

シーケンスカバリングアレイ

2010/11/26 (金)

秋山 浩一

1. はじめに

2010年11月3日に、辰巳さんからTEFのメーリングリストに一通のメールが発信されました。それは、NISTのウェブに掲載された、D. Richard Kuhn, Raghu N. Kacker, Yu Leiの3名による、組合せテスト技法のチュートリアル資料『PRACTICAL COMBINATORIAL TESTING』についての情報でした。

<http://csrc.nist.gov/groups/SNS/acts/documents/SP800-142-101006.pdf>

辰巳さんのメールには上記資料の紹介があり、最後に、チュートリアルの第5章“SEQUENCE-COVERING ARRAYS”について、「これはいいかも、と思いました。組み込み系ソフトウェアのテストでは、このようなテスト設計が必要そうに思いました」と結んでいました。

その翌々日、辰巳さんのメールを受けて、にしさんから、「あまり議論されてなかった気がしますし、困っている方も結構いらっしゃると思います」と返信が出ていましたが、それ以降TEFでその話題は途切れてしまいました。

そこで、今回は、sequence-covering arrayについて書こうと思います。なお、sequence-covering arrayは、HAYST法を使う上でも必要となる重要な関連技術の一つです。

2. Sequence-covering array とは？

さて、そもそもsequence-covering arrayとは何のことでしょう。実は、私も『PRACTICAL COMBINATORIAL TESTING』を読むまで、この用語は知りませんでした。ただ、内容を理解したところHAYST法を現場に適用する際に必ずといっていいほど質問を受けて対応してきたものでした。

sequence-covering arrayを日本語に直訳すると「順序被覆配列」となります。今、Googleで検索してみたのですが「シーケンスカバレッジアレイ」と「順序被覆配列」のどちらについても、1件もヒットしませんでしたので、新しい用語と考えてよさそうです。

被覆とは、一般的には覆いかぶせるという意味ですが、数学の世界では「ある集合がその集合の部分集合の族で覆われるとき、その部分集合の族のことをいう。」となっています (Wikipedia 「被覆 (数学)」より)。

まだよくわかりませんよね。ところで、sequence を取った covering array って何のことだと思いますか？ 実は、ペアワイズや All-pairs 法で作成した表のことを、それが有する性質から covering array (被覆配列) と呼びます。こちらは、Google でも 19 件ヒットしました。

HAYST 法で作った表が直交表であり orthogonal array (直交配列) と呼ぶのと同じ対応構造です。これらをまとめると、表 1 のようになります。

表 1 Covering array と Orthogonal array

		ペアワイズ系	直交表系
技法や別名		All-pair 法、k-way	直交表、HAYST 法
ツール		PICT, ACTS	MatrixTester
数学的な呼び方	英語	covering array	orthogonal array
	日本語	被覆配列	直交配列

covering array は、全ての組合せを網羅しているわけではありません。任意の k 個 (All-pair 法なら任意の 2 個) の因子を取り出したときの水準の組合せを全て生成することで、総当たりの組合せ全体をカバーした (粗く覆った) 集合になっています。表 2 は、PICT というペアワイズ系のツールで作成した covering array の例です。

表 2 Covering array の例

No	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	1	1	1	1	1	1	0	1	1	0	0	0	1	1
2	1	0	0	0	0	0	0	1	0	0	1	1	1	0	0
3	1	1	1	0	1	0	1	1	0	1	1	0	0	0	0
4	0	0	0	1	0	1	0	0	1	0	1	1	1	1	1
5	1	1	0	1	1	0	0	0	0	0	0	0	1	1	0
6	0	0	1	0	0	1	1	1	1	0	0	1	0	0	1
7	1	1	1	1	0	0	0	1	1	1	1	0	0	1	0
8	1	1	0	0	1	1	1	0	0	1	0	1	1	0	1
9	0	0	0	1	0	1	1	1	1	1	0	0	0	0	0
10	0	0	1	0	1	0	0	0	0	1	0	1	1	1	1

表 2 では、どの 2 列を取り出しても必ず 0 と 1 との全ての組合せ、すなわち、{00, 01, 10, 11} の組合せが入っています。試しに、1 列目と 2 列目で確認すると、00 の組合せは 4, 6, 9, 10 行目に、01 の組合せは 1 行目に、10 の組合せは 2 行目に、11 の組合せは 3, 5, 7, 9 行目に出現しています。covering array への出現回数は、{00, 01, 10, 11} = {4 回, 1 回, 1 回, 4 回} と異なりますが、1 回も出現しないということはありません。これが、部分集合で全体が覆われている (= 被覆している) 状態です。

さて、covering array が理解できたところで、本題の sequence-covering array に話を戻します。sequence が追加されただけなので、順序を扱う被覆配列であることが想像できると思います。

ここで言っている順序とは因子を操作する順序のことです。洗濯機のテストを例に考えてみましょう。洗濯機に与える入力 (洗濯の設定)、すなわち因子には、水量 (10 リットル、20 リットル、30 リットル、40 リットル)、水洗時間 (10 分、15 分)、すすぎ回数 (1 回、2 回)、脱水時間 (10 分、15 分)、乾燥時間 (なし、90 分) 等々があります。この組合せテストを HAYST 法で直交表に組むと表 3 のようになります。

表 3 洗濯機のテスト (HAYST 法)

	水量	水洗時間	すすぎ回数	脱水時間	乾燥時間
1	10 リットル	10 分	1 回	10 分	なし
2	10 リットル	15 分	2 回	15 分	90 分
3	20 リットル	10 分	1 回	15 分	90 分
4	20 リットル	15 分	2 回	10 分	なし
5	30 リットル	10 分	2 回	10 分	90 分
6	30 リットル	15 分	1 回	15 分	なし
7	40 リットル	10 分	2 回	15 分	なし
8	40 リットル	15 分	1 回	10 分	90 分

表 3 をテストするときには、水量をセットし、次に水洗時間、すすぎ回数、脱水時間、乾燥時間と順番に値をセットすると思います。しかし、操作順序を変えて、脱水時間、すすぎ回数、水量、乾燥時間、水洗時間の順番でセットすることもできます。せっかく 8 回もテストするわけですから、操作順序を変えてテストしてみたいと思いませんか？ 操作順序を変えてもテスト時間にはそれほど影響がありません。そうであれば工数を増やさずに何か新たなバグを発見することが期待できるので操作順序を変えることには、メリットがありそうです。

さて、それではどのような操作順序でテストしたらよいのでしょうか？

表 3 の例では、わずか 5 つの因子ですが、その操作順序にはかなりのパターンがありそうです (実は、120 通りもあります)。そこで、全ての操作順序 (120 通りの順序) をテストするのではなく、バグ検出に効果的な部分集合を見つけて全体をカバーしようというのが、sequence-covering array です。

3. Sequence-covering array の考え方

まず、因子の操作順序の数について確認しておきましょう。このとき、因子の重複を許さないとします。つまり、同じ設定を繰り返さないということにします。繰り返しを許してしまうと順序はさらに複雑になりテストケースは増加します。

さて、2 つの因子 a、b があった場合、その順序は、ab と ba の 2 通りです。次に 3 つの因子 a、b、c の場合は、abc、acb、bac、bca、cab、cba の 6 通りになります。4 つの因子 a、b、c、d の場合は、abcd、abdc、acbd、acdb、adbc、adcb、bacd、badc、bcad、bcda、bdac、bdca、cabd、cadb、cbad、cbda、cdab、cdba、dabc、dacb、dbac、dbca、dcab、dcba の 24 通りになります。5 つの因子は、 $5! = 120$ 通りもあるので例示することはやめておきます。

一般に、n 個の異なった要素の中から n 個全てを選び出した順列の個数は、

$${}_n P_n = n! / (n-n)! = n! = n(n-1)(n-2) \cdots 3 \times 2 \times 1 \quad (1)$$

となることが分かっています。

上の例では、2 つの因子の場合は $2! = 2 \times 1 = 2$ 、3 つの因子の場合は $3! = 3 \times 2 \times 1 = 6$ 、4 つの因子の場合は $4! = 4 \times 3 \times 2 \times 1 = 24$ とあっています。因子が 5 つになったら 120 通り、6 つになったら 720 通り、7 つで 5,040 通り、因子が 10 個になったら 3,628,800 通りの操作順序があるということです。

これを 2 章で確認した covering、すなわち被覆の概念でうまく減らしてやろうというわけです。いくつかの covering のアイデアが考えられますが、Kuhn は、因子が増えても全ての因子の順列を考えるのではなく、任意の k 個の因子を取りだしたときの順列が全て現れるという配列を作るという方法を取っています。

まず、k=2 の時を考えてみます。このとき因子は、a、b、c、d、e、f の 6 つとします。k=2 というのは、因子 a から因子 f の 6 つの因子の中からどの 2 つの因子を取りだしたとしても ab、ba のように因子の順序について前後関係が出現する並び方の部分集合を集めるということです (間に別の因子が入ることもある)。

これは、簡単で、abcdef と、それを逆順に並べた fedcba の 2 つの操作順序をテストをしてやればよいことが分かります。何故なら、abcdef と fedba の 2 つには、ab と ba、ac と ca、 \dots ef と fe までの 30 通りが全て含まれているからです。

実際のテストは表 4 の通りになります (1 列目から 6 列目の順に因子に値をセットする)。これを 2-way permutations と呼びます。permutation とは、順列 (順序の変化) という意味です。

表 4 2-way permutations

	1	2	3	4	5	6
1	a	b	c	d	e	f
2	f	e	d	c	b	a

次に、 $k=3$ の時を考えましょう。因子は同じく a、b、c、d、e、f の 6 つとします。 $k=3$ というのは、因子 a から因子 f の 6 つの因子の中からどの 3 つの因子を取り出したとしても abc、acb、bac、bca、cab、cba のような 6 通り (3!通り) の組合せが出現する並び方ということです

実際のテストは表 5 になります。これを 3-way permutations と呼びます。

表 5 3-way permutations

	1	2	3	4	5	6
1	a	b	c	d	e	f
2	f	e	d	c	b	a
3	d	e	f	a	b	c
4	c	b	a	f	e	d
5	b	f	a	d	c	e
6	e	c	d	a	f	b
7	a	e	f	c	b	d
8	d	b	c	f	e	a
9	c	e	a	d	b	f
10	f	b	d	a	e	c

表 5 を良く見ると、どの 3 因子を抜き出しても 6 通りのパターンが全て出現していることが確認できます。したがって、この表のとおりテスト 1 では abcdef、テスト 2 では fedcba、テスト 3 では defabc の順番で因子に値をセットしながらテスト 10 まで実施すれば 3-way permutations が網羅されたテストを実施したことになります。

4. Sequence-covering array の作成方法 (入手方法)

残念ながら、最小の sequence-covering array を生成するアルゴリズムは今のところ開発されていません。しかし、第1章で紹介したチュートリアルの作者である Kuhn が 'quick and dirty' greedy algorithm というアルゴリズムで生成した表 (表 5 のアルファベットを数値に置き換えたもの) が、次の URL 先のウェブに CSV ファイルの形式で置かれています。

http://csrc.nist.gov/groups/SNS/acts/sequence_cov_arrays.html

大きさは、3-way permutations で因子数 5 から 100、4-way permutations で、因子数 5 から 60 です。通常のテストで使用する分には十分と思います。CSV ファイルを開くと、数字だけの表が入っています。1 行目から、順序を表していることに注意してください。たとえば、一番小さな test3_5.csv というファイルには表 6 が入っています。実際に使用する際には、表 7 のように、因子名が入る行と、テスト項番が入る列を追加してください。

表 6 test3_5.csv

0	1	2	3	4
4	3	2	1	0
3	0	4	1	2
2	4	0	3	1
1	0	4	3	2
2	1	3	4	0
4	0	2	1	3
3	1	2	0	4

表 7 test3_5.csv に因子名とテスト項番を追加したもの

	1	2	3	4	5
1	0	1	2	3	4
2	4	3	2	1	0
3	3	0	4	1	2
4	2	4	0	3	1
5	1	0	4	3	2
6	2	1	3	4	0
7	4	0	2	1	3
8	3	1	2	0	4

5. Sequence-covering array で見つかるバグ

それでは、sequence-covering array を用いたテストではどのようなバグが見つかるのでしょうか。まず、abcdef と fedcba のように順序を入れ替えた、2-way permutations テストでは、変数の初期化忘れや過剰初期化といった問題が見つかることが期待できます。

たとえば、因子 a をセットするとき全ての変数を初期化するルーチンが動くように作ってあったとします。この場合、因子 a を先頭にしてテストした場合には全く問題なく動作します。ところが、最後に因子 a を実行した場合には、他の因子がセットした変数値を初期化ルーチンが破壊してしまい、それが不具合となって現れます。

3-way や 4-way はどうでしょうか？ 私自身は 3-way permutations テストまでしか実施したことがありません。また、その時に順序を考慮した因子は 9 個でした。greedy algorithm などはありませんでしたので、数独 (81 個の升目を 1 から 9 の数値で埋めるゲーム) の答えの表に足りない数値を足して 3-way permutations 表を作ってテストしました。数独の場合、1 から 9 の数値がよい感じにばらけているため具合が良かったのです。

結果はというと残念ながらそのテストではバグは見つかりませんでした。そもそも、順序にまつわるバグは、それほど多くあるタイプのバグではないので、仕方ないことかもしれません。しかし、順序関係が重要なテストの場合、使う価値は十分にあると考えています。

6. Sequence-covering array の使用場面

さて、sequence-covering array の実際の使用場面はどういったものでしょうか。まず、状態遷移図のイベントの順番といったテストには使用できません。何故なら、状態遷移図ではある状態で与えることが可能なイベントが限定されているからです。ab の順番でイベントを与えることができても ba の順でイベントが来たときの動作が定義できるとは限らないからです。

そうではなく、因子と因子の関連性が薄く、どの因子に対してもいつでも水準を設定できるタイプのテストに使用することができます。

つまり、直交表やペアワイズのテストを行うときに、順序が関係しそうな因子群に対して、sequence-covering array で「順序因子」を作ってやってそれも含めて表に割りつけるようにすればよいのです。

表 8 は、表 3 に sequence-covering array の考え方で作成した「操作順序」列を追加しています。1 番目のテストは順番 (abcde) に、2 番目のテストは逆順

(edcba) に、3 番目のテストは baedc の順番でというようにテストすることで、任意の 3 つの因子の操作順序を全てカバーしたテストを行うことができます。

表 8 洗濯機のテスト (HAYST 法 + sequence-covering array)

	水量 a	水洗時間 b	すすぎ回数 c	脱水時間 d	乾燥時間 e	順序因子
1	10 分	10 分	1 回	10 分	なし	abcde
2	10 分	15 分	2 回	15 分	90 分	edcba
3	20 分	10 分	1 回	15 分	90 分	baedc
4	20 分	15 分	2 回	10 分	なし	ceabd
5	30 分	10 分	2 回	10 分	90 分	deabc
6	30 分	15 分	1 回	15 分	なし	cdbae
7	40 分	10 分	2 回	15 分	なし	aecdb
8	40 分	15 分	1 回	10 分	90 分	bdcea

もし、直交表の大きさが十分に大きければ順序因子を直交表の外側に連結するのではなく、他の因子と同様に、直交表自体に因子として割り付けてしまった方が良いでしょう。そうすることで、操作順序を変えながら水準も同時に変化するからです (sequence-covering array は因子として扱うには大きいので直交表の外側に連結する表 8 のタイプの使い方が多くなると思いますが)。

たとえば、直交表が、 L_{64} であれば、5 つの因子の操作順序は 8 水準ですので、各操作順序は 8 回ずつ出現します。因子・水準の割り付けには、直交表を使用せずにペアワイズ系のツールを使用しても構いません。ペアワイズにおいても同様のことがいえます。

7. 今後に向けて

sequence-covering array を使用することで、これまで、因子数の階乗で考えていた操作順序について、現実的にテスト可能なテストケース数の範囲で粗くカバーすることができるようになりました。

デジタルカメラの撮影条件の設定や、携帯電話の操作など、案外「どの順番でも操作可能」というユーザインターフェイスは多いものです (もちろん、銀行の ATM のように一方向にしか操作できないもの……必ずカードを入力してからでないと引き出し金額を入力できない……もありますが)。

これまでテスト仕様書に「色々な順番で操作してみてください」と書いたことがあると思います。それらに対して、sequence-covering array は操作順序の網羅的なテストを提供します。

網羅的であるという事は2-way、3-way といった品質基準で操作順序の品質を保証できるという事です。つまり、操作順序について「どこまでテストしたのか？」という問いに対する論理的な答えを出すことができます。また、sequence-covering array でテストした結果を蓄積することで「このテスト(2-way, 3-way……) で十分確認したと言えるのか？」という問いに対しても自信を持って答えることができるようになるでしょう。

ところで、sequence-covering array は、あくまでも被覆なので被覆自体の分布については考慮されていません。したがって、ペアワイズに組合せの濃淡があるように、sequence-covering array では、並び順に濃淡があります。

先に紹介した Kuhn のウェブにある test.3_100csv において、0 から 2 の並び順の個数について調べたところ、

$\{012, 021, 102, 120, 201, 210\} = \{9 \text{ 個}, 7 \text{ 個}, 6 \text{ 個}, 7 \text{ 個}, 6 \text{ 個}, 9 \text{ 個}\}$
であり、6 個～9 個の間でばらついていました。

こちらについて、部分集合でカバーする方法を covering array (被覆配列) ではなく orthogonal array (直交配列) タイプで順序表を作成すれば、順序の数は若干増えるものの、順序の出現回数のばらつきについては、解消されるのではないかと考えています。

つまり、sequence-orthogonal array (順序直交配列) です。もし、そのような方法が確立すれば、3-way 表であっても 4-way 表で出現するデータが高率で含まれることやテスト結果の統計処理が期待できます。

以上