Mutational Robustness and Automatic Program Repair

Ethan Fast
SFI REU 2010

Mentor: Stephanie Forrest
YOU KNOW THIS METAL RECTANGLE FULL OF LITTLE LIGHTS?

Yeah.

I SPEND MOST OF MY LIFE PRESSING BUTTONS TO MAKE THE PATTERN OF LIGHTS CHANGE HOWEVER I WANT.

Sounds good.

BUT TODAY, THE PATTERN OF LIGHTS IS ALL WRONG!

Oh god! Try pressing more buttons!

It's not helping!
Automatic Program Repair via Genetic Programming

Weimer and Forrest

An optimization technique inspired by evolution
GP Program Repair

Input

Program source code
Regression tests
Test case illustrating bug

Process

Generate program variants
Run them on test cases
Selection, crossover, mutation

Output

New program that passes tests
Or, no solution
Representation

Individuals represented as ASTs
Weighted Path

A means of fault localization

```
if (a)
  ret a
```

```
print(a)
```

```
print(b)
```

Good Path
Weighted Path
A means of fault localization

```
if (a)
  ret a
print(a)
print(b)
```

Bad Path
Weighted Path
A means of fault localization

```plaintext
Prgm
  if (a)
    ret a
  print(a)
  print(b)
```

Final Path
Mutation: Swap

Exchange two nodes on the tree
Mutation: Append

Copy a node to elsewhere on the tree
Mutation: Delete

Delete a node from the tree
GP Program Repair Details

To compute fitness, compile a variant

If it fails to compile, then fitness = 0

Otherwise, run test cases

Now, fitness = # tests passed

Negative test case(s) more heavily weighted
Does it actually work?

<table>
<thead>
<tr>
<th>deroff</th>
<th>gcd</th>
<th>look</th>
</tr>
</thead>
<tbody>
<tr>
<td>indent</td>
<td>uniq</td>
<td>zune</td>
</tr>
<tr>
<td>atris</td>
<td>leukocyte</td>
<td>imagemagick</td>
</tr>
<tr>
<td>tiff</td>
<td>nullhttpd</td>
<td>python</td>
</tr>
<tr>
<td>php</td>
<td>lighttpd</td>
<td>openldap</td>
</tr>
</tbody>
</table>

A few repaired programs
So what about robustness?
Some Definitions

mutational robustness: the probability of a change in genotype affecting a change in phenotype

neutral fitness landscape: described by region of differing genotypes assigned the same fitness value
Motivation

High mutational robustness seems to support the idea of evolving software.

Robustness and neutral fitness may be key ideas for repairing more complicated bugs.
Questions

How do we measure robustness?

Given a metric, how mutationally robust are typical programs?

How does robustness affect automatic program repair?
Measuring Robustness

Original Program  Apply Mutation (×1000)

Metrics:
Average distance in fitness
Percent of mutations that are neutral
Suppose that we make a single mutation to some arbitrary program.

How often will its behavior change?
Neutral Mutations

Percent of Mutations that are Neutral

- deroff
- look
- uniq
- gcd
- indent
- leukocyte
What mutation operators are likely to result in neutral mutations?
So what about that weighted path?

Shouldn’t one look at programs more generally?
With and Without Path

Percent of Mutations that are Neutral

- deroff
- look
- uniq
- gcd
- leukocyte

with path vs without path
Robustness vs. Code Size

Lines of Code

Percent of Mutations that are Neutral
But perhaps my tests suites are simply quite terrible?

Do these results actually generalize?
Neutral Mutations on Large Suites

Percent of Mutations that are Neutral

- leukocyte: 39
- potion: 39
- vyquon: 29.25
- redis: 29.25
A Non-Trivial Test Suite

#001 AUTH fails when a wrong password is given
#002 Arbitrary command gives an error when AUTH is required
#003 AUTH succeeds when the right password is given
...
#045 RENAME where source and dest key is the same
#046 DEL all keys again (DB 0)
#047 DEL all keys again (DB 1)
#048 MOVE basic usage
#049 MOVE against key existing in the target DB
...
#255 SORT with BY against the newly created list
#256 SORT with BY (hash field) against the newly created list
#257 SORT with GET (key+hash) with sanity check of each element (list)
#258 SORT with BY, but against the newly created set
#259 SORT with BY (hash field), but against the newly created set
#260 SORT with BY and STORE against the newly created list
#261 SORT with BY (hash field) and STORE against the newly created list
By Mutation Operators

- append
- swap
- delete

Percent Neutral Mutations

leukocyte
potion
vyquon
redis
Stepping Back

Surprising to see such high levels of mutational robustness, at this level of representation

Possibly contributes * to the success of Program Repair via GP

Quite counter-intuitive (so we assert)

* robustness != good (tradeoff with evolvability)
Relating Robustness to Repair Difficulty
A Problem?

Fitness

- Easy
- OK
- Hard

Mutations to Repair
Three-Step Repair

A pathological case study
How might we solve this?

Use a **new** representation, with a **higher** degree of **mutational robustness**

Inspiration: a **diploid** chromosomal structure

Change the **gradient** of the fitness landscape leading to repair
The Basic Idea

![Graph showing fitness over mutations to repair with two lines: New Rep and Old Rep.](image)
New Representation

$V_a$  

$V_b$  

$V_{final}$

diploid structure
Mutations can be made to program segments that are *not applied functionally*. A *smoother* fitness gradient to repair

**Innovation**: Occasionally these non-functional mutations will be transformed into functional mutations
New Rep More Robust?

Percent neutral mutations

Old Rep | New Rep
---|---
deroff | 60 | 70
look | 80 | 70
uniq | 60 | 60
indent | 50 | 60
gcd | 70 | 70
leukocyte | 60 | 70
average | 60 | 70

New Rep More Robust?
Preliminary Results

Of a Mixed Nature

Two-step repair found $3x$ as often

Three-step repair never found

Working on Additional Strategies

Different representations

Fitness function
Conclusions

Programs are surprisingly robust

Result holds for large and complicated programs and test suites

But more robust representations may help in repairing certain kinds of bugs
Questions?

Suggestions are also welcome
```c
int main(int argv, char * argc[]) {
    int x = atoi(argv[1]);
    int p1 = 0;
    int p2 = 0;
    int p3 = 0;
    // p1 = 7;
    // p2 = 3;
    // p3 = 4;
    int now = p1+p2+p3;
    if(x == 1) {
        printf("%d:%d:%d\n", x, p1-p2-p3, now==p1+p2+p3);
    }
    if(x == 2) {
        printf("%d:%d:%d\n", x, p1-p2-p3, now==p1+p2+p3);
    }
    if(x == 3) {
        printf("%d:%d:%d\n", x, p1-p2-p3, now==p1+p2+p3);
    }
    if(x == 4) {
        printf("%d:%d:%d\n", x, p1-p2-p3, now==p1+p2+p3);
    }
    if(x == 5) {
        printf("%d:%d:%d\n", x, p1-p2-p3, now==p1+p2+p3);
    }
    if(x == 666) {
        printf("%d:%d:%d\n", x, p1+p2+p3, p1-p2-p3, now==p1+p2+p3);
    }
    p1 = 7;
    p2 = 3;
    p3 = 4;
}
```
Robustness Benchmark

<table>
<thead>
<tr>
<th>Program</th>
<th>MR *</th>
<th>Neutral **</th>
</tr>
</thead>
<tbody>
<tr>
<td>deroff</td>
<td>20%</td>
<td>34%</td>
</tr>
<tr>
<td>look</td>
<td>20%</td>
<td>40%</td>
</tr>
<tr>
<td>uniq</td>
<td>24%</td>
<td>38%</td>
</tr>
<tr>
<td>indent</td>
<td>16%</td>
<td>48%</td>
</tr>
<tr>
<td>gcd</td>
<td>23%</td>
<td>30%</td>
</tr>
<tr>
<td>leukocyte</td>
<td>19%</td>
<td>41%</td>
</tr>
</tbody>
</table>

* measured average change in test case fitness
** percent of mutations that do not affect fitness
# With Mutation Operators

<table>
<thead>
<tr>
<th>Program</th>
<th>MR *</th>
<th>Neutral **</th>
<th>Append</th>
<th>Swap</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>deroff</td>
<td>20%</td>
<td>31%</td>
<td>30%</td>
<td>11%</td>
<td>59%</td>
</tr>
<tr>
<td>look</td>
<td>20%</td>
<td>43%</td>
<td>40%</td>
<td>14%</td>
<td>46%</td>
</tr>
<tr>
<td>uniq</td>
<td>24%</td>
<td>34%</td>
<td>43%</td>
<td>14%</td>
<td>43%</td>
</tr>
<tr>
<td>gcd</td>
<td>23%</td>
<td>34%</td>
<td>55%</td>
<td>19%</td>
<td>25%</td>
</tr>
<tr>
<td>leukocyte</td>
<td>19%</td>
<td>39%</td>
<td>12%</td>
<td>23%</td>
<td>64%</td>
</tr>
</tbody>
</table>

* measured average change in test case fitness  
** percent of mutations that do not affect fitness
### With No Path Weights

<table>
<thead>
<tr>
<th>Program</th>
<th>Neutral</th>
<th>Append</th>
<th>Swap</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>deroff</td>
<td>60%</td>
<td>28%</td>
<td>20%</td>
<td>52%</td>
</tr>
<tr>
<td>look</td>
<td>53%</td>
<td>34%</td>
<td>15%</td>
<td>51%</td>
</tr>
<tr>
<td>uniq</td>
<td>55%</td>
<td>27%</td>
<td>17%</td>
<td>56%</td>
</tr>
<tr>
<td>gcd</td>
<td>37%</td>
<td>61%</td>
<td>11%</td>
<td>28%</td>
</tr>
<tr>
<td>leukocyte</td>
<td>39%</td>
<td>32%</td>
<td>13%</td>
<td>56%</td>
</tr>
</tbody>
</table>

Even more robust to random mutations
For Larger Test Suites?

<table>
<thead>
<tr>
<th>Program</th>
<th>Neutral</th>
<th>Append</th>
<th>Swap</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>leukocyte</td>
<td>35%</td>
<td>26%</td>
<td>14%</td>
<td>60%</td>
</tr>
<tr>
<td>potion</td>
<td>39%</td>
<td>18%</td>
<td>6%</td>
<td>76%</td>
</tr>
<tr>
<td>vyquon</td>
<td>32%</td>
<td>22%</td>
<td>4%</td>
<td>74%</td>
</tr>
<tr>
<td>redis</td>
<td>31%</td>
<td>26%</td>
<td>10%</td>
<td>64%</td>
</tr>
</tbody>
</table>

Seems not to be artifact of small test suites
New Rep More Robust?

<table>
<thead>
<tr>
<th>Program</th>
<th>Old Rep *</th>
<th>New Rep *</th>
</tr>
</thead>
<tbody>
<tr>
<td>deroff</td>
<td>34%</td>
<td>64%</td>
</tr>
<tr>
<td>look</td>
<td>40%</td>
<td>79%</td>
</tr>
<tr>
<td>uniq</td>
<td>38%</td>
<td>63%</td>
</tr>
<tr>
<td>indent</td>
<td>48%</td>
<td>54%</td>
</tr>
<tr>
<td>gcd</td>
<td>30%</td>
<td>69%</td>
</tr>
<tr>
<td>leukocyte</td>
<td>41%</td>
<td>66%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>38.5%</strong></td>
<td><strong>65.8%</strong></td>
</tr>
</tbody>
</table>

* percent of mutations that do not affect fitness
Weighted Path

GCD

if (a == 0)
{ block }
printf(...a)
return

while (b != 0)
{ block }
printf(...b)

if (a > b)
{ block }
a = a - b

Negative Test Case
Weighted Path

```
if (a == 0)
{ block }
printf(...a)
return

while (b != 0)
{ block }
if (a > b)
{ block }
a = a - b
{ block }
b = b - a
{ block }
printf(...b)
{ block }
```

Positive Test Case
Weighted Path

GCD

if (a == 0)

while (b != 0)

printf(...a)

return

{ block }

{ block }

printf(...b)

if (a > b)

{ block }

{ block }

a = a - b

b = b - a

Final Path
Swapping

\[ F(M_1 \text{ or } M_2) = 0 \]

\[ F_a = 5 \]

\[ F_b = 5 \]

\[ F_{\text{final}} = 15 \]