Combinatorial Test Set Generation

Combinatorial Testing
- Combinatorial Testing allows for efficient testing of large systems while maintaining certain coverage guarantees.
- In a combinatorial test set, every t-way interaction appears in at least λ tests, where t is the called strength and λ the index.
- In practice, greedy algorithms have proven the most versatile approach and are therefore used in many combinatorial test generators.

FIPOG and Tie-Breaker Evaluation
Various algorithmic and implementation-level improvements to the well-known In-Parameter-Order family of algorithms, including
- Simultaneous coverage gain computation.
- Skipping of fully covered column configurations.
- Partitioning of suitability checks.
- Compile-time strength.

Result: Vastly improved generation times.

![Figure 1: Speedup of FIPOG compared to the ACTS implementation of IPOG](image)

Further, different tie-breaking strategies were evaluated.

CAgem: A tool for Fast t-Way Test Set Generation
- t-way test set generation up to strength t = R.
- Implements the FIPOG, FIPOG-F and FIPOG-F2 algorithms.
- Support for constraints.
- Generation of test sets of higher index.
- Various export and import options.
- Compatible with other generation tools.

Requirements for Combinatorial Test Generation Tools:
- Fast generation
- Small number of tests
- Easy to use
- Many different features

The CAgem Web GUI

<table>
<thead>
<tr>
<th>Name</th>
<th>Values</th>
<th>Continuity</th>
</tr>
</thead>
<tbody>
<tr>
<td>PWH</td>
<td>1.23</td>
<td>23</td>
</tr>
<tr>
<td>J00</td>
<td>2.34</td>
<td>12</td>
</tr>
<tr>
<td>INT</td>
<td>3.45</td>
<td>13</td>
</tr>
<tr>
<td>PHS</td>
<td>4.56</td>
<td>14</td>
</tr>
<tr>
<td>J2</td>
<td>5.67</td>
<td>15</td>
</tr>
<tr>
<td>W2</td>
<td>6.78</td>
<td>16</td>
</tr>
<tr>
<td>DMW</td>
<td>7.89</td>
<td>17</td>
</tr>
</tbody>
</table>

Constraints:
- PWH = 23
- J00 = 12
- INT = 13
- PHS = 14
- J2 = 15
- W2 = 16
- DMW = 17

- In the Input Parameter Model tab, the model can be edited.

Future Work and HPC

Optimization Algorithms
- Devise new metaheuristic algorithms, mathematical constructions and post-optimization methods for CA generation.
- Combine our experience in the field to devise efficient hybrid heuristics.
- Enhance greedy algorithms using metaheuristics.
- Combine mathematical constructions and reductions with other generation techniques.
- Use Artificial Intelligence to enhance heuristic methods.
- Develop a hyper-heuristic framework.

High Performance Computing
- Develop scalable parallel algorithms.
- Use super computing for constructing combinatorial test sets.