Budget-aware Random Testing with T3
Benchmarking at the SBST2016 Testing Tool Contest

Wishnu Prasetya, Utrecht University
http://www.cs.uu.nl/~wishnu
https://git.science.uu.nl/prase101/t3/wikis/home
T3

- Random testing tool for Java Class
  - Provide convenient way for user to specify custom test data/generator

- Typical use case:
  - to fastly generate large amount of test sequences

- Test suites can be generated interactively
  - combined interactively: suite = suite1 + suite2
  - interactive query
  - analyzed, e.g. to infer invariants
Querying test suite

- \( H = \text{hoare}( \{ s \rightarrow s\text{.arg}[0] \leq s\text{.tobj.cutOff}() \}, \)
  
  "calculateTax",
  
  \( \{ s \rightarrow s\text{.retval} == 0 \} \)

- \text{ltlquery}(\text{suite}).with(\text{always}(H)).valid()

- \text{filter}(\text{suite}).with(\text{eventually}(H.\text{antecedent}()))
Budget aware suite generation

• Use case: running automated testing on a whole project, with an overall budget e.g. 1 hour.
• Current implementation: pre-calculated fixed budget per class, e.g. 1 minute.
• Class-level budgeting:
  • over inner classes
  • over test goals per target class
Test Goal

• Test goal: a public/protected method of CUT. \(\rightarrow\) generate a test suite for it.

• All TGs are put in a **worklist**, to be processed in some order
  • Process TG m: generate/refine its suite. If **not done**, put m back in the worklist.
    • There is a limit on the max. number of this put-backs (in the competition: set to 8)

• Repeat until either worklist is empty, or we run out of budget.
Refining suites

• Let m be a TG. We maintain a test suite $S_m$, generated for m so far. Generate new set of test sequences, each of the form:

$$\sigma \ ++ \ o.m(...) \ ++ \ \tau$$

• Only add a new sequence to $S_m$ if it improves coverage.
• Keeping in mind: proportionality.
Generating prefixes

• For efficiency, prefixes are generated collectively and incrementally over all TGs

• Maintain a set $P$ of prefixes so we have so far, and only grow it incrementally:
  • If all TGs of generation $k$ are processed, and worklist is not empty, we grow $P$ by generating $K$ fresh prefixes, but only adding those than can refine $P$.

• Refinement: also keep track “unique” object structures
  • project object structures to trees
  • project primitive values to logarithmic representations
Processing order policy of the TGs

• Random?

• Used policy:
  • when budget is still ok (0.5 B), we just pick the next TG randomly
  • after that “easier” TG is favored.
  • linear over generations, to enforce fairness
Overall budget policy

• CUT-level dynamic budget allocation:
  • Given a CUT and time budget $B_0$, determine the set of classes in CUT to target. Each $C$ gets is allocated a fragment of $B_0$, proportional to its complexity.
  • When we are done with $C$, budget allocation is re-calculated based on remaining time at that moment.

• T3 is tuned to use budget considerately, and not aggressively trying to exhaust all budget.
### Result

<table>
<thead>
<tr>
<th></th>
<th>60s</th>
<th>120s</th>
<th>240s</th>
<th>480s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C</td>
<td>M</td>
<td>T</td>
<td>C</td>
</tr>
<tr>
<td>RAN</td>
<td>54.0</td>
<td>64.1</td>
<td>1439</td>
<td>57.2</td>
</tr>
<tr>
<td>T3</td>
<td>59.2</td>
<td>74.4</td>
<td>1062</td>
<td>63.6</td>
</tr>
<tr>
<td>EVO</td>
<td>44.1</td>
<td>63.1</td>
<td>1410</td>
<td>50.2</td>
</tr>
<tr>
<td>JT</td>
<td>63.5</td>
<td>72.5</td>
<td>1653</td>
<td>68.1</td>
</tr>
</tbody>
</table>

On subset of 22 CUTs of the original 80 CUTs in the SBST2016 benchmark, on which no tools crash, and on which the benchmarking tool itself has no issue.
## Productivity

<table>
<thead>
<tr>
<th></th>
<th>60s</th>
<th>120s</th>
<th>240s</th>
<th>480s</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAN</td>
<td>0.14 (7)</td>
<td>0.06 (18)</td>
<td>0.03 (36)</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>0.51 (2)</td>
<td>0.15 (7)</td>
<td>0.06 (17)</td>
<td></td>
</tr>
<tr>
<td>EVO</td>
<td>0.31 (3)</td>
<td>0.28 (4)</td>
<td>0.07 (13)</td>
<td></td>
</tr>
<tr>
<td>JT</td>
<td>0.23 (4)</td>
<td>0.03 (32)</td>
<td>0.02 (48)</td>
<td></td>
</tr>
</tbody>
</table>

productivity = additional % coverage gained per additional minute spent.
Conclusion & future work

• When budget efficiency matters, enforcing a budget control algorithm makes sense.
• On big budget, T3’s BCA is justified to stop its effort.
• On low budget, T3’s BCA stops too early. Future work: smarter BCA.
• Future work: project-level BCA.