Software Product Line Testing Part IV : A Framework for Variability Coverage

Myra Cohen Matthew Dwyer

Laboratory for Empirically-based Software Quality Research
Department of Computer Science
University of Nebraska - Lincoln

Work supported by NSF CCF through awards 0429149 and 0444167, by the U.S. Army Research Office through award DAAD190110564 and by an NSF EPSCoR FIRST award.

e2 ESQuaReD

University of Nebraska Lincoln

Outline

- Software Product Lines : What and Why?
- Modeling Variability in Software Product Lines
- Validating Product Lines
- A Framework for Variability Coverage
- Toward Product Line Driven Test Processes

Outline

A Framework for Variability Coverage

- 1. Adapting Covering Arrays to Real Systems
2. Building Covering Arrays
    1. Overview
    2. One-row-at-a-time Greedy Algorithms
    3. Meta-heuristic Search

CA(9;2,4,3)

(also an OA(2,4,3))

A set of product line instances that covers all pair-wise interactions.

Config	Browser	OS	Connection	Printer
1	Netscape	Windows XP	LAN	Local
2	Netscape	Linux	ISDN	Networked
3	Netscape	OS X	PPP	Screen
4	IE	Windows XP	ISDN	Screen
5	IE	OS X	LAN	Networked
6	IE	Linux	PPP	Local
7	Mozilla	Windows	PPP	Networked
8	Mozilla	Linux	LAN	Screen
9	Mozilla	OS X	ISDN	Local

Interaction Strength

- We can quantify the "coverage" for a particular interaction strength.
- Example:
  - 4 factors
  - Each has 3 values
  - Quantify 2-way coverage

  Any single test case can cover at most $\binom{4}{2}$ or 6 possible pairs.

  The system has $\binom{4}{2}3^2$ or 54 pair-wise interactions.

  The addition of one new test can contribute at most 6/54 or 11.1% pair wise coverage.

Adaptations for Real Systems

- So far we have seen a covering array with *v* symbols. This means each factor has the same number of values.

BUT:

- This is usually not the case in a real system.

•Model : Civic, Accord
•Package : Sedan, Coupe, Hybrid, GX, Si
•Transmission : manual, auto, cvt
•Power : gas, hybrid, natural gas
•Doors : 2, 4
•Cylinders : 4, 6
•Nav system : Y/N

Mixed Level Covering Arrays

$MCA_{\lambda}(N;t,k,(v_1,v_2,\ldots,v_k))$

Is an *N* x *k* array on *v* symbols where:

$$v = \sum_{i=1}^{k} v_i$$

And:

- For each column *i* where (i ≤ *i* ≤ *k*)
- The rows of each *N x t* sub-array cover all *t*-tuples or values from the t columns at least λ times.

Shorthand Notation:

$MCA_{\lambda}(N;t,(w_1^{k_1} w_2^{k_2} \ldots w_s^{k_s}))$

Example: MCA(12;2,4,(4, 3,3,2)) ≡ $MCA(12;2,(4^1\ 3^2 2^1))$

MCA(12;2,$4^1 3^2 2^1$)

Netscape	Windows XP	LAN	Local
Mozilla	OS X	ISDN	Networked
Safari	Linux	PPP	Networked
Mozilla	Linux	LAN	Local
Netscape	OS X	PPP	Local
IE	Windows XP	ISDN	Networked
IE	OS x	LAN	Networked
Safari	Windows XP	ISDN	Networked
Netscape	Linux	ISDN	Local
Mozilla	Windows XP	PPP	Local
Safari	OS X	LAN	Local
IE	Linux	PPP	Networked

MCA(12;2,$4^1 3^2 2^1$)

Netscape	Windows XP	LAN	Local
Mozilla	OS X	ISDN	Networked
Safari	Linux	PPP	Networked
Mozilla	Linux	LAN	Local
Netscape	OS X	PPP	Local
IE	Windows XP	ISDN	Networked
IE	OS x	LAN	Networked
Safari	Windows XP	ISDN	Networked
Netscape	Linux	ISDN	Local
Mozilla	Windows XP	PPP	Local
Safari	OS X	LAN	Local
IE	Linux	PPP	Networked

Limitation

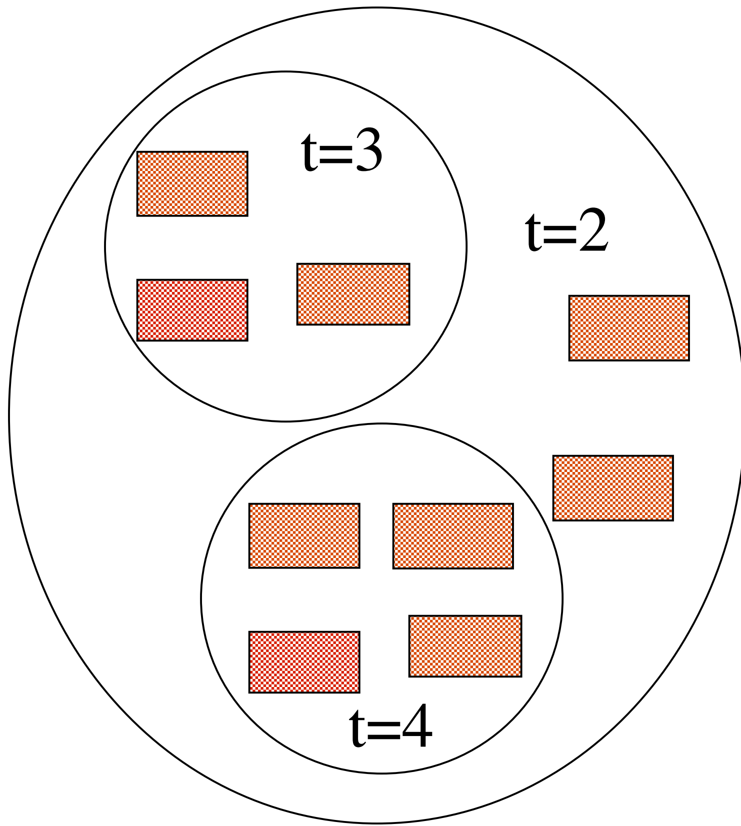
- Mixed level covering arrays are closer to real systems; they can be used for an arbitrary software system.

- But they view a system flatly. They force a (perhaps arbitrary) restriction on the importance of various parts of the system.

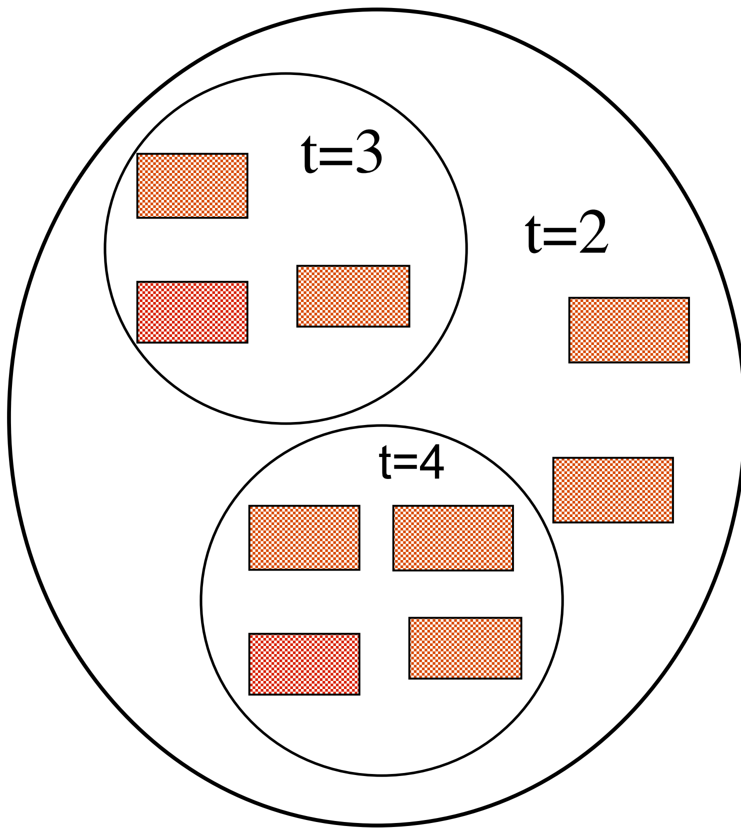
Scenarios

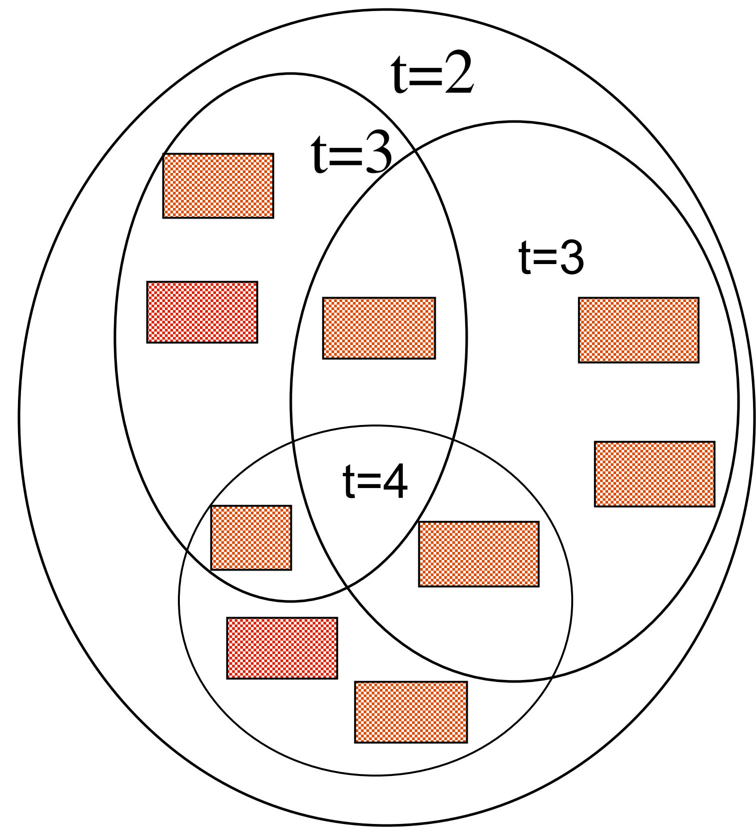
1. When testing a software system certain components may be closely interrelated. This may be determined by static analysis.
2. Operational profiles give us information that certain areas of the system are used more often than others.
3. In modifying a system only certain regions are changed therefore we want to test more strongly in this area.
4. Failures in certain parts of a system are more costlier than in others.

Possible Models

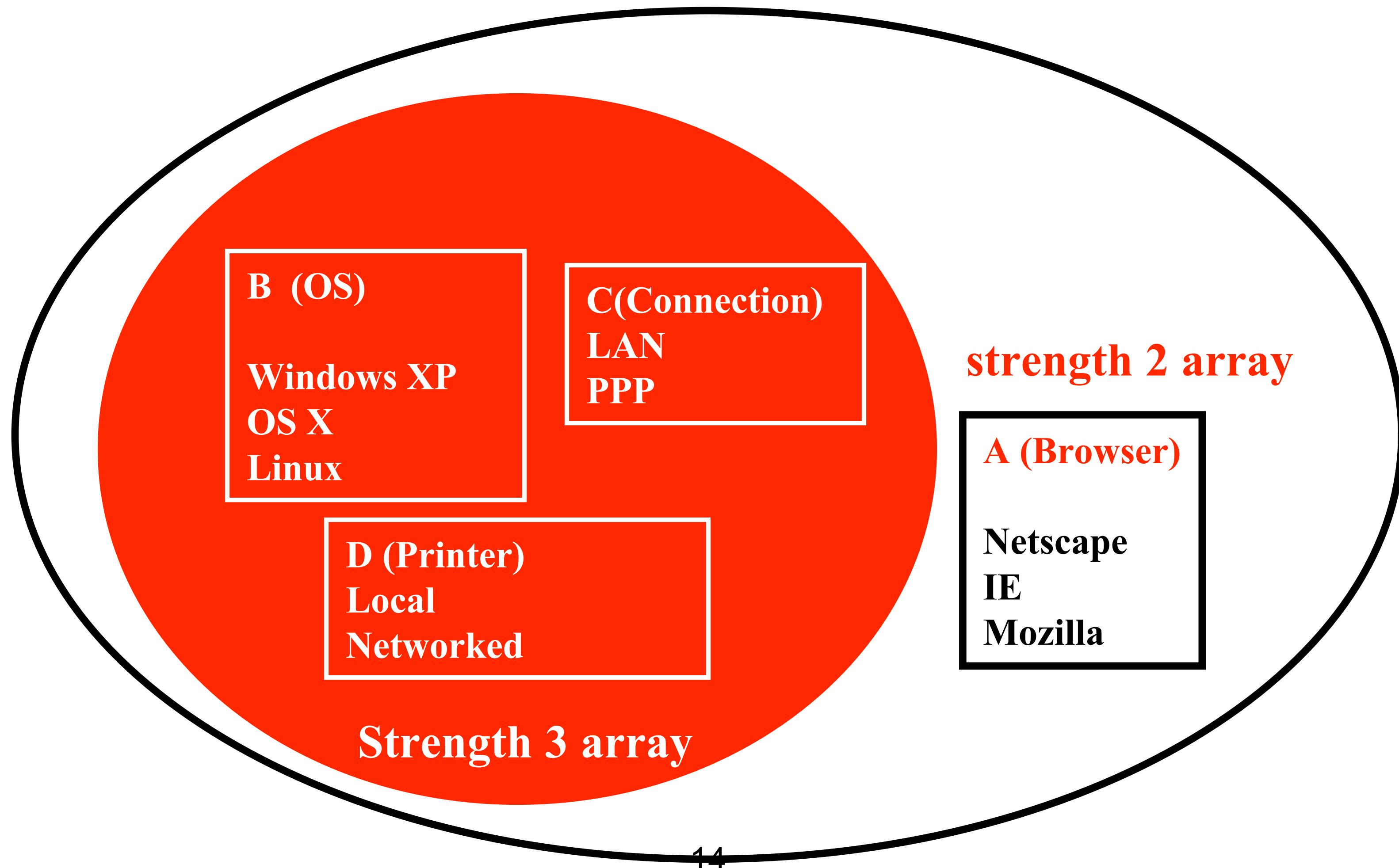


Some Possible Models

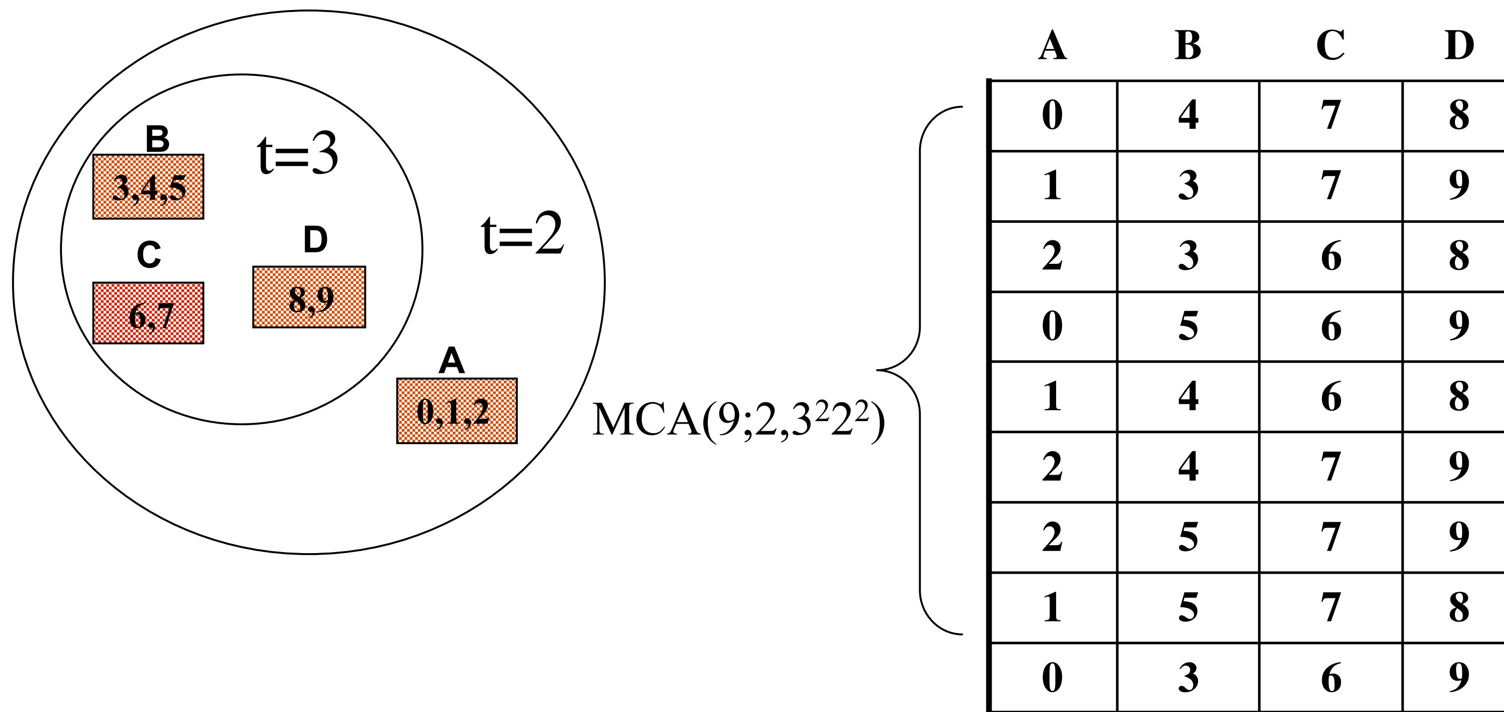




An Initial Model



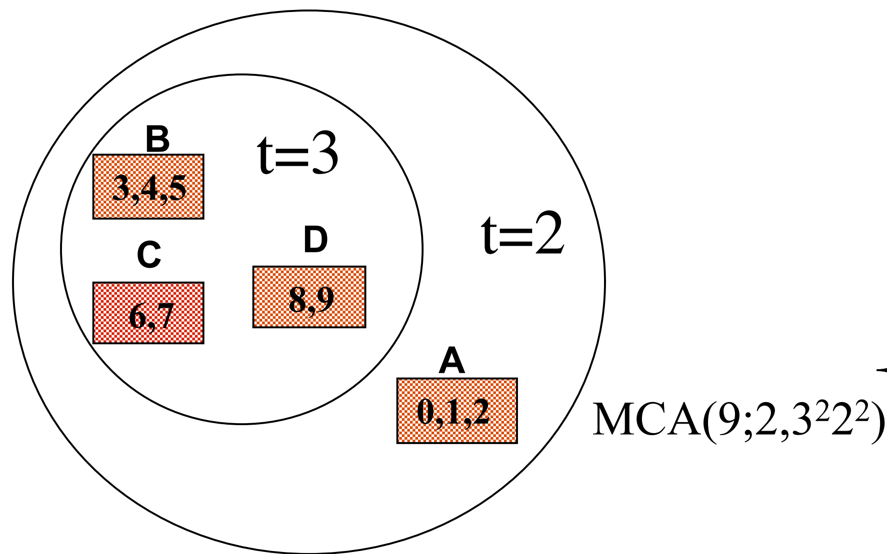
Variable Strength Covering Arrays



A	B	C	D
0	4	7	8
1	3	7	9
2	3	6	8
0	5	6	9
1	4	6	8
2	4	7	9
2	5	7	9
1	5	7	8
0	3	6	9

A 3-way array would have 18 rows

Variable Strength Covering Arrays



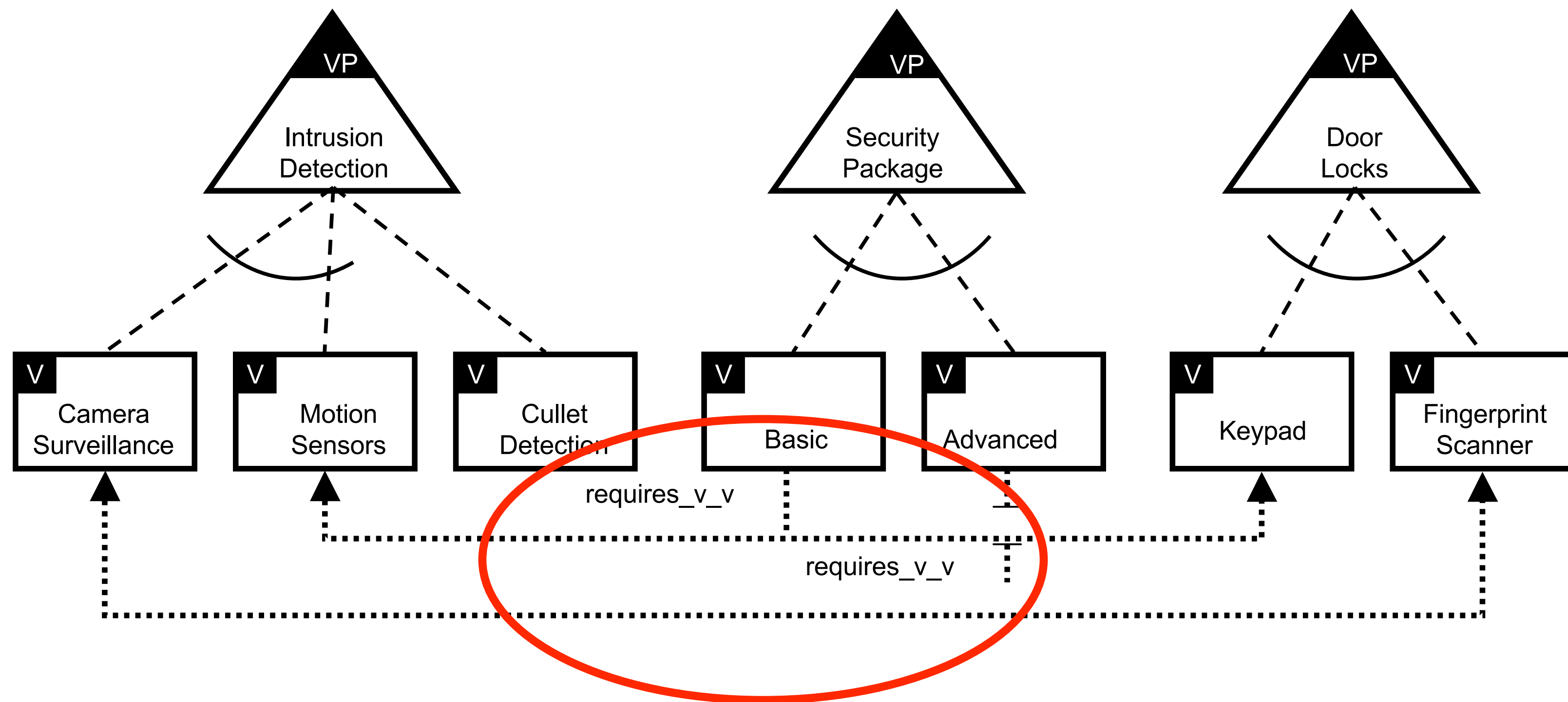
A	B	C	D
0	4	7	8
1	3	7	9
2	3	6	8
0	5	6	9
1	4	6	8
2	4	7	9
2	5	7	9
1	5	7	8
0	3	6	9
0*	3	7	8
2*	4	6	9
1*	5	6	8

Additional rows to guarantee subset of 3-way coverage

Variable Strength Covering Array

- A VCA($N;t,k,(v_1,v_2,\ldots v_k), C$) is a t-way mixed level covering array on v symbols with a vector, C, of covering arrays each with strength > t and defined on a subset of the k columns of the VCA.

Adding Constraints



Adding Constraints

Constrained Set of Product Instances

"Basic requires Motion Sensors and Keypad"

"Basic excludes Cullet Detection"

Intrusion Detection A	**Intrusion Detection B**	**Security Package**	**Door Locks**
Camera Surveillance	None	Advanced	Fingerprint Scanner
Motion Sensors	Cullet Detection	Advanced	Fingerprint Scanner
Camera Surveillance	Cullet Detection	Advanced	Keypad
Motion Sensors	None	Basic	Keypad
Camera Surveillance	Cullet Detection	Basic	Fingerprint Scanner

Adding Constraints

Constrained Set of Product Instances

"Basic requires Motion Sensors and Keypad"

"Basic excludes Cullet Detection"

Intrusion Detection A	**Intrusion Detection B**	**Security Package**	**Door Locks**
Camera Surveillance	None	Advanced	Fingerprint Scanner
Motion Sensors	Cullet Detection	Advanced	Fingerprint Scanner
Camera Surveillance	Cullet Detection	Advanced	Keypad
Motion Sensors	None	Basic	Keypad
~~Camera Surveillance~~	~~Cullet Detection~~	~~Basic~~	~~Fingerprint Scanner~~

Another set of Constraints

Constrained Set of Product Instances

"*Camera Surveillance requires Fingerprint Scanner*"

Intrusion Detection A	**Intrusion Detection B**	**Security Package**	**Door Locks**
Camera Surveillance	None	Advanced	Fingerprint Scanner
Motion Sensors	Cullet Detection	Advanced	Fingerprint Scanner
Camera Surveillance	Cullet Detection	Advanced	Keypad
Motion Sensors	None	Basic	Keypad
Camera Surveillance	Cullet Detection	Basic	Fingerprint Scanner

Another set of Constraints

Constrained Set of Product Instances

"*Camera Surveillance requires Fingerprint Scanner*"

Intrusion Detection A	**Intrusion Detection B**	**Security Package**	**Door Locks**
Camera Surveillance	None	Advanced	Fingerprint Scanner
Motion Sensors	Cullet Detection	Advanced	Fingerprint Scanner
~~Camera Surveillance~~	~~Cullet Detection~~	~~Advanced~~	~~Keypad~~
Motion Sensors	None	Basic	Keypad
Camera Surveillance	Cullet Detection	Basic	Fingerprint Scanner
Motion Sensors	Cullet Detection	Advanced	Keypad

Constraints in our Original Example

"*Mozilla does not run on Linux*"

"Linux does not support print to the screen"

Test Case	Browser	OS	Connection	Printer
1	Netscape	Windows XP	LAN	Local
2	Netscape	Linux	ISDN	Networked
3	Netscape	OS X	PPP	Screen
4	IE	Windows XP	ISDN	Screen
5	IE	OS X	LAN	Networked
6	IE	Linux	PPP	Local
7	Mozilla	Windows XP	PPP	Networked
8	Mozilla	Linux	LAN	Screen
9	Mozilla	OS X	ISDN	Local

Constraints in our Original Example

"Mozilla does not run on Linux"

"Linux does not support print to the screen"

Test Case	Browser	OS	Connection	Printer
1	Netscape	Windows XP	LAN	Local
2	Netscape	Linux	ISDN	Networked
3	Netscape	OS X	PPP	Screen
4	IE	Windows XP	ISDN	Screen
5	IE	OS X	LAN	Networked
6	IE	Linux	PPP	Local
7	Mozilla	Windows XP	PPP	Networked
8	Mozilla	Linux	LAN	Screen
8	Mozilla	OS X	ISDN	Local
9	IE	Linux	LAN	Local
10	Netscape	Windows XP	LAN	Screen

Other Practical Issues

- Seeded or default instances.
- Aggregate factors.
- Cost of testing specific instances.
  - May consider order of validating instances and set-up costs.

Outline : Interaction Testing in Practice

A Framework for Variability Coverage

1. Adapting Covering Arrays to Real Systems
2. Building Covering Arrays
   1. Overview
   2. One-row-at-a-time Greedy Algorithms
   3. Meta-heuristic Search

Combinatorial Results

- How do we know the covering array number (CAN)?
- How can we find the a subset of product instances that satisfies the properties of a covering array?

Results on Covering Arrays

There are two types of results:

1. Probabilistic:
   - We can prove a bound exists but can't necessarily create the subset. Only useful for finding the CAN.

Results on Covering Arrays

There are two types of results:

2. Constructive:
    - prove a new bound by giving a direct or algebraic construction.
        - Often recursive in nature.
        - Requires extensive mathematical knowledge.
        - Only works for certain values of t, k, v.

Constructive Results

- Although direct or algebraic constructions often give us the smallest subset for a covering array, they are not general.

- There are less constructions for mixed level covering arrays and none yet for variable strength arrays.

Computational Search

- These are constructive techniques.

  Pros:

  – They are general.

  – They extend easily to mixed level and can be adapted for variable strength, and seeds.

  Cons:

  – May not always give us the optimal CAN size.

  – May take a long computational time to find a CA.

Computational Search

- The problem of finding whether a minimal covering array exists for a given size is a difficult problem.
- Variants of this problem have been formulated and shown to be NP Hard problems.
- We cannot exhaustively search for a solution!

Algorithmic Techniques

Greedy algorithms:

- AETG - *Automatic Efficient Test Case Generator*
- TCG - *Test Case Generator*
- DDA -*Deterministic Density Algorithm*
- IPO - *In Parameter Order*

Heuristic/Meta-heuristic Search:

- Hill Climbing
- Simulated Annealing
- Genetic Algorithms
- Tabu Search

Some Available Tools

- *AETG* - *Telcordia*, Inc.: Commercial product

  http://aetgweb.argreenhouse.com/

- *WHITCH* - *Alan Hartman*: Eclipse plugin. It includes a tool called CTS (uses algebraic constructions) and a tool called tofu. You can plug in your own algorithms as well.

  http://alphaworks.ibm.com/tech/whitch

- *TestCover* - *George Sherwood*: Commercial tool: uses algebraic constructions. Has a student use license available.

  http://www.testcover.com/

- TConfig - *Alan Williams*: Uses algebraic constructions. Also includes an implementation of IPO.

  http://www.site.uottawa.ca/~awilliam/

Outline : Interaction Testing in Practice

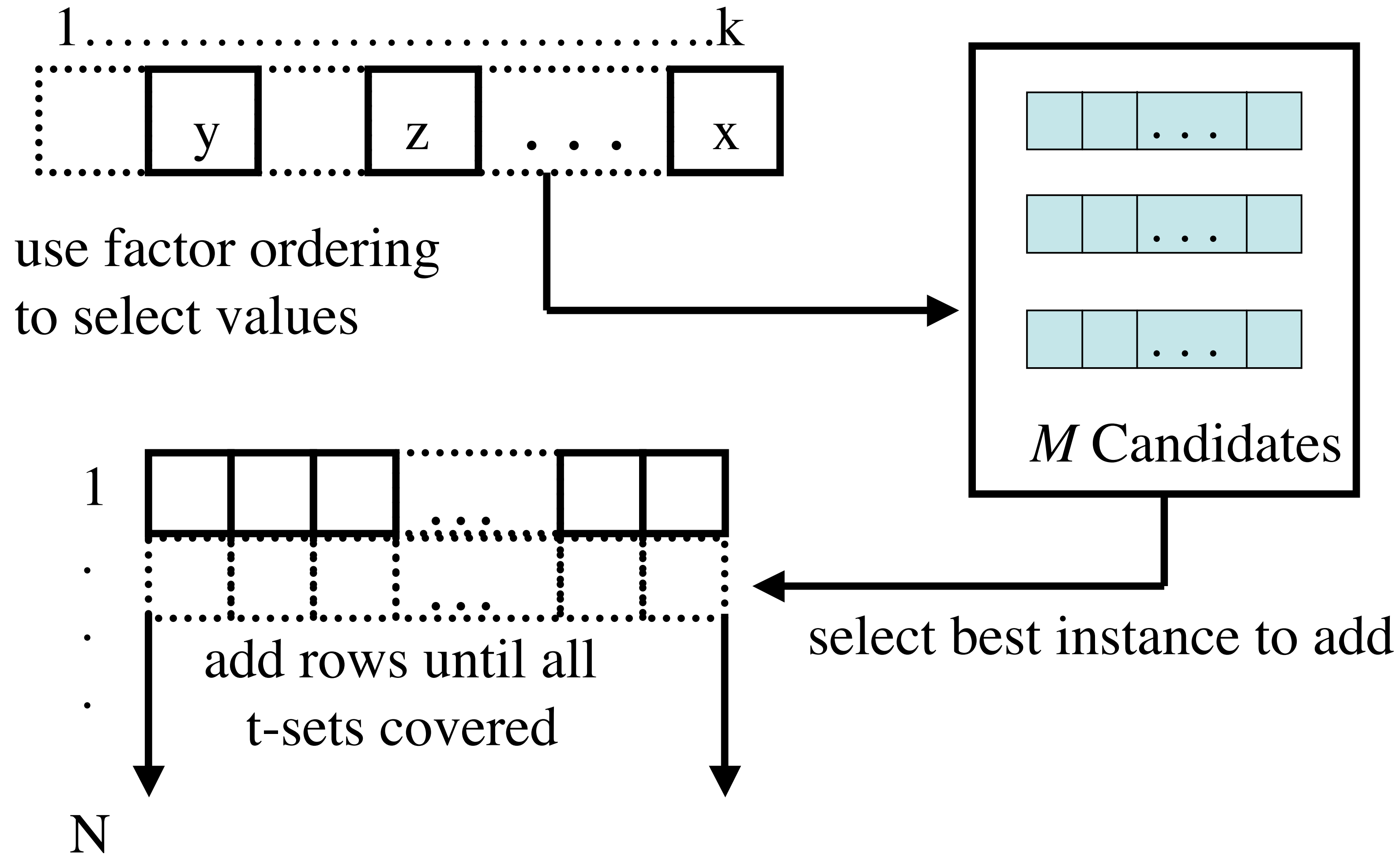
A Framework for Variability Coverage

1. Adapting Covering Arrays to Real Systems
2. Building Covering Arrays
   1. Overview
   2. One-row-at-a-time Greedy Algorithms
   3. Meta-heuristic Search

Greedy Algorithms

- One class of greedy algorithms add one-row at a time until the covering array is complete.
- The algorithms included in this brief overview include: AETG, TCG, DDA

One Row at a Time Greedy Algorithms



AETG

- The commercial AETG tool is patented.
- It uses some algebraic constructions, handles constraints and includes some post-processing steps.

- The original algorithm is presented next, but it does not include these improvements.

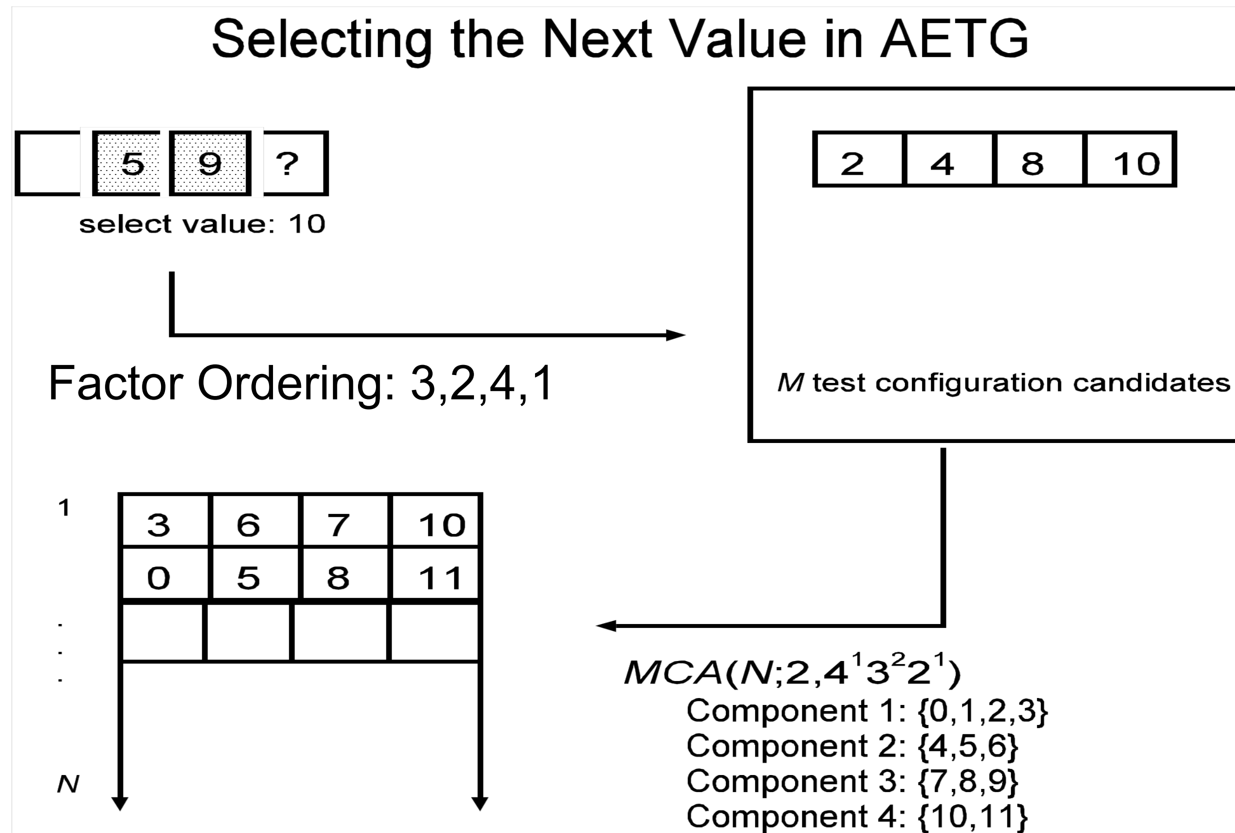
AETG Summary

- For each test case create *M* (50) new test cases.
- For each of the *M* test cases
  - Permute the order of the factors so that the first column has a symbol with the largest number of uncovered *pairs*. We fix this symbol and column. Randomly permute the rest.
  - Fill in the values for factors in the permutation order
    - For each factor choose the symbol that creates the most new pairs with the other columns already filled.
- Select the product instance (from *M*) which covers the most new *pairs*.

AETG

Pairs Covered:

3,6	0,5
3,7	0,8
3,10	0,11
6,7	5,8
6,10	5,11
7,10	8,11

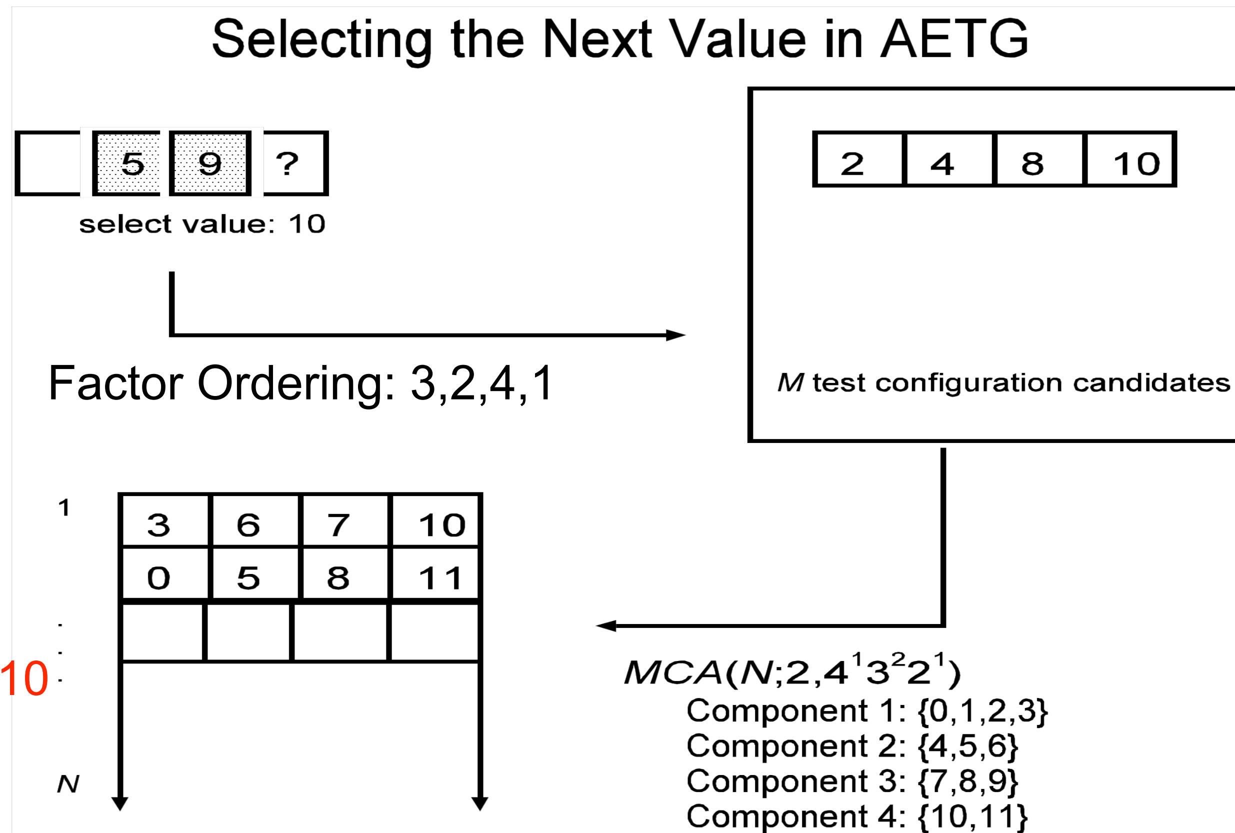


AETG

Pairs Covered:

3,6	0,5
3,7	0,8
3,10	0,11
6,7	5,8
6,10	5,11
7,10	8,11

SELECT Value 10



Other Members of the Framework

- TCG - Test Case Generator

  (Tung and Aldiwan)

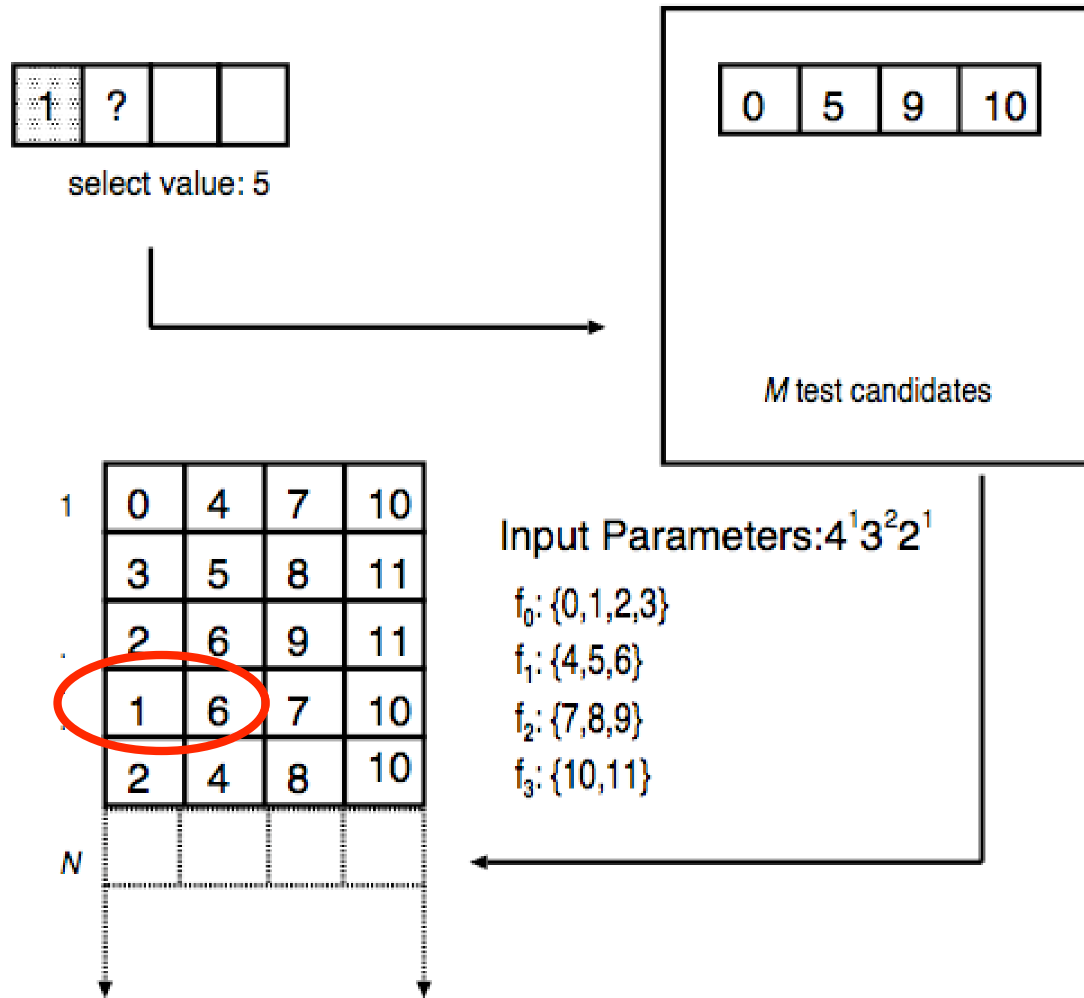
- DDA - Deterministic Density Algorithm

  (Colbourn, Cohen, Bryce)

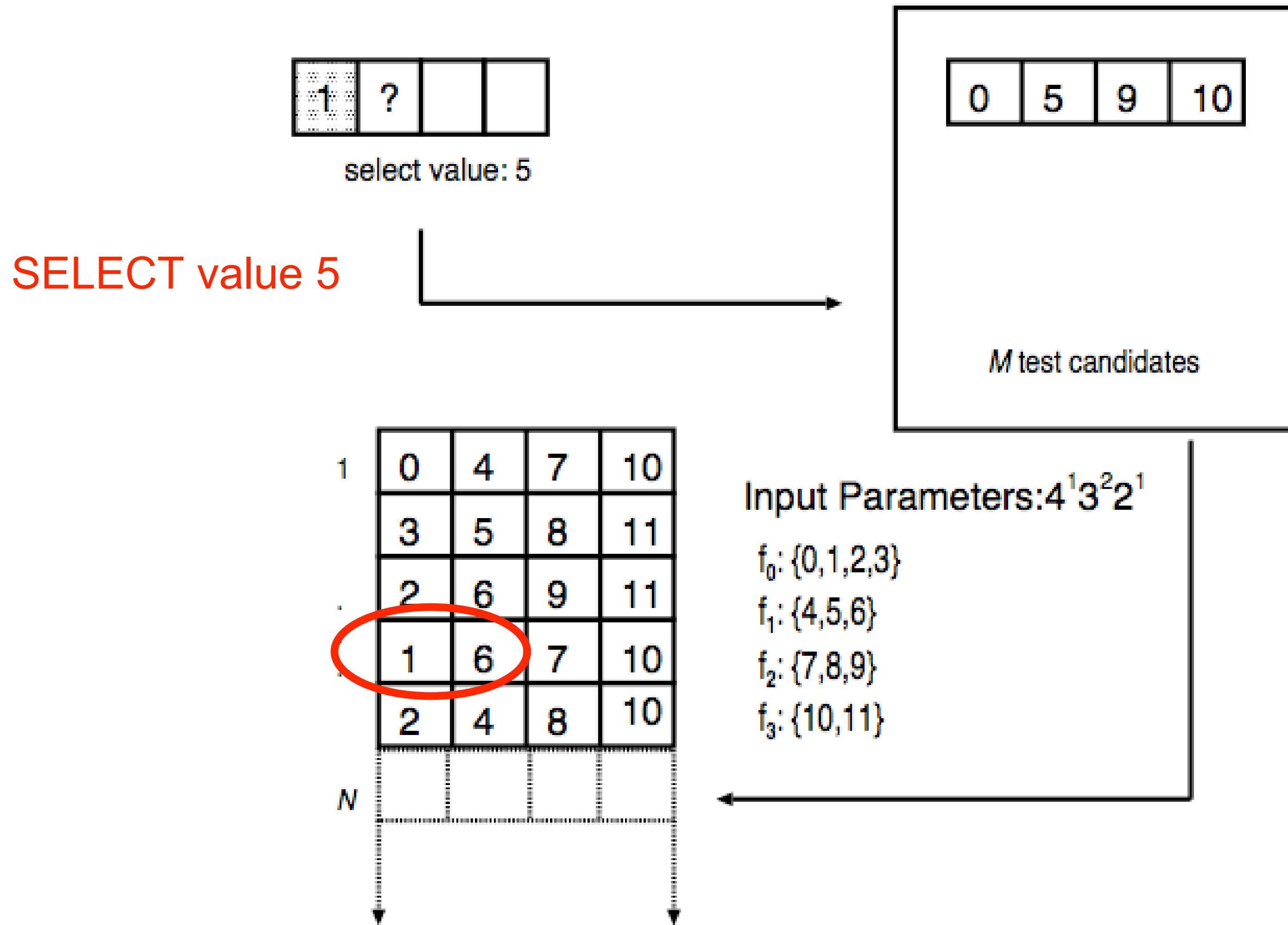
TCG

- Sorts factors in decreasing order of number of values.
- Always fills in the values in this order.
- When there is a tie between two values the one that has been used least often is selected.

TCG



TCG



Outline : Interaction Testing in Practice

A Framework for Variability Coverage

1. Adapting Covering Arrays to Real Systems
2. Building Covering Arrays
    1. Overview
    2. One-row-at-a-time Greedy Algorithms
    3. Meta-heuristic Search

Heuristic Search

- Heuristic techniques: methods that seek a good (close to optimal) solution using a reasonable amount of computational cost. These techniques cannot guarantee optimality and may not guarantee feasibility.

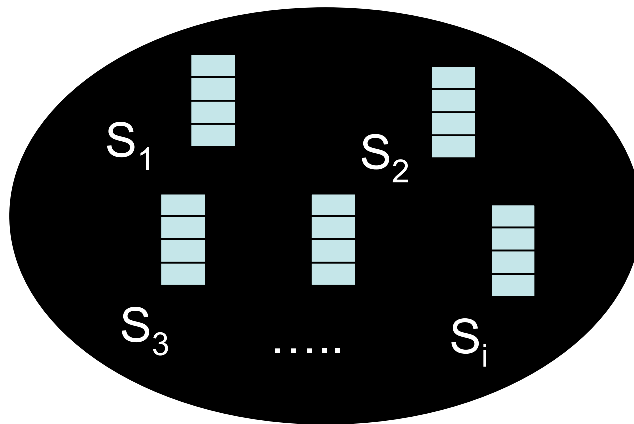
Heuristic Search

Combinatorial Optimization algorithms

1) Define a Set of Feasible solution
2) Each solution assigned a cost
3) Perform a series of transitions to new solutions chosen at random
4) If the new solution has the same or better cost – commit the change.
5) Otherwise don't commit.
6) We continue until we have an optimal cost or we are frozen

Hill Climbing

∑ -Set of feasible solutions



cost(S) : number of uncovered *t*-sets in S

When cost=0 we have a covering array

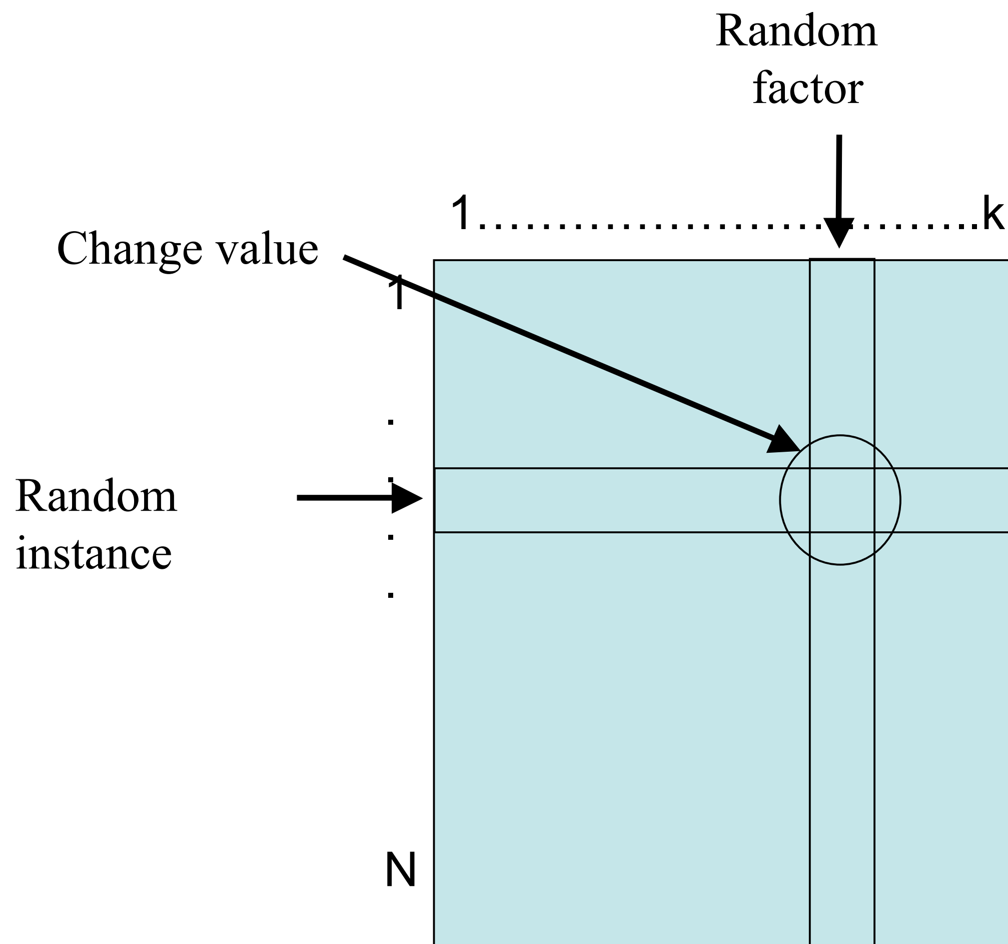
•Start with a random feasible solution

•Randomly choose/change one symbol to form a new feasible solution

Example

Cost

uncovered *t*-sets



Example of Hill Climb

CA(N;3,3,2)

t-sets

000 - 2
001 - 0
010 - 1
011 - 2
100 - 1
101 - 1
110 - 1
111 - 0

Cost = 2

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

Change?

Example of Hill Climb

CA(N;3,3,2)

t-sets

000 - 2

001 - 0

010 - 1

011 - ~~2~~ 1

100 - 1

101 - 1

110 - 1

111 - ~~0~~ 1

Cost = 1

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

Improves the Cost - keep this change

Example of Hill Climb

CA(N;3,3,2)

t-sets

000 - 2
001 - 0
010 - 1
011 - 2
100 - 1
101 - 1
110 - 1
111 - 0

Cost = 2

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

Change?

Example of Hill Climb

CA(N;3,3,2)

t-sets

000 - 2

001 - 0

010 - 1

011 - 2

100 - ~~1~~ 0

101 - 1

110 - ~~1~~ 2

111 - 0

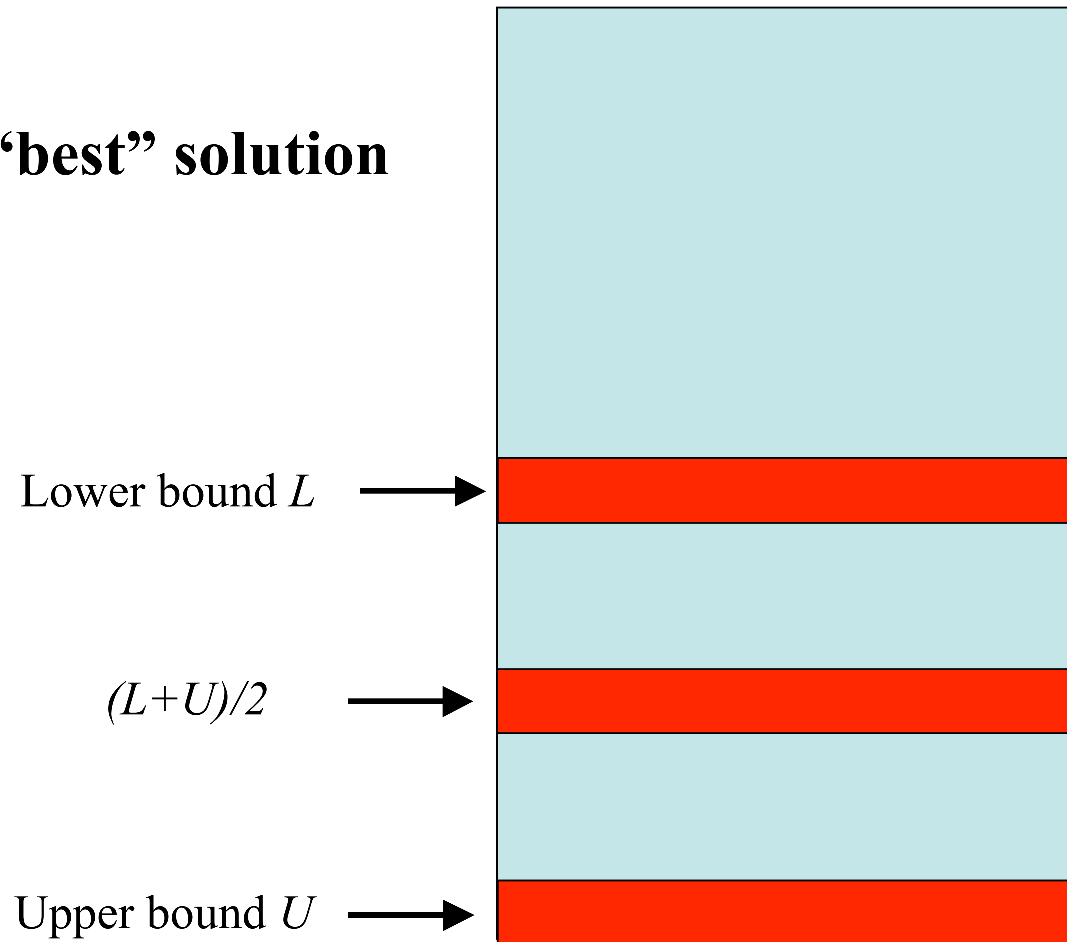
Cost = 3

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

Makes the solution worse! Do not keep

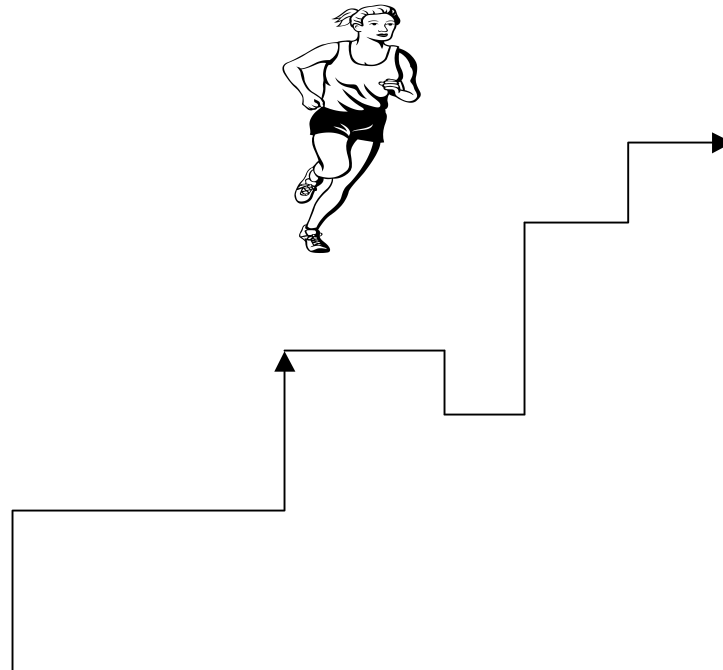
Finding “N”

Binary search for “best” solution



Problems with Hill Climbing

- May get stuck in local optimums.



Meta-heuristic Algorithms

- Provide mechanisms to escape local optimums. Sometimes we make accept a worse choice in the hope we will find a better route to a good solution.
- We want to control the probability of making a bad choice.

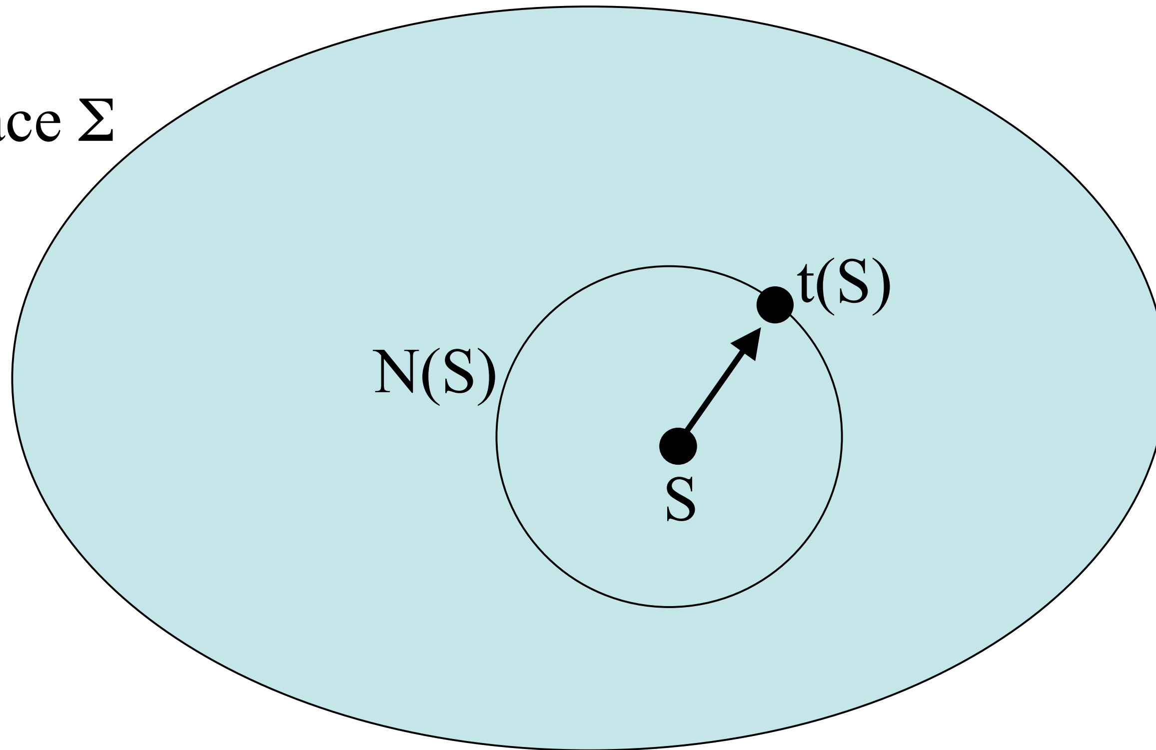
Some Examples

- Genetic Algorithms
- Simulated Annealing
- Tabu Search
- Ant Colony Algorithm

Simulated Annealing

- Based on the physical annealing process that is used to cool metal.
- Idea is to start at a high temperature. At each temperature the molecules stabilize and then the metal is cooled to the next temperature.

Simulated Annealing



Select t(S) such that

cost(t(S)) ≤ cost(S) ***or***

cost(t(S)) > cost(S) subject to controlled probability

Simulated Annealing

***Metropolis condition**:*

Accept bad move with probability $e^{-\Delta/T}$

*Design decisions (**cooling schedule**):*

(1) *Value of* T_0

(2) *Generating* T_k *from* T_{k-1}

(3) *Length of Markov chain* L_k

(4) *Stopping condition*

Other Meta-heuristic Search

- Genetic algorithms: population based search.
  (J. Stardom 2000)

- Tabu Search: steepest descent. Maintain a tabu list to prevent getting stuck in a cycle.
  (K. Nurmela 2004)

References

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 Transactions on Software Engineering*, vol. 23, no.7, pp. 437--444, 1997.

M. B. Cohen, C.J. Colbourn, J.Collofello, P.B. Gibbons, and W. B. Mugridge,Variable strength interaction testing of components in *'Proc. of 27th Intl. Computer Software and Applications Conference (COMPSAC)*, November 2003, pp. 413--418.

M.B. Cohen, C.J. Colbourn, P.B. Gibbons, and W.B. Mugridge, Constructing test suites for interaction testing, in Proc. of the Intl. Conf. On Software Engineering (ICSE), May 2003, pp. 38--48.

M. B. Cohen, C. J. Colbourn, and A. C. H. Ling, Constructing strength three covering arrays with augmented annealing, *Discrete Mathematics*, to appear.

M..B. Cohen, Designing test suites for software interaction testing, PhD Dissertation, University of Auckland, Department of Computer Science, 2004.

References

C.J. Colbourn, M. B. Cohen, and R..C. Turban, A deterministic density algorithm for pairwise interaction coverage, in *IASTED Proc. of the Intl. Conference on Software Engineering*, February 2004, pp. 345--352.

A.Hartman and L.Raskin, Problems and algorithms for covering arrays, *Discrete Math*, vol. 284, pp. 149 -- 156, 2004.

K.J. Nurmela, Upper bounds for covering arrays by tabu search, *Discrete Applied Mathematics*, vol. 138, no. 1-2, pp. 143--152, 2004.

J. Stardom, Metaheuristics and the search for covering and packing arrays, Master's thesis, Simon Fraser University, 2001

K..C. Tai and Y. Lei, A test generation strategy for pairwise testing, *IEEE Transactions on Software Engineering*, vol.28, no.1, pp. 109--111, 2002.

References

Y.-W. Tung and W. S. Aldiwan, Automating test case generation for the new generation mission software system,'' in *Proc. IEEE Aerospace Conf.*, 2000, pp. 431--437.

A. W. Williams, Determination of test configurations for pair-wise interaction coverage, in *Thirteenth Intl. Conf. Testing Communication Systems*, 2000, pp. 57--74.