Embedded functions in combinatorial testing: Progress in automating test design

George B. Sherwood

Testcover.com

Combinatorial testing

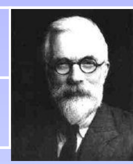
- A way to design tests to verify many inputs and configurations, with a small number of test cases
  - E.g. 30 factors: 6 with 3 values, 24 with 2 values
  - Possible test cases: $3^6 2^{24} = 12,230,590,464$
  - Pairwise test cases: 15 (2103 interactions)
  - Most failures are caused by only 1 or 2 factors
- Automated process based on some analysis
- Black-box perspective (requirements, interfaces)

Topics

- Origins of combinatorial testing and early tools
- Constraint needs and approaches
- Embedded functions concepts and uses
  - Combination functions
  - Substitution functions
- Higher strength needs and approaches
- An example. Designs to verify:
  - Equivalence classes (ECs) of expected results
  - Boundaries of expected results ECs
  - Interactions between EC boundaries

Testcover.com

Origins of combinatorial testing



1920		
	Design of Experiments	
1940	Orthogonal Arrays	
	OAs for Manufacturing	
1960		
	Covering Arrays	
1980	OAs for Software	
	OATS, CATS, AETG	Pairwise testing
2000	…	…
	Many more tools	Much more testing
2020		

Testcover.com

Orthogonal arrays

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

N size, number of rows or test cases

k number of columns or test factors

v number of symbols or values for each column

t strength

In OA(N; v^k) every N x t subarray contains every t-tuple from v symbols an equal number of times

In OA(N; $v_1^{k_1}$ $v_2^{k_2}$ … $v_s^{k_s}$) every N x t subarray contains every t-tuple an equal number of times

Testcover.com

Orthogonal arrays for software testing

- Keizo Tatsumi, Robert Mandl & others applied these ideas to software testing
- Taguchi visited Bell Labs and collaborated with Madhav Phadke
- Orthogonal Array Test System (OATS) offered statistical coverage where exhaustive testing was not possible

StarLAN and StarGROUP

- Client installation test
  - 101 combinations of PCs and OS versions
  - 15 network cards
- Tried to use OATS to reduce number of test configurations
- OATS could not handle our complex configuration constraints correctly
- Generated impossible test configurations

Constraint violation example

The example has 3 factors:

- 2 OS values
- 3 Browser values
- 3 Application values

These test cases cover all pairs of factor values:

- 6 OS-Browser pairs
- 6 OS-Application pairs
- 9 Browser-Application pairs

But the Linux-IE pair is disallowed

Constraint violation yields impossible test case			
Test Case	***OS***	***Browser***	***Application***
1	Windows	Chrome	App1
2	Windows	Firefox	App2
3	Windows	IE	App3
4	Linux	Chrome	App2
5	Linux	Firefox	App3
6	Linux	IE	App1
7	Windows	Chrome	App3
8	Windows	Firefox	App1
9	Windows	IE	App2

What to do?

Skip test case 6? Then Linux-App1 pair and IE-App1 pair are not covered

Change any 1 value in test case 6? Then at least one of the pairs still is not covered

Solution: Restrict search to allowed combinations only

Testcover.com

Constrained array test system (CATS)

- Search among allowed test cases to handle constraints
- Find next “best” test case to minimize uncovered combinations
- Higher strength: $t \leq k$
- Secondary result: fewer test cases than OATS
- CATS was searching for covering arrays, not orthogonal arrays

Orthogonal & covering arrays

N size, number of rows or test cases
k number of columns or test factors
v number of symbols or values for each column
t strength

In OA(N; $v_1^{k_1}\ v_2^{k_2} \ldots v_s^{k_s}$) every N x t subarray contains every t-tuple an equal number of times

In CA(N; $v_1^{k_1}\ v_2^{k_2} \ldots v_s^{k_s}$) every N x t subarray contains every t-tuple at least once

Testcover.com

Covering array templates

Covering arrays can be smaller than orthogonal arrays		
	Covering arrays	***Orthogonal Arrays***
Fixed values	$CA(12; 3^7)$	$OA(18; 3^7)$
Mixed values	$CA(6; 3^1 2^4)$	$OA(12; 3^1 2^4)$

Many authors have found covering arrays using diverse methods
Charlie Colbourn, Arizona State University:
Covering array tables for t = 2, 3, 4, 5, 6
www.public.asu.edu/~ccolbou/src/tabby/catable.html

Constraint needs

- Invalid configurations can mask valid combinations, e.g. Linux with Internet Explorer
- Combinations of valid inputs can be invalid, as in the date Feb 30, 2017
- Combinations of factor values determine ECs of expected results

Constraint approaches

- Logic based: (OS = “Windows”) =>
  (Browser = “IE” || Browser = “Firefox” || Browser = “Chrome”)
- Embedded functions: fBrowser($OS) represents Browser value according to $OS value

Testcover.com

Functionally dependent test factor values

- Constraints can be described using functionally dependent test factor values
- Functional dependence:

  1 or more values of a dependent factor are identified by other, determinant factors

  Determinant factors' values → dependent factor values
- Example: The last day of any month is identified by its month and year

  Month, Year → Last day values

  ℓ = number of determinant factors (ℓ = 2 in this example)
- Use Direct Product Block (DPB) notation with or without embedded combination functions

Direct product block (DPB) notation

Fixed values form

```
Calendar Example without last_day function
Month
Day
Year
#ok All good dates
jan feb mar apr may jun jul aug sep oct nov dec
1 10
2015 2016 2017
+ long month last day
jan mar may jul aug oct dec
31
2015 2016 2017
+ short month last day
apr jun sep nov
30
2015 2016 2017
+ feb last day
feb
28
2015 2017
+ leap day
feb
29
2016
```

- Valid calendar dates example with boundary checking
- Factor values are on separate lines
- All combinations in a block are allowed
- Partition of multiple blocks includes union of their allowed combinations

Direct product block (DPB) notation

Fixed values form

```
Calendar Example without last_day function
Month
Day
Year
#ok All good dates
jan feb mar apr may jun jul aug sep oct nov dec
1 10
2015 2016 2017
+ long month last day
jan mar may jul aug oct dec
31
2015 2016 2017
+ short month last day
apr jun sep nov
30
2015 2016 2017
+ feb last day
feb
28
2015 2017
+ leap day
feb
29
2016
```

Functionally dependent form

```
Calendar Example with last_day function
$month
Day
$year
#ok All good dates
jan feb mar apr may jun jul aug sep oct nov dec
1 10 last_day($month,$year)
2015 2016 2017
```

- Month, Year → Last day value
- Factors renamed as variables for function arguments:
  $month $year
- Day values:
  1 10 last_day($month,$year)
- 5 blocks now represented by only 1 block

Testcover.com

Combination functions

- last_day($month,$year) is a combination function
- Combination functions return dependent values for all allowed combinations of determinant factor values
- Generator uses these fixed values to construct test cases
- last_day($month,$year) needs to return the last day for any month in the years 2015 2016 2017
- PHP built-in function cal_days_in_month is reused:

```
function last_day($month,$year) {
        $mo_num=array('jan'=>1,'feb'=>2,'mar'=>3,'apr'=>4,'may'=>5,'jun'=>6,
                'jul'=>7,'aug'=>8,'sep'=>9,'oct'=>10,'nov'=>11,'dec'=>12);
        return(cal_days_in_month(CAL_GREGORIAN,$mo_num[$month],(int)$year));
}
```

Testcover.com

Substitution functions

- A substitution function returns a value for each test case after test case generation
- A substitution function value can be determined by other factor values its test case
- Substitution functions can identify equivalence classes for the expected results of each test case
  - To evaluate expected results classes automatically
  - To assess coverage of expected results classes
- Equivalence class functions help manage verification of results classes

Interaction rule

- Studies show a high proportion of failures are caused by faults in only 1 or 2 factors
- Higher strength designs (t > 2) can find more failures
  - at a diminishing rate
  - with an increase in test cases (N)

Higher strength needs

- Nature of failure, more determinant factors
- ECs of expected results depend on more inputs
- Difficulty conforming to constraints for valid test cases
- Inadequate specifications, e.g. interfaces, behavior

Higher strength approaches

- Increase strength until no more failures are found
  - thorough & expensive: $N \sim v^t \log k$
- Apply higher strength to a subset of factors
  - less expensive (smaller N)
  - test factor risk estimate
- Use embedded functions to constrain factor values
  - even less expensive (t = 2)
  - EC functions to verify classes of expected results
  - simplifies constraints for valid test cases

Testcover.com

BMI report requirements

R1. The listed input data will be stored in the patient database table.

a.	Age in years	(integer) $2 \le Age < 130$
b.	Weight in pounds	(integer) $20 \le Weight < 500$
c.	Height in inches	(integer) $30 \le Height < 90$
d.	Sex	(female, male)
e.	Intake in kilocalories per day	(integer) $1 \le Intake < 10000$

R2. The BMI will be calculated (in kilograms per meter squared) as $703.06957964\ Weight / Height^2$ and stored in the patient database table.

R3. If Age is 65 years or older, the Medicare report will be generated.

R4. If Age is younger than 20 years, the Child report containing the BMI percentile will be generated for the corresponding listed classification.

a. Girl, from the female BMI-age table

b. Boy, from the male BMI-age table

R5. If Age is 20 years or older, the Adult report will be generated for the corresponding listed classification.

a.	Underweight	$BMI < 18.5$
b.	Normal	$18.5 \le BMI < 25.0$
c.	Overweight	$25.0 \le BMI < 30.0$
d.	Obese	$30.0 \le BMI$

BMI report equivalence classes

- Equivalence classes group test factor combinations by similar expected results
- Classes help insure test design coverage

  Example: The Medicare, Child and Adult reports each have multiple, valid equivalence classes

Report	Valid equivalence classes				
Medicare	no	yes			
Child	no	girl	boy		
Adult	no	underweight	normal	overweight	obese

- Equivalence classes are functionally dependent

  Input, configuration values → result → equivalence class
- Report classes can be expressed as 3 functions

Testcover.com

Equivalence class functions 1

```
function Medicare_report($Age) {           /* Medicare_report equivalence class function */
   if($Age>=65) return('yes');             /* Medicare_report is expected result */
   if($Age>0) return('no');                /* Medicare_report is not expected result */
}
```

```
function Child_report($Age,$Sex) {         /* Child_report equivalence class function */
   if($Age>0) {
      switch($Sex) {
         case 'female':
            if($Age<20) return('girl');    /* Child_report for girl is expected result */
            else return('no');            /* Child_report is not expected result */
         case 'male':
            if($Age<20) return('boy');     /* Child_report for boy is expected result */
            else return('no');             /* Child_report is not expected result */
      }
   }
}
```

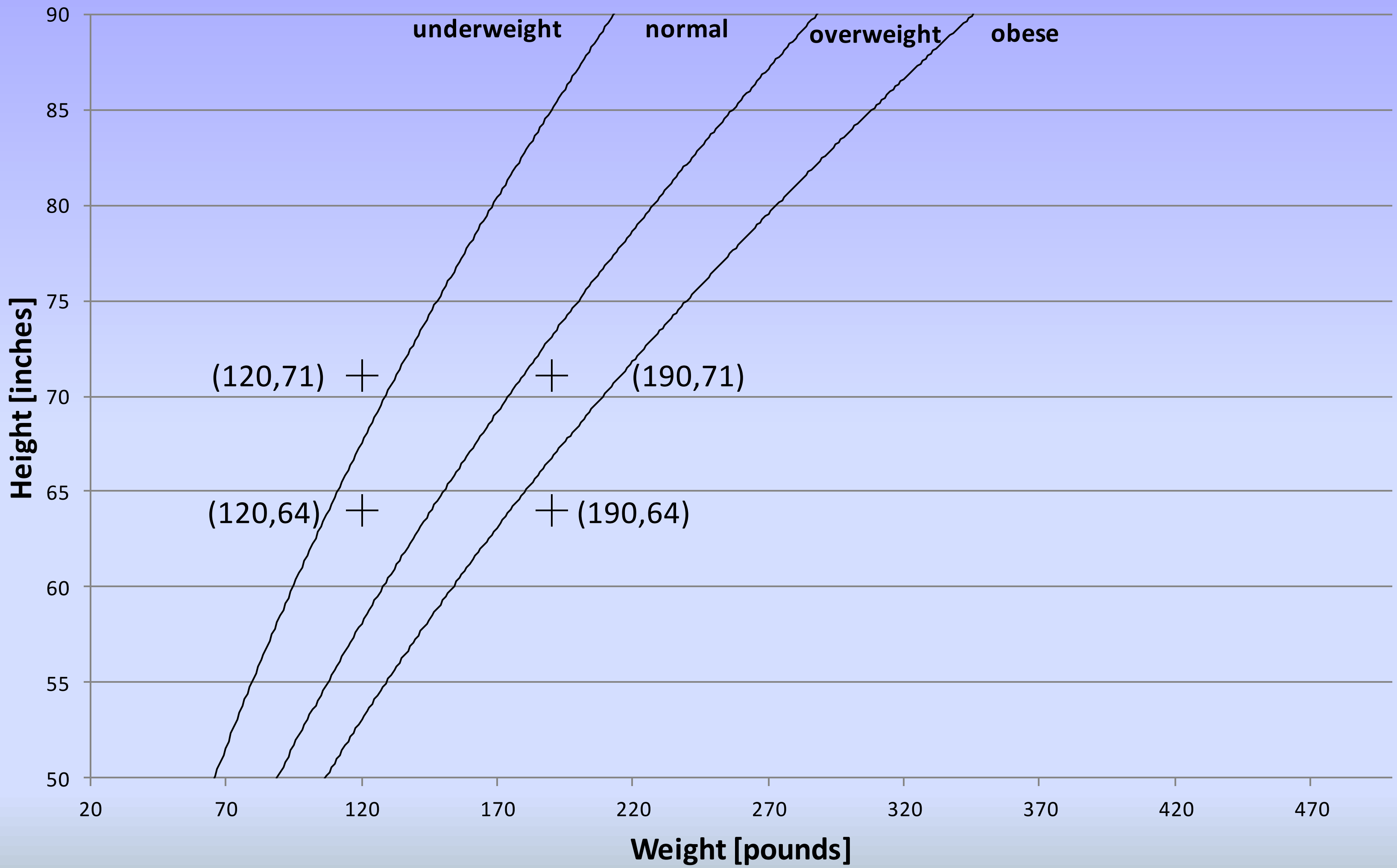
Testcover.com

Equivalence class functions 2

```
function Adult_report($Age,$Weight,$Height) {    /* Adult_report equivalence class function */
   $bmi_value=BMI($Weight,$Height);              /* Use BMI function to get BMI value */
   if($Age>=20&&$bmi_value>0) {
      if($bmi_value>=30) return('obese');        /* Adult_report for obese is expected result */
      if($bmi_value>=25) return('overweight');   /* Adult_report for overweight is expected result */
      if($bmi_value>=18.5) return('normal');     /* Adult_report for normal is expected result */
      if($bmi_value>0) return('underweight');    /* Adult_report for underweight is expected result */
   }
   if($Age>0&&$bmi_value>0) return('no');        /* Adult_report is not expected result */
}
```

```
function BMI($Weight,$Height) {                  /* BMI function for Adult_report */
   if($Weight>0&&$Height>0) {
      $bmi_value=703.06957964*$Weight
         /$Height/$Height;
      return($bmi_value);                        /* return valid BMI value */
   }
}
```

Weight-height values for BMI classes



Testcover.com

Equivalence class substitution functions

Accidental equivalence class coverage	**Incomplete equivalence class coverage**
$Age	$Age
$Weight	$Weight
$Height	$Height
$Sex	$Sex
Intake	Intake
Medicare equivalence class	Medicare equivalence class
Child equivalence class	Child equivalence class
Adult equivalence class	Adult equivalence class
#	#
15 42 67	**42 67 15**
120 190	120 190
64 71	64 71
female male	female male
2000 3000	2000 3000
Medicare_report($Age)	Medicare_report($Age)
Child_report($Age,$Sex)	Child_report($Age,$Sex)
Adult_report($Age,$Weight,$Height)	Adult_report($Age,$Weight,$Height)

- Equivalence class functions return expected ECs from determinant factors in generated test cases
- Same pairwise requests, but with Age values rotated

Testcover.com

Equivalence class substitution functions

Accidental equivalence class coverage								
	Input factors					Equivalence class factors		
Test Case	***Age***	***Weight***	***Height***	***Sex***	***Intake***	***Medicare***^{s}	***Child***^{s}	***Adult***^{s}
1	67	120	71	male	3000	yes	no	underweight
2	67	190	64	female	2000	yes	no	obese
3	42	190	71	male	2000	no	no	overweight
4	15	120	64	male	2000	no	boy	no
5	42	120	64	female	3000	no	no	normal
6	15	190	71	female	3000	no	girl	no
				Equivalence classes covered:		**2 of 2**	**3 of 3**	**5 of 5**

Incomplete equivalence class coverage								
	Input factors					Equivalence class factors		
Test Case	***Age***	***Weight***	***Height***	***Sex***	***Intake***	***Medicare***^{s}	***Child***^{s}	***Adult***^{s}
1	15	120	71	male	3000	no	boy	no
2	15	190	64	female	2000	no	girl	no
3	67	190	71	male	2000	yes	no	overweight
4	42	120	64	male	2000	no	no	normal
5	67	120	64	female	3000	yes	no	normal
6	42	190	71	female	3000	no	no	overweight
				Equivalence classes covered:		**2 of 2**	**3 of 3**	**3 of 5**

- Same input factors in test cases, but with Age values rotated
- Different expected ECs: underweight & obese classes missing
- Strength $t = 2 < 3 = \ell$, number of Adult determinant factors

Equivalence class combination functions

Equivalence class coverage using EC factors
$Age
$Weight
$Height
$Sex
Intake
Medicare equivalence class
Child equivalence class
Adult equivalence class
#
15 42 67
120 190
64 71
female male
2000 3000
Medicare_report($Age)
Child_report($Age,$Sex)
Adult_report($Age,$Weight,$Height)

- Same pairwise request, but with EC combination functions
- EC functions return results classes from determinant factors for test cases
- 1 functionally dependent block → 10 fixed values blocks
- Every allowed EC value appears in at least one test case with a combination of its determinant factor values

Testcover.com

Equivalence class combination functions

Equivalence class coverage using equivalence class factors								
	Input factors					Equivalence class factors		
Test Case	***Age***	***Weight***	***Height***	***Sex***	***Intake***	***Medicare***[c]	***Child***[c]	***Adult***[c]
1	42	190	71	female	2000	no	no	overweight
2	67	120	64	male	3000	yes	no	normal
3	15	120	64	male	2000	no	boy	no
4	15	190	64	female	3000	no	girl	no
5	67	120	71	female	2000	yes	no	underweight
6	42	190	64	male	3000	no	no	obese
7	67	190	71	male	3000	yes	no	overweight
8	42	120	71	male	3000	no	no	underweight
9	42	120	64	female	2000	no	no	normal
10	15	190	71	male	2000	no	boy	no
11	67	190	64	female	2000	yes	no	obese
12	15	120	64	female	2000	no	girl	no
13	15	190	64	male	3000	no	boy	no
14	15	190	71	female	2000	no	girl	no
				Equivalence classes covered:		**2 of 2**	**3 of 3**	**5 of 5**

- All 10 expected ECs covered by constraints in pairwise design
- Equivalence classes are paired with nondeterminant factor values

Testcover.com

Age boundary values

Age boundary and edge values				
Boundaries and edges		**Equivalence class factors**		
Age	***Limit***	***Medicare***	***Child***	***Adult***
129	max	yes		
65	min			
64	max	no		
20	min			
19	max		girl	
2	min			

Coverage of univariate Age boundaries using EC factors
$Age
$Weight
$Height
$Sex
Intake
Medicare equivalence class
Child equivalence class
Adult equivalence class
#
19 20 64 65
120 190
64 71
female male
2000 3000
Medicare_report($Age)
Child_report($Age,$Sex)
Adult_report($Age,$Weight,$Height)

- Boundaries separate ECs
- Edges are extreme values
- Same pairwise request, now with Age boundary values
- EC combination functions cover expected results as before

Testcover.com

Age boundaries with EC factors

Coverage of univariate equivalence class boundaries using equivalence class factors								
	Input factors					Equivalence class factors		
Test Case	***Age***	***Weight***	***Height***	***Sex***	***Intake***	***Medicare***[c]	***Child***[c]	***Adult***[c]
2	65	120	64	male	3000	yes	no	normal
6	65	120	71	female	2000	yes	no	underweight
8	65	190	71	male	3000	yes	no	overweight
12	65	190	64	female	2000	yes	no	obese
4	64	190	64	female	3000	no	no	obese
7	64	120	71	male	2000	no	no	underweight
15	64	190	71	male	2000	no	no	overweight
16	64	120	64	male	2000	no	no	normal
1	20	190	71	female	2000	no	no	overweight
9	20	120	64	female	2000	no	no	normal
10	20	190	64	male	3000	no	no	obese
13	20	120	71	male	3000	no	no	underweight
3	19	120	64	male	2000	no	boy	no
5	19	120	71	female	3000	no	girl	no
11	19	190	71	male	2000	no	boy	no
14	19	120	64	female	2000	no	girl	no
17	19	190	64	male	3000	no	boy	no
18	19	190	71	female	2000	no	girl	no
Equivalence classes covered:						**2 of 2**	**3 of 3**	**5 of 5**

- Test cases are sorted to show age boundaries
- All 10 expected ECs covered by constraints in pairwise design

Testcover.com

Weight function for BMI boundary value

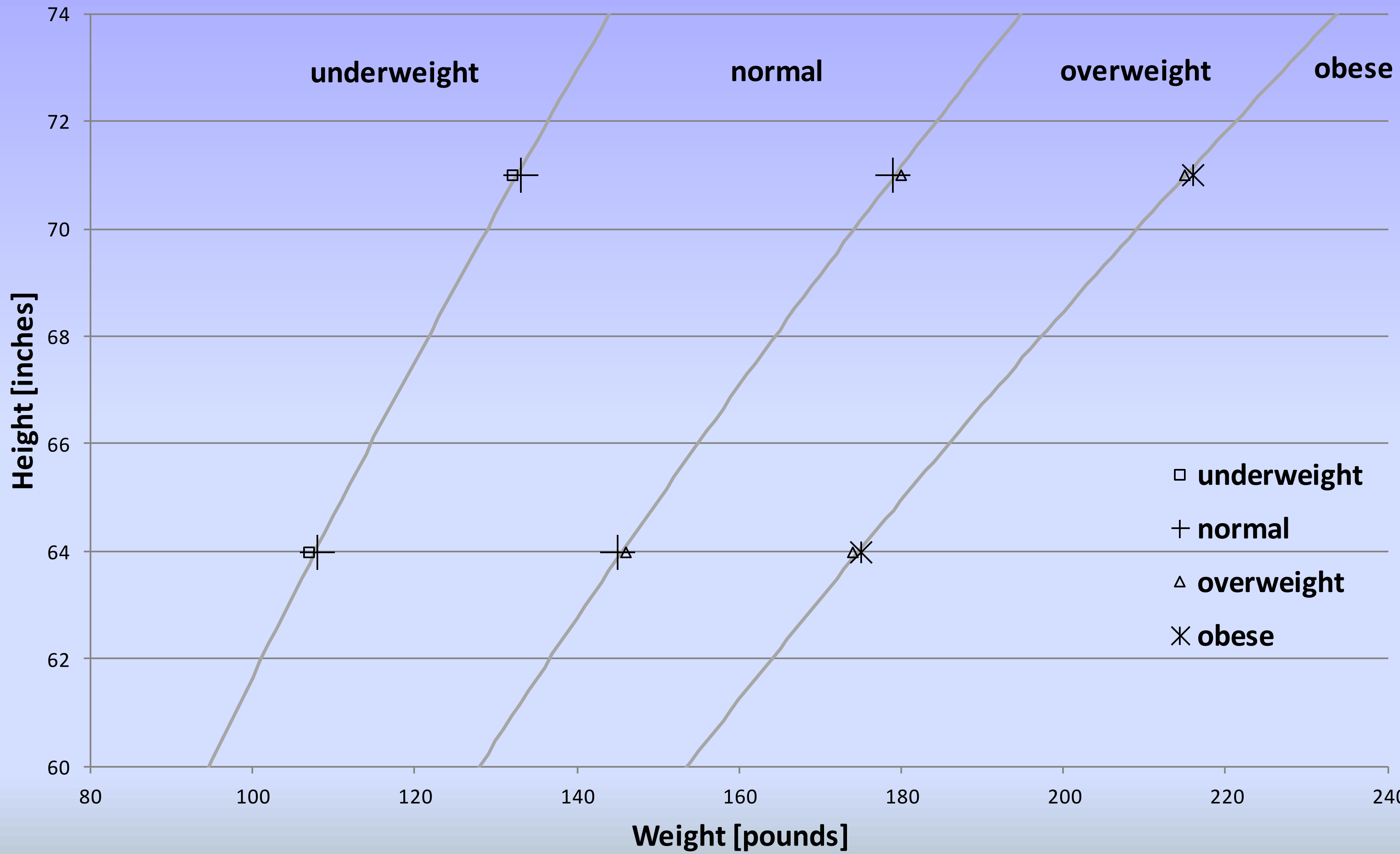
BMI = 703.06957964 Weight / $Height^2$
Weight = $Height^2$ · BMI / 703.06957964

Height values: 64 71
BMI_boundary values: 18.5 25 30
Input_limit values: min max

```
function Weight_boundary($Height,$BMI_boundary,$Input_limit) {     /* compute weight input */
   if($Height>0&&$BMI_boundary>0) {
      $w_hi=ceil($Height*$Height*$BMI_boundary/703.06957964);      /* round up $w_hi */
      switch($Input_limit) {                                      /* so BMI >= $BMI_boundary */
         case 'min':
            return($w_hi);                                        /* return min integer value */
         case 'max':                                              /* for higher BMI class */
            $w_lo=$w_hi-1;
            return $w_lo;                                         /* return max integer value */
      }                                                           /* for lower BMI class */
   }
}
```

Testcover.com

BMI boundary values



Testcover.com

Age & BMI boundaries – all corners

Coverage of Age & BMI boundaries using BV factors with dependent Weight values – all corners
$Age
$Weight
$Height
$Sex
Intake
Medicare equivalence class
Child equivalence class
Adult equivalence class
$BMI_boundary
$Input_limit
#
19 20 64 65
Weight_boundary($Height,$BMI_boundary,$Input_limit)
64 71
female male
2000 3000
Medicare_report($Age)
Child_report($Age,$Sex)
Adult_report($Age,$Weight,$Height)
18.5 25 30
min max

Testcover.com

Age & BMI – all corners – 1 of 2

Coverage of Age & BMI boundaries using boundary value factors with dependent Weight values – all corners										
	Input factors					**Equivalence class factors**			**BMI BV factors**	
Test Case	***Age***	***Weight***[c]	***Height***	***Sex***	***Intake***	***Medicare***[s]	***Child***[s]	***Adult***[s]	***BMI***	***Limit***
1	19	180	71	female	2000	no	girl	no	25	min
2	65	174	64	male	3000	yes	no	overweight	30	max
3	20	132	71	male	2000	no	no	underweight	18.5	max
4	64	108	64	female	3000	no	no	normal	18.5	min
5	20	216	71	female	3000	no	no	obese	30	min
6	19	145	64	male	3000	no	boy	no	25	max
7	64	215	71	male	2000	no	no	overweight	30	max
8	65	179	71	female	2000	yes	no	normal	25	max
9	20	146	64	male	2000	no	no	overweight	25	min
10	65	133	71	female	2000	yes	no	normal	18.5	min
11	19	107	64	female	2000	no	girl	no	18.5	max
12	19	175	64	female	2000	no	girl	no	30	min
13	64	146	64	female	3000	no	no	overweight	25	min
14	65	180	71	male	3000	yes	no	overweight	25	min
15	64	132	71	female	3000	no	no	underweight	18.5	max
16	19	179	71	male	3000	no	boy	no	25	max
17	64	216	71	male	2000	no	no	obese	30	min
18	20	215	71	female	3000	no	no	overweight	30	max
19	19	133	71	male	3000	no	boy	no	18.5	min
20	19	174	64	female	2000	no	girl	no	30	max
21	65	107	64	male	3000	yes	no	underweight	18.5	max
22	65	145	64	female	2000	yes	no	normal	25	max
23	65	175	64	male	3000	yes	no	obese	30	min
24	20	108	64	male	2000	no	no	normal	18.5	min

Age & BMI – all corners – 2 of 2

Coverage of Age & BMI boundaries using boundary value factors with dependent Weight values – all corners										
	Input factors					Equivalence class factors			BMI BV factors	
Test Case	***Age***	***Weight*** [c]	***Height***	***Sex***	***Intake***	***Medicare*** [s]	***Child*** [s]	***Adult*** [s]	***BMI***	***Limit***
25	20	180	71	male	2000	no	no	overweight	25	min
26	64	180	71	female	3000	no	no	overweight	25	min
27	19	132	71	female	2000	no	girl	no	18.5	max
28	65	132	71	female	2000	yes	no	underweight	18.5	max
29	20	179	71	male	2000	no	no	normal	25	max
30	64	179	71	female	3000	no	no	normal	25	max
31	19	216	71	female	2000	no	girl	no	30	min
32	65	216	71	female	2000	yes	no	obese	30	min
33	19	215	71	female	2000	no	girl	no	30	max
34	65	215	71	female	2000	yes	no	overweight	30	max
35	20	133	71	male	2000	no	no	normal	18.5	min
36	64	133	71	female	3000	no	no	normal	18.5	min
37	20	174	64	male	2000	no	no	overweight	30	max
38	64	174	64	female	3000	no	no	overweight	30	max
39	20	107	64	male	2000	no	no	underweight	18.5	max
40	64	107	64	female	3000	no	no	underweight	18.5	max
41	19	146	64	female	2000	no	girl	no	25	min
42	65	146	64	female	2000	yes	no	overweight	25	min
43	20	145	64	male	2000	no	no	normal	25	max
44	64	145	64	female	3000	no	no	normal	25	max
45	20	175	64	male	2000	no	no	obese	30	min
46	64	175	64	female	3000	no	no	obese	30	min
47	19	107	64	female	2000	no	girl	no	18.5	min
48	65	108	64	female	2000	yes	no	normal	18.5	min

Testcover.com

Age-BMI 48-corner map

Test cases covering all Age-BMI corners

		BMI	18.5		25		30	
Height	Age		max	min	max	min	max	min
71	65	min	28	10	8	14	34	32
	64	max	15	36	30	26	7	17
	20	min	3	35	29	25	18	5
	19	max	27	19	16	1	33	31
64	65	min	21	48	22	42	2	23
	64	max	40	4	44	13	38	46
	20	min	39	24	43	9	37	45
	19	max	11	47	6	41	20	12

Age & BMI boundaries – some corners

Coverage of Age & BMI boundaries using BV factors with dependent Weight values – **some corners**
$Age
$Weight
$Height
$Sex
Intake
Medicare equivalence class
Child equivalence class
Adult equivalence class
$BMI_boundary
$Input_limit
#
19 20 64 65
Weight_boundary($Height,$BMI_boundary,$Input_limit)
64 71
female male
2000 3000
Medicare_report($Age)
Child_report($Age,$Sex)
Adult_report($Age,$Weight,$Height)
18.5 25 30
min max

Testcover.com

Age & BMI – some corners

Coverage of Age & BMI boundaries using boundary value factors with dependent Weight values – some corners										
	Input factors					Equivalence class factors			BMI BV factors	
Test Case	*Age*	*Weight*[s]	*Height*	*Sex*	*Intake*	*Medicare*[s]	*Child*[s]	*Adult*[s]	*BMI*	*Limit*
1	65	179	71	male	2000	yes	no	normal	25	max
2	19	216	71	female	3000	no	girl	no	30	min
3	20	107	64	male	2000	no	no	underweight	18.5	max
4	64	145	64	female	3000	no	no	normal	25	max
5	64	216	71	male	2000	no	no	obese	30	min
6	65	108	64	female	3000	yes	no	normal	18.5	min
7	20	216	71	female	3000	no	no	obese	30	min
8	19	108	64	female	2000	no	girl	no	18.5	min
9	20	146	64	male	3000	no	no	overweight	25	min
10	19	179	71	male	2000	no	boy	no	25	max
11	65	174	64	male	2000	yes	no	overweight	30	max
12	64	132	71	male	3000	no	no	underweight	18.5	max

Testcover.com

Age-BMI 12-corner map

Test cases covering some Age-BMI corners								
		BMI	18.5		25		30	
Height	Age		max	min	max	min	max	min
71	65	min			1			
	64	max	12					5
	20	min						7
	19	max			10			2
64	65	min		6			11	
	64	max			4			
	20	min	3			9		
	19	max		8				

Testcover.com

Conclusions

- Decades of progress: orthogonal arrays, covering arrays, constraints, greedy searches, higher strength
- Embedded functions:
  - constraints: simple functions in a familiar language
  - control: higher strength for determinant factors
  - efficiency: fewer test cases
  - flexibility: variety of test objectives
  - automation: less manual analysis
- 21st century testing: increasing dependency on software, networks and distributed applications

Testcover.com