Feedback-controlled Random Test Generation

Kohsuke Yatoh\textsuperscript{1*}, Kazunori Sakamoto\textsuperscript{2}, Fuyuki Ishikawa\textsuperscript{2}, Shinichi Honiden\textsuperscript{12}

\textsuperscript{1}:University of Tokyo, \textsuperscript{2}:National Institute of Informatics

* He is currently affiliated with Google Inc., Japan. All work is done in Univ. Tokyo and nothing to do with Google.
My First Motivation

Software testing

• Very important
• Tedious, labor-intensive and error-prone

I want someone ELSE to write tests for me!

→ Automatic Test Generation
Two Sides of Automated Test Generation

1. Input generation (data)
   Generating interesting test data

2. Output verification (assertions)
   Oracles – specifications, domain specific knowledge

This paper
Background

Feedback-directed random test generation (FDRT) [Pacheco.07]

Random test generation for OOP languages

Classes under test \rightarrow FDRT \rightarrow Random method sequences

Usage

- Test by contracts [Pacheco.07]
- Regression test gen. [Robinson.11]
- Specification mining [Pradel.12]
- Test by property [Yatoh.14]
- Combination with other automated test generation [Garg.13, Zhang.14]
Example

Input: Class list

```java
class AddressBook {
    AddressBook(int capacity) {
        assert capacity >= 0;
        ...
    }
    void add(Person person) {...}
}

class Person {
    Person(String name) {
        assert name != null;
        ...
    }
}
```

Output: Method sequences

```java
AddressBook a1 =
    new AddressBook(10);
Person p1 =
    new Person("foo");
a1.add(p1);
//AddressBook a2 =
//    new AddressBook(-1);
//Person p2 =
//    new Person(null);
Person p3 =
    new Person("bar");
a1.add(p3);
a1.add(p1);
```
FDRT Pros & Cons

Good

Applicable to wider range of SUT than other methods like symbolic execution

Bad

Coverage of generated tests are low and unstable
→ less possibility to detect faults

Our Contributions

1. Analyzed characteristics of FDRT and found one cause of low and unstable coverage

2. Proposed a new method to mitigate the low coverage (Feedback-controlled Random Test Generation)
→ 2x - 3x coverage for utility libraries
FDRT Algorithm

Classes Under Test

class Person {
    Person(String name) {
    }
    bool equals(Person p) {
    }
}

Value Pool

“foo”, “bar”, 1, -1, true, false, ...

Pool of Candidate Arguments
(Initialized with random primitives)
FDRT Algorithm

Classes Under Test

```java
class Person {
    Person(String name) {
    }...
    bool equals(Person p) {
    }...
}
```

1. Choose Method `Person()`

2. Choose Argument "foo"

3. Save Return Value `p1`

Value Pool

"foo", "bar", 1, -1, true, false,…

Person `p1 = new Person(“foo”);`
FDRT Algorithm

**Classes Under Test**

```java
class Person {
    Person(String name) {
    }...
    bool equals(Person p) {
    }...
}
```

1. Choose Method
   Person()

2. Choose Argument
   “bar”

3. Save Return Value
   p2

```
Value Pool
“foo”, “bar”, 1, -1, true, false, p1, ...
```

```
Person p2 = new Person(“bar”);
```
FDRT Algorithm

Classes Under Test

class Person {
    Person(String name) {
        ...
    }
    bool equals(Person p) {
        ...
    }
}

1. Choose Method
equals()

Value Pool

“foo”, “bar”, 1, -1, true, false, p1, p2, ...

2. Choose Argument
p1, p2

3. Save Return Value
b1

bool b1 = p1.equals(p2);
FDRT Algorithm

Value Pool

“foo”, “bar”, 1, -1, true, false, p1, p2, ...

Feedback

1. Choose Method
   equals()

2. Choose
   Argument
   p1, p2

3. Save
   Return Value
   b1

bool b1 = p1.equals(p2);
Problems When Applying to Real Libraries

1. Low test coverage

2. Unstable dependency on seed
Cause of Low and Unstable Coverage

Positive feedback loop of FDRT
⇒ Bias grows in pool
⇒ Less diversity of generated tests

Bias in pool is amplified by feedback (e.g. List)
Proposed Method

Feedback-controlled Random Test Generation

- Keep diversity by multiple pools
  - Hold multiple pools at the same time
  - Use multiple pools concurrently

- Promote diversity by manipulating pools
  1. Select pool
  2. Add pool
  3. Delete pool
  4. Global reset

14
Keep Diversity by Multiple Pools

• Hold multiple pools at the same time
  Each pool may be biased, but keep diversity as whole

• Use multiple pools concurrently (in turn)
  Enable pool manipulation described later

Single pool

Set of pools

Original method

Proposed method
Promote Diversity by Manipulating Pools

1. **Select pool**
   Prioritize pools by ‘score’ function
   (High priority for pools that are likely to archive higher coverage)

2. **Add pool**
   Add new pools dynamically

3. **Delete pool**
   Delete similar pools using ‘uniqueness’ function

4. **Global reset**
   Reset all pools + Restart JVM

See the paper for the definition of score and uniqueness function
Evaluation

Compared 3 methods

- baseline: FDRT, one run
- reset: FDRT, reset every 100 sec.
- control: Proposed method

SUT

- 8 popular Java libraries from MVNRepository

Configuration

- Generate tests using 3600 sec. and record coverage of generated tests
- Conduct experiments with 30 different random seeds

Xeon X5650 (2.67GHz), 100GB RAM, CentOS 7.0
Isolated by Docker Ubuntu 14.04 w/ OpenJDK 1.7
Results – after 3600 seconds

Pattern (1)

Pattern (2)

Pattern (3)

8 Libraries x 3 methods (baseline, reset, control)
(1) Large Utility Libraries

4 utility libraries with 50K ~ 200K LOC

Large improvement on average and variance of coverage

Random testing is semantically suitable for this kind of libraries

Commons Collections

Commons Lang
(2) Small Libraries

2 libraries with 10K LOC

Small improvement, as the original FDRT do very well

Improvement on increase speed
(3) Configuration-intensive Libraries

2 libraries (Database / Web server)

No improvement, very low coverage

Needs careful configuration to work properly

![H2](image1.png)

![Jetty Server Core](image2.png)
Summary

Problem
Low and unstable coverage of FDRT
Cause: Bias of pool due to positive feedback loop

Method
Feedback-controlled Random Test Generation
• Keep diversity by multiple pools
• Promote diversity by pool manipulation

Result
Three result patterns depending on SUT
• Large utility libraries: Large improvement
• Small libraries: Small improvement, Less time for fixed coverage
• Configuration-intensive libraries: No changes
Appendix
Bias and Limited Diversity

e.g. Black or non-black stone

class Stone {
    bool black;
    Stone(bool black) {...}
    bool isBlack() {...}
    Stone clone() {...}
}

# of generated stones

Bias

Feedback

Feedback

Larger Bias

non-black

black
1. Select Pool

- Select pool that is most likely to increase coverage
- Scoring function

\[
\text{score}(pool) = \begin{cases} 
\frac{\text{coverage}(pool)}{\text{consumedTime}(pool)} & (\text{coverage}(pool) > 0) \\
\infty & (\text{coverage}(pool) = 0)
\end{cases}
\]

Improves average coverage
2. Add Pool

• Add a new pool every 1 second
3. Delete Pool

- Delete pools with similar contents, when \( \# \text{pools} \) exceeds a threshold
- Uniqueness function

\[
\text{uniqueness}(\text{pool}) = \frac{\sum_{c \in \text{covered}(\text{pool})} \text{uniqueness}(\text{pool}, c)}{|\text{covered}(\text{pool})|}
\]

\[
\text{uniqueness}(\text{pool}, c) = \frac{\text{count}(c, \text{pool})}{\sum_{p \in \text{all pools}} \text{count}(c, p)}
\]

Improves (decreases) Variance of coverage
4. Global Reset

• Reset every pool and restart JVM
• In order to remedy effect of nondeterministic behaviors and JVM instability
Results

3 result patterns, depending on SUT property

<table>
<thead>
<tr>
<th></th>
<th>Name</th>
<th>LOC</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Commons Collections</td>
<td>58,186</td>
<td>Collections</td>
</tr>
<tr>
<td></td>
<td>Commons Lang</td>
<td>66,628</td>
<td>Core Utilities</td>
</tr>
<tr>
<td></td>
<td>Guava</td>
<td>129,249</td>
<td>Core Utilities</td>
</tr>
<tr>
<td></td>
<td>Commons Math</td>
<td>202,839</td>
<td>Math Libraries</td>
</tr>
<tr>
<td>2</td>
<td>Commons Codec</td>
<td>13,948</td>
<td>Base64 Libraries</td>
</tr>
<tr>
<td></td>
<td>Gson</td>
<td>12,216</td>
<td>JSON Libraries</td>
</tr>
<tr>
<td>3</td>
<td>H2 Database Engine</td>
<td>158,926</td>
<td>Embedded SQL Databases</td>
</tr>
<tr>
<td></td>
<td>Jetty Server Core</td>
<td>32,316</td>
<td>Web Servers</td>
</tr>
</tbody>
</table>
Related Work

• Adaptive random testing [Ciupa.08]
  • Similar concept as our approach (Avoid testing with similar values)
  • Heavy computation cost due to calculating distances between every generated values [Arcuri.11]

• Combination with Dynamic Symbolic Execution (DSE)
  • Use FDRT to create seed sequences for DSE [Bounimova.13, Zhang.14]
  • Alternatively execute FDRT and DSE [Garg.13]
    Replacing FDRT with our approach would improve the effectiveness and efficiency of these techniques