Identifying Test-Suite-Overfitted Patches through Test Case Generation

Qi Xin and Steven P. Reiss
Brown University
Automated Program Repair

Faulty Program

Test Suite

Patched Program

Passed Test Suite
Automated Program Repair

Faulty Program → Test Suite → Weak

Patched Program → Passed Test Suite

Bug not actually fixed
Automated Program Repair

Faulty Program

Test Suite

Test-Suite-Overfitted (Overfitting)

Patched Program

Passed Test Suite

Bug not actually fixed
An Overfitting Patch

if (fa * fb >= 0.0) { throw new ConvergenceException(...); }

if (fa * fb > 0.0) { throw new ConvergenceException(...); }

if (fa * fb >= 0.0) { throw new ConvergenceException(...); }
The Overfitting Problem

• Early repair techniques (GenProg, AE, and RSRepair)
  – Generated patches are often overfitting (Smith et al. 15)
  – The majority of generated patches are overfitting (Qi et al. 15)

• The state-of-the-art repair techniques (SPR & Prophet)
  – More than 60% first-found patches are overfitting (Long & Rinard 15, 16)
A Study by Long & Rinard

Search Space of test-passing patches

Correct Patch

Overfitting Patch
A Study by Long & Rinard

Hard to Isolate a Correct Patch from so many Overfitting ones
Our Approach

Identifying an Overfitting Patch through Test Case Generation
Our Approach

faultprog  patchprog

Faulty & Patched Programs
Our Approach

INPUT

faultprog  i  test generator  patchprog

Faulty & Patched Programs
Our Approach

INPUT

faultprog

patchprog

OUTPUT

\[ O_{fp} \neq O_{pp} \]

Faulty & Patched Programs

test generator
Our Approach

If incorrect, then overfitting

Faulty & Patched Programs

test generator
Our Approach

INPUT

FAULTY & PATCHED PROGRAMS

OUTPUT

If incorrect, then overfitting

test case
DiffTGen

• Input
  – Faulty program \( \text{faultprog} \)
  – Patched program \( \text{patchprog} \)
  – Syntactic modifications \( \Delta_{\text{syn}} \)
    • \( \delta_{\text{syn}} = \langle \text{faultstmt}, \text{patchstmt} \rangle \)

• Output
  – Test Case
  – Instrumented faulty program (for testing)
DiffTGen

• Work in three stages:
  1. Test Target Generation
  2. Test Method Generation
  3. Test Case Generation
Test Target Generation

patchprog

patchstmt

patchstmt
dummystmt

Targetprog

Coverage Goal
Test Target Generation

Coverage Goal

A test input covering `dummystmt` can exercise a $\delta_{\text{syn}}$
Test Method Generation

patchstmt
__________

patchprog

patchstmt

patchstmt
dummystmt

targetprog

test generator

test method
Test Method Generation

May not expose a semantic difference b/w faultprog & patchprog
Test Method Generation

faultprog \overset{?}{\rightarrow} \text{test method} \overset{?}{\rightarrow} \text{patchprog}

O_{fp} \neq O_{pp}
Test Method Generation

\[
\text{faultprog} \quad \rightarrow \quad \text{faultprog}_1 \quad \rightarrow \quad \text{test method} \quad \rightarrow \quad \text{patchprog}_1 \quad \rightarrow \quad \text{patchprog}
\]

\[
O_{fp} \quad \neq \quad O_{pp}
\]
Test Method Generation

Instrumentation (x to $x_1$):
Create printing statements for arguments, return value, and throwables of $\delta_m$
Test Case Generation

\[ o_{fp} \neq o_{pp} \]

Oracle

\[ \checkmark \text{ or } \times \]

\[ \times \rightarrow \text{Overfitting} \]

\[ o_{fp} \text{ is } \checkmark, \ o_{pp} \text{ is } \times: \text{Regression-Containing} \]

\[ o_{fp} \text{ is } \times, \ o_{pp} \text{ is } \times: \text{Defective} \]
Test Case Generation

Test method

faultprog

faultprog_2

O_e

 oracle

≠

O_{fp}  O_{pp}

✔ or ✗

✗ → Overfitting

test case
Empirical Evaluation

• RQ1
  – Can DiffTGen identify overfitting patches effectively?

• RQ2
  – Can DiffTGen help a repair technique produce less overfitting patches and more correct ones?
Experimental Setup

• Patch Dataset
  – 89 patches collected from two repair experiments (by Martinez et al. and by Le et al.)
    • Generated by four repair techniques
    • Generated for bugs from Defects4J dataset
  – Manually determine 10 patches to be non-overfitting
    • By comparing them to the developer patches from Defects4J
    • The other 79 patches are likely to be overfitting
Experimental Setup

- **DiffTGen Implementation**
  - $\text{faultprog, patchprog, } \Delta_{\text{syn}}$
    - Manually provided
  - Oracle
    - bug-fixed program (developer-patched program) from Defects4J
- Test generator
  - EvoSuite-1.0.2
- Test method generation
  - Run EvoSuite in 30 trials with the timeout being 60s for each trial
DiffTGen Running Result (#bugs in total: 89)

<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>SynDiff</td>
<td>72</td>
</tr>
<tr>
<td>SemDiff</td>
<td>61</td>
</tr>
<tr>
<td>Overfitting</td>
<td>39</td>
</tr>
<tr>
<td>Regression</td>
<td>34</td>
</tr>
<tr>
<td>Defective</td>
<td>18</td>
</tr>
</tbody>
</table>

Avg. Running Time: 6.9m

Can identify 49.4% (39/79) overfitting patches in less than 7 minutes on average
Empirical Evaluation

• RQ1
  – Can DiffTGen identify overfitting patches effectively?

• RQ2
  – Can DiffTGen help a repair technique produce less overfitting patches and more correct ones?
Experimental Process

• For each of the 39 bugs whose patches were found to be overfitting,
  – we added the generated test case to the original test suite
  – then …
Experimental Process

Repeat 1 & 2 until either fails
The correct patches do not exist in the repair techniques’ search spaces, and no patches were generated.
The correct patches **do exist**, but the repair techniques failed to find them.
Correct patches were generated through using HDRepair & DiffTGen
Result

Iterative Repair Results

- No Patch Generated (Expected)
- No Patch Generated (Unexpected)
- Correct Patch Generated
- Incorrect Patch Generated

Invalid patches generated for one bug, and DiffTGen failed to identify three patches generated for two bugs as overfitting
RQ2

• Based on our results,
  • DiffTGen can help a repair technique avoid yielding an overfitting patch
RQ2

• Based on our results,
  • DiffTGen can help a repair technique avoid yielding an overfitting patch
  • Often times, a repair technique fails to generate a patch using the augmented test suite
RQ2

• Based on our results,
  • DiffTGen can help a repair technique avoid yielding an overfitting patch
  • Often times, a repair technique fails to generate a patch using the augmented test suite
  • Whether a repair technique can eventually produce a correct patch largely depends on its repair ability
Conclusion

- Current automated program repair technique suffers from patch overfitting
- DiffTGen can identify an overfitting patch through test case generation
- The generated test case can help a repair technique avoid yielding an overfitting patch
Future Work

• Use a human oracle
  – Need to be human-amenable
• Use more sophisticated differential testing techniques
• Use more repair techniques for experiments
Automated Program Repair

Our Approach

Test-Suite-Overfitted (Overfitting)

Faulty Program

Test Suite

Patched Program

Passed Test Suite

Bug not actually fixed

INPUT

OUTPUT

faultprog

patchprog

Oracle

If incorrect, then overfitting

test generator

Faulty & Patched Programs

INPUT

OUTPUT

i

ofp

opp

oe

test case

Result

Iterative Repair Results

Can identify 49.4% (39/79) overfitting patches in less than 7 minutes on average

DiffTGen Running Result (#bugs in total: 89)

Avg. Running Time: 6.9m

#SynDiff  #SemDiff  #Overfitting  #Regression  #Defective

72  61  39  34  18

No Patch Generated (Expected)

No Patch Generated (Unexpected)

Correct Patch Generated

Incorrect Patch Generated

33

1

3

3

3

1

1