Integrating Artificial Intelligence in Software Testing

Roni Stern and Meir Kalech, ISE department, BGU
Niv Gafni, Yair Ofir and Eliav Ben-Zaken, Software Eng., BGU
Abstract

Artificial Intelligence

Planning  Diagnosis

Software Engineering

Testing
Testing → Planning → Diagnosis → Testing
Testing is Important

• Quite a few software development paradigms
• All of them recognize the importance of testing
• There are several types of testing
  – E.g., unit testing, black-box (functional) testing ....

Programmer
• Write programs
• Follows spec.
• Goal: fix bugs!

Tester
• Runs tests
• Follows a test plan
• Goal: find bugs!
Handling a Bug

1. **Bugs are reported** by the tester
   - “Screen A crashed on step 13 of test suite 4…”

2. **Prioritized** bugs are assigned to programmers

3. **Bugs are diagnosed**
   - What caused the bug?

4. **Bugs are fixed**
   - Hopefully…
Why is Debugging Hard?

- The developer needs to reproduce the bug.
- Reproducing (correctly) a bug is non-trivial:
  - Bug reports may be inaccurate or missing.
  - Software may have stochastic elements:
    - Bug occurs only once in a while.
  - Bugs may appear only in special cases.

Let the tester provide more information!
Introducing AI!

Tester
- Run a test suite
- Find a bug

AI
- Compute
- Diagnoses
- Plan next tests

Programmer
- Fix the bug

Diagnose
- Fix the bug
Test, Diagnose and Plan (TDP)

1. The tester runs a set of planned tests (test suite)
2. The tester finds a bug
3. AI generates possible diagnoses
4. If there is only one candidate – pass to programmer
5. Else, AI plans new tests to prune candidates
Diagnosis

Find the reason of a problem given observed symptoms

Requirement: knowledge of the diagnosed system

• Given by experts
• Learned by AI techniques
Model-Based Diagnosis

• Given: a model of how the system works
  ➔ Infer what went wrong

• Example:

  \[
  \text{IF ok(battery) AND ok(ignition) THEN start(engine)}
  \]

• What if the engine doesn’t start?
Where is MBD applied?

- Printers
  (Kuhn and de Kleer 2008)
- Vehicles
  (Pernestål et. al. 2006)
- Robotics
  (Steinbauer 2009)
- Spacecraft - Livingstone
  (Williams and Nayak, 1996; Bernard et al., 1998)
  ....
Software is Difficult to Model

- Need to code how the software should behave
  - *Specs are complicated to model*
- Static code analysis *ignores runtime*
  - Polymorphism
  - Instrumentation

...
Zoltar [Abrue et. al. 2011’]

- Construct a **model** from the **observations**
- Observations should include **execution traces**
  - Functions visited during execution
  - Observed outcome: bug / no bug
- **Weaker model**
  
  $$\text{Ok}(\text{function1}) \rightarrow \text{function1 outputs valid value}$$
  
  $$\Rightarrow \text{A bug entails that at least one comp was not Ok}$$
A key component in Zoltar is the **execution matrix**

It is built from the observed execution traces

- Observation 1 (BUG): F1→F5→F6
- Observation 2 (BUG): F2→F5
- Observation 3 (OK): F2→F6→F7→F8

<table>
<thead>
<tr>
<th></th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
<th>F8</th>
<th>Bug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obs1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Obs2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Obs3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Diagnosis = Hitting Sets of Conflicts

In every BUG trace at least one function is faulty

- **Observations:**
  - Observation 1 (BUG) : F1 $\rightarrow$ F5 $\rightarrow$ F6
  - Observation 2 (BUG) : F2 $\rightarrow$ F5

- **Conflicts:**
  - Ok(F1) AND Ok(F5) AND Ok(F6)
  - Ok(F2) AND Ok(F5)

- **Possible diagnoses:** {F5}, {F1,F2}, {F6,F2}
Software Diagnosis with Zoltar

Bug Report

Zoltar

Set of Candidate Diagnoses

Prioritize Diagnoses
Plan More Tests

Bug Report

AIEngine

Set of Possible Diagnoses

Suggest New Test to Prune Candidates
Plan More Tests

Bug Report → AIEngine → A single diagnosis → Suggest New Test to Prune Candidates

AI@BGU
Pacman Example
Pacman Example
Pacman Example
Pacman Example
Pacman Example
Pacman Example
Pacman Example
Execution Trace

F1 Move

F2 Stop

F3 Eat Pill
Which Diagnosis is Correct?
Knowledge

- Most probable cause: Eat Pill
Knowledge

- Most probable cause: Eat Pill
- Most easy to check: Stop
Objective

Plan a sequence of tests to find the best diagnosis with minimal tester actions
1. Highest Probability (HP)

- Compute the prob. of every function $C_i$
  
  \[ \text{Prob. of candidates containing } C_i \]

- Plan a test to check the most probable function
1. Highest Probability (HP)

• Compute the prob. of every function $C_i$
  $= \text{Sum of prob. of candidates containing } C_i$

• Plan a test to **check the most probable function**
2. Lowest Cost (LC)

- Consider only functions $C_i$ with $0 < P(C_i) < 1$
- **Test the closest function** from this set
3. Entropy-based

- A test $\alpha$ = a set of components
  - $\Omega_+$ = set of candidates that assume $\alpha$ will pass
  - $\Omega_-$ = set of candidates that assume $\alpha$ will fail
  - $\Omega_?$ = set of candidates that are indifferent to $\alpha$
- Prob. $\alpha$ will pass = prob. of candidates in $\Omega_+$
- Choose test with lowest $\text{Entropy}(\Omega_+, \Omega_-, \Omega_?)$
  = highest information gain
4. Planning Under Uncertainty

• Outcome of test is unknown
  ➔ we would like to minimize expected cost

• Formulate as Markov Decision Process (MDP)
  – States: set of performed tests and their outcomes
  – Actions: Possible tests
  – Transition: $\Pr(\Omega_+)$, $\Pr(\Omega_-)$ and $\Pr(\Omega?)$
  – Reward (cost): Cost of performed tests

• Current solver uses myopic sampling
Preliminary Experimental Results

• Synthetic code

• Setup
  – Generated random call graphs with 300 nodes
  – Generate code from the random call graphs
  – Injected 10/20 random bugs
  – Generate 15 random initial tests

• Run until a diagnosis with prob. > 0.9 is found
  – Results on 100 instances
Preliminary Experimental Results

- HP and Entropy perform worse than (LC) and MDP
- Probability based (HP and Entropy) perform worse than the cost based approach.
- MDP which considers cost and probability outperforms the others.
- 20 bugs is more costly for all algorithms than 10 bugs.

<table>
<thead>
<tr>
<th>Prob.</th>
<th>MDP</th>
<th></th>
<th>LC</th>
<th></th>
<th>Entropy</th>
<th></th>
<th>HP</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>20</td>
<td>10</td>
<td>20</td>
<td>10</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>0.1</td>
<td>7.5</td>
<td>10.1</td>
<td>11.0</td>
<td>10.6</td>
<td>24.3</td>
<td>63.0</td>
<td>29.4</td>
<td>87.4</td>
</tr>
<tr>
<td>0.2</td>
<td>13.2</td>
<td>15.7</td>
<td>16.9</td>
<td>20.4</td>
<td>39.3</td>
<td>92.7</td>
<td>38.5</td>
<td>115.7</td>
</tr>
<tr>
<td>0.3</td>
<td>15.7</td>
<td>19.4</td>
<td>20.2</td>
<td>24.8</td>
<td>43.2</td>
<td>99.4</td>
<td>44.7</td>
<td>118.8</td>
</tr>
<tr>
<td>0.4</td>
<td>17.2</td>
<td>22.1</td>
<td>22.6</td>
<td>30.0</td>
<td>49.2</td>
<td>103.3</td>
<td>52.7</td>
<td>120.7</td>
</tr>
<tr>
<td>0.5</td>
<td>20.1</td>
<td>24.9</td>
<td>27.5</td>
<td>35.8</td>
<td>56.2</td>
<td>107.0</td>
<td>71.3</td>
<td>130.8</td>
</tr>
<tr>
<td>0.6</td>
<td>23.8</td>
<td>25.6</td>
<td>31.1</td>
<td>37.7</td>
<td>57.8</td>
<td>111.0</td>
<td>71.8</td>
<td>132.7</td>
</tr>
<tr>
<td>0.7</td>
<td>25.8</td>
<td>26.8</td>
<td>31.2</td>
<td>38.7</td>
<td>65.1</td>
<td>111.9</td>
<td>74.3</td>
<td>133.9</td>
</tr>
<tr>
<td>0.8</td>
<td>26.0</td>
<td>32.2</td>
<td>32.4</td>
<td>39.9</td>
<td>69.9</td>
<td>115.3</td>
<td>77.6</td>
<td>135.3</td>
</tr>
<tr>
<td>0.9</td>
<td>26.9</td>
<td>33.0</td>
<td>35.0</td>
<td>42.2</td>
<td>73.0</td>
<td>143.9</td>
<td>79.0</td>
<td>136.4</td>
</tr>
</tbody>
</table>
Preliminary Experimental Results - NUnits

- Real code
- Well-known testing framework for C#
- Setup
  - Generated call graphs with 302 nodes from NUnits
  - Injected 10, 15, 20, 25, 30 random bugs
  - Generate 15 random initial tests
- Run until a diagnosis with prob. > 0.9 is found
  - Results on 110 instances
### Preliminary Experimental Results - NUnits

<table>
<thead>
<tr>
<th>Prob.</th>
<th>MDP</th>
<th>LC</th>
<th>Entropy</th>
<th>HP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>2.34</td>
<td>1.92</td>
<td>4.00</td>
<td>5.15</td>
</tr>
<tr>
<td>0.2</td>
<td>3.16</td>
<td>2.50</td>
<td>5.79</td>
<td>7.06</td>
</tr>
<tr>
<td>0.3</td>
<td>4.18</td>
<td>3.54</td>
<td>15.42</td>
<td>17.47</td>
</tr>
<tr>
<td>0.4</td>
<td>4.88</td>
<td>7.00</td>
<td>50.70</td>
<td>66.58</td>
</tr>
<tr>
<td>0.5</td>
<td>5.08</td>
<td>7.00</td>
<td>50.75</td>
<td>66.75</td>
</tr>
<tr>
<td>0.6</td>
<td>6.45</td>
<td>8.06</td>
<td>53.85</td>
<td>70.14</td>
</tr>
<tr>
<td>0.7</td>
<td>6.45</td>
<td>8.06</td>
<td>53.85</td>
<td>70.14</td>
</tr>
<tr>
<td>0.8</td>
<td>6.45</td>
<td>8.09</td>
<td>53.85</td>
<td>70.14</td>
</tr>
<tr>
<td>0.9</td>
<td>6.45</td>
<td>8.17</td>
<td>55.69</td>
<td>70.41</td>
</tr>
</tbody>
</table>

- Similar results
- The cost increases with the probability bound
- **MDP which considers cost and probability outperforms the others**
Conclusion

• The test, diagnose and plan (TDP) paradigm:
  – **AI diagnoses observed bug** with MBD algorithm
  – **AI plans further tests** to find correct diagnosis

• Empowers tester using AI
  – Bug diagnosis done by tester+AI