Fixing software bugs in 10 minutes or less using evolutionary computation

University of New Mexico
   Stephanie Forrest
   ThanhVu Nguyen
University of Virginia
   Claire Le Goues
   Westley Weimer
Summary of method

• Assume:
  • Access to C source code
  • Negative test case that executes the buggy code
  • Positive test cases to encode required program functionality
• Construct Abstract Syntax Tree (AST)
• Evolve repair that avoids negative test case and passes positive test case
• Minimize repair using program analysis methods
Repairing the Zune bug using GP

- Infinite loop when input is last day of a leap year.
- Microsoft sold about 1.2 million units of Zune 30, generating thousands of complaints.
- Repair is not trivial. Microsoft’s recommendation was to let Zune drain its battery and then reset.
- GP discovered the repair in 42 seconds.

```c
void zunebug_repair(int days) {
    int year = 1980;
    while (days > 365) {
        if (isLeapYear(year)) {
            if (days > 366) {
                // days -= 366; // repair deletes
                year += 1;
            } else {
                days -= 366;     // repair inserts
            } else {
                days -= 365;
                year += 1;
            }
        } printf("current year is %d\n", year);
    }
}
```

Downloaded from http://pastie.org/349916 (Jan. 2009).
Example repairs

<table>
<thead>
<tr>
<th>Program</th>
<th>Version</th>
<th>LOC</th>
<th>Time to Repair</th>
<th>Program Description</th>
<th>Fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcd</td>
<td>example</td>
<td>22</td>
<td>15s</td>
<td>handcrafted example</td>
<td>infinite loop</td>
</tr>
<tr>
<td>zune</td>
<td></td>
<td>28</td>
<td>42s</td>
<td>media player</td>
<td>infinite loop</td>
</tr>
<tr>
<td>uniq</td>
<td>ultrix 4.3</td>
<td>1146</td>
<td>34s</td>
<td>duplicate text processing</td>
<td>segfault</td>
</tr>
<tr>
<td>look</td>
<td>ultrix 4.3</td>
<td>1169</td>
<td>45s</td>
<td>dictionary lookup</td>
<td>segfault</td>
</tr>
<tr>
<td>look</td>
<td>svr4.0.1.1</td>
<td>1363</td>
<td>55s</td>
<td>dictionary lookup</td>
<td>infinite loop</td>
</tr>
<tr>
<td>units</td>
<td>svr4.0.1.1</td>
<td>1504</td>
<td>109s</td>
<td>metric conversion</td>
<td>segfault</td>
</tr>
<tr>
<td>deroff</td>
<td>ultrix 4.3</td>
<td>2236</td>
<td>131s</td>
<td>document processing</td>
<td>segfault</td>
</tr>
<tr>
<td>indent</td>
<td>1.9.1</td>
<td>9906</td>
<td>546s</td>
<td>source code processing</td>
<td>infinite loop</td>
</tr>
<tr>
<td>flex</td>
<td>2.5.4a</td>
<td>15775</td>
<td>230s</td>
<td>lexical analyzer generator</td>
<td>segfault</td>
</tr>
<tr>
<td>atris</td>
<td>1.0.6</td>
<td>21553</td>
<td>88s</td>
<td>graphical tetris game</td>
<td>loc. stack buffer exploit</td>
</tr>
<tr>
<td>nullhttpd</td>
<td>0.5.0</td>
<td>5575</td>
<td>578s</td>
<td>webserver</td>
<td>rem. heap buffer exploit</td>
</tr>
<tr>
<td>openldap io.c</td>
<td>2.3.4i</td>
<td>6519</td>
<td>666s</td>
<td>directory protocol</td>
<td>non-overflow DOS</td>
</tr>
<tr>
<td>lighthttpd fastcgi.c</td>
<td>1.4.17</td>
<td>13984</td>
<td>49s</td>
<td>webserver</td>
<td>rem. heap buf overflow</td>
</tr>
<tr>
<td>php string.c</td>
<td>5.2.1</td>
<td>26944</td>
<td>6s</td>
<td>scripting language</td>
<td>int overflow</td>
</tr>
<tr>
<td>wu-ftp</td>
<td>2.6.0</td>
<td>35109</td>
<td>2256s</td>
<td>FTP server</td>
<td>format string</td>
</tr>
<tr>
<td>total</td>
<td></td>
<td>144933</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Why is this human competitive?

- Software is THE (indisputably difficult) problem
- Time to discover a repair
- Quality of repair
Software is THE problem

- Software faults and debugging are expensive:
  - US corporate development organizations spend $5.2 - $22 million annually fixing software defects (IDC Software Quality Survey, 2008)
  - Cost of repairing bugs increases throughout the development process. A $25 fix while the program is under development increases to $16,000 after the software has gone live (IBM Rational group, 2008)

- Security violations are expensive:
  - Average total per-incident costs in 2008 were $6.65 million, compared to an average per-incident cost of $6.3 million in 2007.
  - Monetary loss by 639 companies in 2005 totaled $130 million (FBI 2005)
Software is THE problem

- Bugs are plentiful:
  - Mozilla project received 51,154 bug reports in 2002-2006.
  - In 2005, a Mozilla developer reported that “almost 300 bugs appear every day that need triaging.”
- Fixing bugs is time-consuming:
  - Industrial software repair: 1/2 of all fixed bugs in Mozilla from 2002-2006 took more than 29 days for developers to fix; Median repair time for ArgoUML project in 2002-2003 was 190 days; Median repair time per bug for PostgreSQL was 200 days.
Time to discover repair

• To date, we have repaired 15 programs totaling nearly 150,000 lines of code
• Average time to repair: 3 minutes (for first 11 programs shown)
• Time includes:
  • GP algorithm (selection, mutation, calculating fitness, etc.)
  • Running test cases
  • Pretty printing and memoizing ASTs
  • gcc (compiling ASTs into executable code)
Quality of repair

- Microsoft requires that security-critical changes be subjected to 100,000 fuzz inputs (randomly generated structured input strings).
- Used SPIKE black-box fuzzer (immunitysec.com) to generate 100,000 held-out fuzz requests for web server examples.
  - In no case did GP repairs introduce errors that were detected by the fuzz tests, and in every case the GP repairs defeated variant attacks based on the same exploit.
  - Thus, the GP repairs are not fragile memorizations of the input.
- GP repairs also correctly handled all subsequent requests from indicative workload.
Why should we win the prize?

• The idea of computational evolution (genetic algorithms) was introduced nearly 50 years ago (by JHH).
  • Led to many successes in engineering and science.

• Yet, the dream of “automatic programming” is still largely unfilled.
  • Why does the evolutionary approach to design work throughout nature and engineering, but not in software?

• A gap in the evolutionary record that needs to be filled.