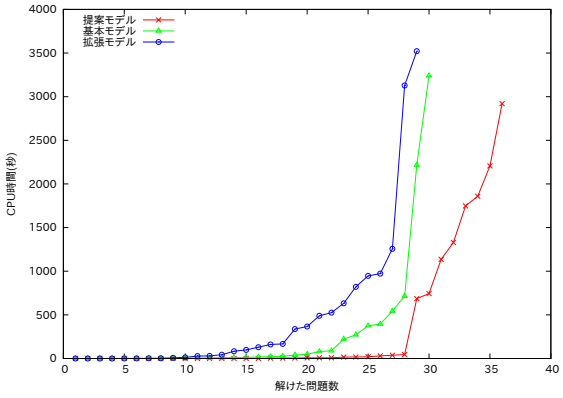


図 5: 制約モデルの比較：カクタスプロット

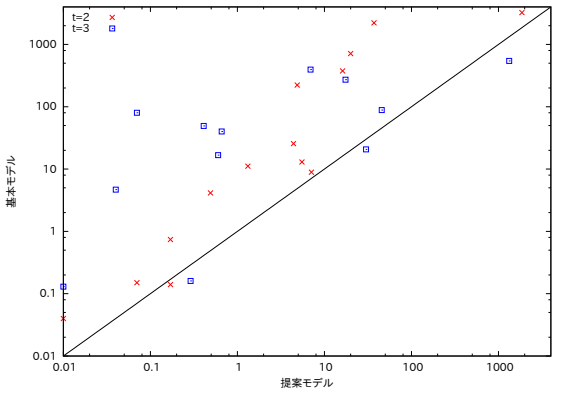


図 6: 提案モデルと基本モデルの比較：散布図 (対数スケール)

## 5. 実験

提案する制約モデルの有効性を評価するために，前節で述べた提案モデル，および Hnich の基本モデルと結合モデルの CASP 符号化を用いて，比較実験を行った．

ベンチマーク問題には，強さ $2 \le t \le 3$ の $CA(b;t,k,v)$ 生成問題 78 問を用いた ($t=2$ が 50 問，$t=3$ が 28 問)．大きさ $b$ には，様々な既存アプローチによって得られた既知のカバリング配列数 $CAN(t,k,v)$ または最良の上限を使用した [*1]．実行には，CASP に対応した ASP システム gringo-4.4.0/clasp-3.1.1 を使用し，タイムアウトは 3,600 秒とした．実験環境は Linux マシン (Xeon 3.16GHz，32GB メモリ) である．

各制約モデルで解けた問題数は，提案モデルが 36 問，基本モデルが 30 問，結合モデルが 29 問であり，提案モデルが最も多くの問題を解くことができた．図 5 にカクタスプロットを示す．横軸は解けた問題数を，縦軸は昇順に並び替えた clasp の CPU 時間を示している．すなわち，この表はグラフが右にあるほど多くの問題を解き，下にあるほど高速に求解していることを示す．図 5 から，提案モデルが，基本モデルと結合モデルに比べて，高速に求解していることがわかる．

図 6 に基本モデルと提案モデルの CPU 時間を縦軸と横軸にとった散布図を示す (両軸とも対数スケール)．プロットが左上に集中していることから，個々の問題を見ても，提案モデルが基本モデルよりも高速に求解していることがわかる．なお，結合モデルと比較しても，提案モデルが優れていた．以上のことから，提案モデルは，Hnich の基本モデル，結合モデルと比

*1 http://www.public.asu.edu/~ccolbou/src/tabby/catable.html

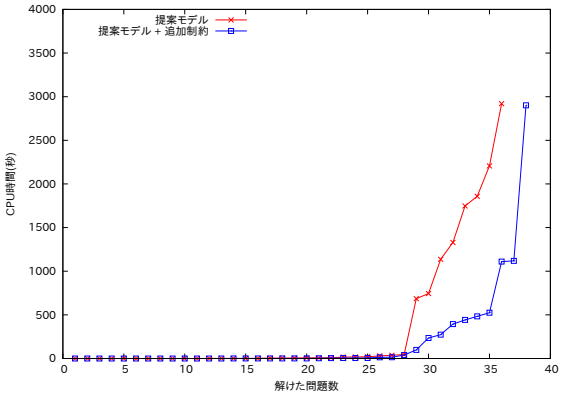


図 7: 追加制約の効果：カクタスプロット

較して，CASP に適していることが確認できた．

$CA$ 生成問題に対して CASP を用いる利点の一つは，これまで他のアプローチで提案されたヒューリスティックス，対称性除去などの手法を柔軟に取込むことができる点である．本稿では，以下の簡単な例を用いて説明を行う．

(1) $CA$ の 1 行目をすべて 0 に固定
(2) $CA$ を 1 行目から順に行単位で生成

カバリング配列は対称性が高く，対称解が多く存在する．制約プログラミングでは，これらの対称性を除去する方法の研究が進んでいる．なかでも (1) は最も簡単な対称性除去手法の一つである．これ以外にも “行と列に関する対称性”，“値に関する対称性” を除去する手法が提案されている [Hnich 06]．なお，対称解を除去しても，解の存在性は保たれる．貪欲法は，問題の規模に関係なくある程度良質なカバリング配列を効率よく生成することができる．(2) は貪欲法でよく用いられるヒューリスティックスの一つである [Cohen 97]．

```
1 $x(1,I) $= 0 :- col(I).
2
3 _heuristic(covered(R,(I,J),(D1,D2)),init,b-R) :-
4         row(R), col(I;J), I<J, dom(D1;D2).
5 covered(R,(I,J),(D1,D2)) :-
6         $x(R,I) $= D1, $x(R,J) $= D2,
7         row(R), dom(D1;D2), col(I;J), I<J.
```

図 8: (1) と (2) に対応するコード

CASP では (1) と (2) を，図 8 のように簡潔に記述できる．1 行目のルールは (1) に対応し，任意の列 `I` に対して，$CA$ の `(1,I)` 成分を 0 に固定している．3 行目〜7 行目のルールは，(2) に対応している．述語 `_heuristic(`$L$`,init,`$N$`)` は，リテラル $L$ の真偽値割当の優先度の初期値を $N$ にすることを表す (通常のリテラルの初期値は 1)．3 行目〜4 行目のルールでは，任意の行 `R`，相異なる列の組 `(I,J)`，値の組 `(D1,D2)` に対して，リテラル `covered(R,(I,J),(D1,D2))` の初期値を `b-R` (`b` は $CA$ の大きさ) に設定することにより，(2) のヒューリスティックスを模擬的に実現している．

提案モデルの CASP 符号化に，図 8 を追加制約として加えた結果，提案モデルよりも 2 問多い 38 問を解くことができた．また，図 7 から，追加制約を加えることにより，求解速度が向上していることが確認できる．

## 6. まとめ

本稿では，$CA$ 生成問題に対して，新しい制約モデルを提案し，CASP を用いたアプローチの有効性を検証した．まず最初に，CASP を用いて，提案した制約モデルおよび Hnich らの制約モデルが簡潔に記述できることを示した．小中規模の $CA$ 生成問題 78 問を用いて実行実験を行った結果，提案した制約モデルは，Hnich らの基本モデル，結合モデルと比較して，より多くの問題をより高速に求解することができ，CASP に適していることが確認できた．次に，簡単な追加制約の例を通して，他のアプローチで提案されたヒューリスティックス，対称性除去手法を柔軟に取込めることを示した．

今後の課題は，$CA$ 生成問題への有効性が知られているより複雑なヒューリスティックス，対称性除去手法を導入し，SAT 等の他のアプローチとの性能比較を行うことである．提案手法を，水準の異なる因子・禁則処理に対応した実用的な組合せテストケース生成問題へ拡張することも今後の課題の一つである．

## 参考文献

[Balduccini 13] Balduccini, M. and Lierler, Y.: Integration Schemas for Constraint Answer Set Programming: a Case Study, *TPLP*, Vol. 13, No. 4-5-Online-Supplement (2013)

[Banbara 10] Banbara, M., Matsunaka, H., Tamura, N., and Inoue, K.: 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*, pp. 112–126 (2010)

[Cohen 97] Cohen, D. M., Dalal, S. R., Fredman, M. L., and Patton, G. C.: The AETG System: An Approach to Testing Based on Combinatiorial Design, *IEEE Transactions on Software Engineering*, Vol. 23, No. 7, pp. 437–444 (1997)

[Colbourn 07] Colbourn, C. J. and Dinitz, J. H.: *Handbook of Combinatorial Designs*, Chapman & Hall/CRC (2007)

[Drescher 10] Drescher, C. and Walsh, T.: A translational approach to constraint answer set solving, *TPLP*, Vol. 10, No. 4-6, pp. 465–480 (2010)

[Gelfond 88] Gelfond, M. and Lifschitz, V.: The Stable Model Semantics for Logic Programming, in *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pp. 1070–1080, MIT Press (1988)

[Hartman 04] Hartman, A. and Raskin, L.: Problems and Algorithms for Covering Arrays, *Discrete Mathematics*, Vol. 284, No. 1–3, pp. 149–156 (2004)

[Hnich 06] Hnich, B., Prestwich, S. D., Selensky, E., and Smith, B. M.: Constraint Models for the Covering Test Problem, *Constraints*, Vol. 11, No. 2-3, pp. 199–219 (2006)

[井上 08] 井上 克巳, 坂間 千秋：論理プログラミングから解集合プログラミングへ, コンピュータソフトウェア, Vol. 25, No. 3, pp. 20–32 (2008)

[Ostrowski 12] Ostrowski, M. and Schaub, T.: ASP modulo CSP: The clingcon system, *TPLP*, Vol. 12, No. 4-5, pp. 485–503 (2012)

[Rossi 06] Rossi, F., Beek, P. v., and Walsh, T.: *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*, Elsevier Science Inc., New York, NY, USA (2006)

[Tamura 09] Tamura, N., Taga, A., Kitagawa, S., and Banbara, M.: Compiling Finite Linear CSP into SAT, *Constraints*, Vol. 14, No. 2, pp. 254–272 (2009)

[Zhang 09] Zhang, H.: Combinatorial Designs by SAT Solvers, in *Handbook of Satisfiability*, pp. 533–568, IOS Press (2009)