Variable Strength Covering Arrays

Applications and Challenges

Myra Cohen

Laboratory for Empirically-based Software Quality Research and Development

Covering Arrays

CA_\lambda(N;\ell,k,v)
- An \( N \times k \) on \( v \) symbols array where each \( N \times \ell \) sub-array contains all ordered \( \ell \)-sets at least \( \lambda \) times.
- \( \ell \) is the strength of the array

\[
\begin{array}{cccc}
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 \\
\end{array}
\]

CA(6;2,5,2)

Mixed Level Covering Arrays

MCA_\lambda(N;\ell,k,(v_1,v_2,\ldots,v_k))
Is an \( N \times k \) array on \( v \) symbols where:
\[
v = \sum_{i=1}^{k} v_i
\]
And:
- For each column \( i \) where \( (i \leq i \leq k) \)
- The rows of each \( N \times \ell \) sub-array cover all \( \ell \)-tuples or values from the \( \ell \) columns at least \( \lambda \) times.

Shorthand Notation:
MCA_\lambda(N;\ell,w_1^{k_1}w_2^{k_1}w_3^{k_3})
e.g. MCA(12;2,4,\{4,3,2\}) = MCA(12;2,(4^1 3^2 2^1))

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

- Mixed level covering arrays have practical applications in software testing.

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

Motivation

Scenarios:

- When testing a software system certain components may be closely interrelated
- Operational profiles give us information that certain areas of the system are used more often than others
- In modifying a system only certain regions are changed therefore we want to test more strongly in this area
- Failures in certain parts of a system are costlier than in others

Possible Models

Variable Strength Covering Arrays

A 3 way array would have 18 rows
Variable Strength Covering Arrays

- A VCA(N; t, k, (v1, v2, ..., vk), 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.

Variable Strength Arrays Using SA

An Empirical Study

- Distributed Quality Assurance (Skoll)
  - Distribute instances of the system configuration for testing in the field
  - Results can be returned to a centralized location
  - Includes fault localization techniques

1. [Yilmaz, Cohen, Porter - 2006]
Fault Characterization Process

Identifies configuration options and their settings which are responsible for the manifestation of failures.

<table>
<thead>
<tr>
<th>Config</th>
<th>Result</th>
<th>Config</th>
<th>Result</th>
<th>Config</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>o1 0 0</td>
<td>PASS</td>
<td>o1 0 0</td>
<td>ERR #1</td>
<td>o1 0 0</td>
<td>ERR #1</td>
</tr>
<tr>
<td>o1 0 1</td>
<td>PASS</td>
<td>o1 0 1</td>
<td>ERR #1</td>
<td>o1 0 1</td>
<td>ERR #2</td>
</tr>
<tr>
<td>o1 0 2</td>
<td>PASS</td>
<td>o1 1 0</td>
<td>ERR #1</td>
<td>o1 1 0</td>
<td>ERR #2</td>
</tr>
<tr>
<td>o1 1 1</td>
<td>PASS</td>
<td>o1 1 1</td>
<td>ERR #1</td>
<td>o1 1 1</td>
<td>ERR #2</td>
</tr>
<tr>
<td>o1 1 2</td>
<td>PASS</td>
<td>o1 1 2</td>
<td>ERR #1</td>
<td>o1 1 2</td>
<td>ERR #2</td>
</tr>
<tr>
<td>o1 2 0</td>
<td>PASS</td>
<td>o1 2 0</td>
<td>ERR #1</td>
<td>o1 2 0</td>
<td>ERR #2</td>
</tr>
<tr>
<td>o1 2 1</td>
<td>PASS</td>
<td>o1 2 1</td>
<td>ERR #1</td>
<td>o1 2 1</td>
<td>ERR #2</td>
</tr>
<tr>
<td>o1 2 2</td>
<td>PASS</td>
<td>o1 2 2</td>
<td>ERR #1</td>
<td>o1 2 2</td>
<td>ERR #2</td>
</tr>
</tbody>
</table>

Covering Array Approach

- Systematically sample the configuration space, test only the selected configurations, and conduct fault characterization on the resulting data.
- How good are the resulting characterizations?

Software System

- ACE+TAO: an open source distributed CORBA middleware system
  - Large code base - 2M+ lines of C++ code
  - Over 500 configuration options
  - Dozens of OS, compiler and hardware platform combinations
Mappings

- 10 Static Binary Options (constraints reduce this to 92 feasible static configurations)
  - Only 29 of these compile successfully
    - Our model aggregates all of these into a single static option with 29 values
- 6 run time options with 2-4 values each

The Covering Array: \( MCA(N; t, 29^{14^{4}2^{1}}) \)

Software System

- Test suite: 96 regression tests
  - Each designed to emit an error message in the case of failure
  - The error messages were captured, indexed, and recorded
- Almost a year of machine time for the exhaustive testing of 18,792 configurations - just a small portion of actual space

Constructing Covering Arrays

- Created 5 different t-way covering arrays for \( 2 \leq t \leq 6 \)
- Size of covering arrays:
  \[ MCA(N; t, 29^{14^{4}2^{1}}) \]

<table>
<thead>
<tr>
<th>t</th>
<th># of configurations</th>
<th>% reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>116</td>
<td>99.4</td>
</tr>
<tr>
<td>3</td>
<td>348</td>
<td>98.2</td>
</tr>
<tr>
<td>4</td>
<td>1229-1236</td>
<td>93.5-93.4</td>
</tr>
<tr>
<td>5</td>
<td>3369-3372</td>
<td>82.1-82.0</td>
</tr>
<tr>
<td>6</td>
<td>9433-9453</td>
<td>49.8-49.7</td>
</tr>
</tbody>
</table>
Fault Localization

- Covering arrays performed better than random arrays of same size.
- Did almost as well as full configuration space at a reduced cost
(Use F Measure to determine how good our characterization is. It combines precision and recall)

But

Given:
- Many of the faults were localized in the runtime options
- We had a large number of options for the one static factor

Question:
- Can we improve our fault localization by using VCAs?

VCAs Created

<table>
<thead>
<tr>
<th></th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCA t=2</td>
<td>116</td>
</tr>
<tr>
<td>2c4r</td>
<td></td>
</tr>
<tr>
<td>VCA(2,294^3^2, MCA(4,4^3^2))</td>
<td>116</td>
</tr>
<tr>
<td>2c5r</td>
<td>324</td>
</tr>
<tr>
<td>MCA t=3</td>
<td>348</td>
</tr>
<tr>
<td>3c5r</td>
<td>367-368</td>
</tr>
</tbody>
</table>
### Results

<table>
<thead>
<tr>
<th>Failure</th>
<th>OS</th>
<th>2-way</th>
<th>2c4r</th>
<th>2c5r</th>
<th>3-way</th>
<th>3c5r</th>
<th>4-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-17</td>
<td>Linux</td>
<td>.78</td>
<td>.81</td>
<td>.83</td>
<td>.81</td>
<td>.83</td>
<td>.81</td>
</tr>
<tr>
<td>80-22</td>
<td>Linux</td>
<td>.34</td>
<td>.51</td>
<td>.65</td>
<td>.61</td>
<td>.65</td>
<td>.67</td>
</tr>
<tr>
<td>4-18</td>
<td>Win</td>
<td>.69</td>
<td>.79</td>
<td>.83</td>
<td>.84</td>
<td>.85</td>
<td>.88</td>
</tr>
</tbody>
</table>

**Simulation**

- Real data was encouraging but inconclusive
- Most of our characterizations were almost perfect at lower strengths - we may not have many high order faults.
- We performed a simulation of 4 way (runtime) faults in our system at varying levels of determinism.

**Constructing VCAs**

- (Biyani) - IBM internal tool (tofu)
- Simulated Annealing - M.B. Cohen et. al
- Constructions: new: C. Cheng 2006

Roux Reference:

- Roux, Gilbert, k-Propriétés dans des tableaux de n colonnes; cas particulier de la k-surjectivité et de la k-permutivite, PhD dissertation, University of Paris, Department of Mathematics, 1987.
Constructing VCAs

- If our model of VCA’s, we used a restricted model - the sub-arrays of higher strength are disjoint. We can easily adapt simulated annealing (or other algorithms) to build these.

- Use sum of the missing tuples across all strength arrays as the cost.

- At any point in time a change to an individual value of a factor in the array can effect only 2 CA’s - the overall array and the sub-array containing this factor.

Some Challenges

- Develop constructions and other computational techniques to build these:
  - Can we leverage the don’t care positions?
  - Do these need to be disjoint or can we build any VCA?

- Need a better notation and shorthand for describing VCAs.

Conclusions

- Variable strength arrays provide a way to model a software system that is flexible.

- We have successfully applied these to a real software system. (more work is being done on other systems....)

- We do not know a lot about bounds or constructing them.

- The model used to date may be too restrictive.

Acknowledgements

- This work was funded in part by an NSF EPSCoR FIRST award.