UCL DEPARTMENT OF
COMPUTER SCIENCE

Research Note

RN/16/04

The Genetic Improvement Fitness Landscape

10 June 2016

William B. Langdon

Abstract

Trying all simple changes (first order mutations) to executed source code shows software engineering artefacts are more robust than is often assumed. Of those that compile, up to 89% run without error. Indeed a few non equivalent mutants are improvements.

Keywords

theory, genetic improvement, genetic algorithms, genetic programming, software engineering, SBSE, search, heuristic methods,

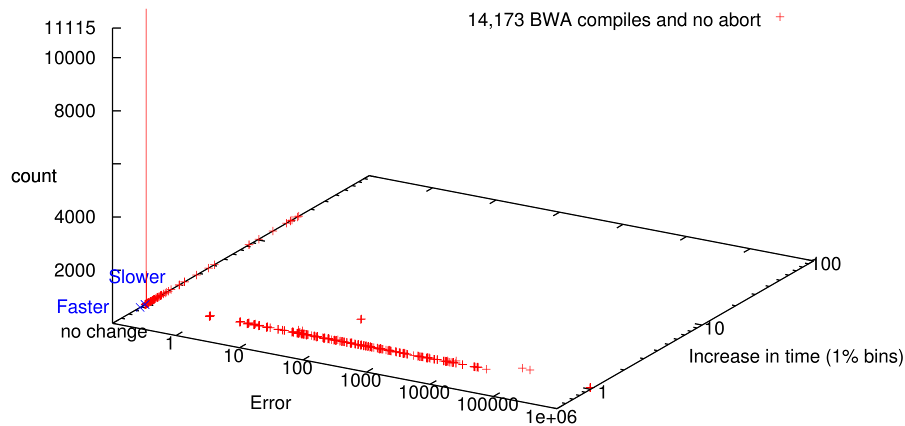

Figure 1: Impact of all (61 775) possible executable changes on BWA [18] [17, sec. 2.1] Of the 23% which compile and run normally 89% produce the same answer as the original code ("no change"). Indeed 3 of them are faster (×). From [17]

Genetic improvement (GI) [1] is the use of search heuristics, typically genetic programming (GP) [2], on existing software to find one or more better programs. For example, better could be having fewer bugs [3], a smaller source code [4], going faster [5, 6], using less resources [7, 8], having more features [9, 10] or simply being different. We deal with mutating source code, however GI has also been successfully applied directly to machine [11, 12] and Java byte code [13, 14].

Figure 1 show the impact of single mutations on a real C++ program. Whilst many changes do damage the source code, the plot is dominated by a large spike at the origin, showing many mutants do *not* damage the program. Schulte [11] and other have shown that contrary to popular assumptions such software resilience is wide spread. In mutation analysis [15] these are known as test equivalent mutants. Although equivalent mutants [16] are well known in software engineering, software engineers cling to the notion that software is a precious fragile thing, even though accepting that many random changes do not destroy it.

Figure 1 and similar graphs (e.g. [17]) are a start to trying to visualise the GI fitness landscape. They only show the impact of one change at a time. In a more academic example (comparison changes to the triangle program [15, Fig. 3]) we looked at all second, third and fourth order changes [Tab. 3][15]. This gave the alarming picture that although the number of equivalent mutants grows rapidly as the new code becomes more distant from the original the total number of possible changes grows even more rapidly. Thus the chance of finding a large change with no effect at *random* falls rapidly. However Figure 2 shows if a set of tests fail to detect a single change they are more likely to fail to detect both two and three simultaneous changes. I.e., in a fitness landscape, if a one step change passes the test suite then two step changes will also tend to pass it. Such non-uniform behaviour suggests that perhaps starting from a passing mutation will make it easier to find a passing two step mutation. However experiments based on the triangle program may have limited generality.

We wish to debunk the myth that any random change will destroy human-written programs. Whilst a random change might be bad, there is increasing evidence that if you are prepared to try multiple times, you can quickly find test equivalent mutants. However we have only a limited map of multiple changes which are needed to analyse the GI fitness landscape.

References

[1] Langdon, W.B.: Genetically improved software. In Gandomi, A.H., et al., eds.: Handbook of Genetic Programming Applications. Springer (2015) 181–220

[2] Poli, R., et al.: A field guide to genetic programming. Published via `http://lulu.com` and

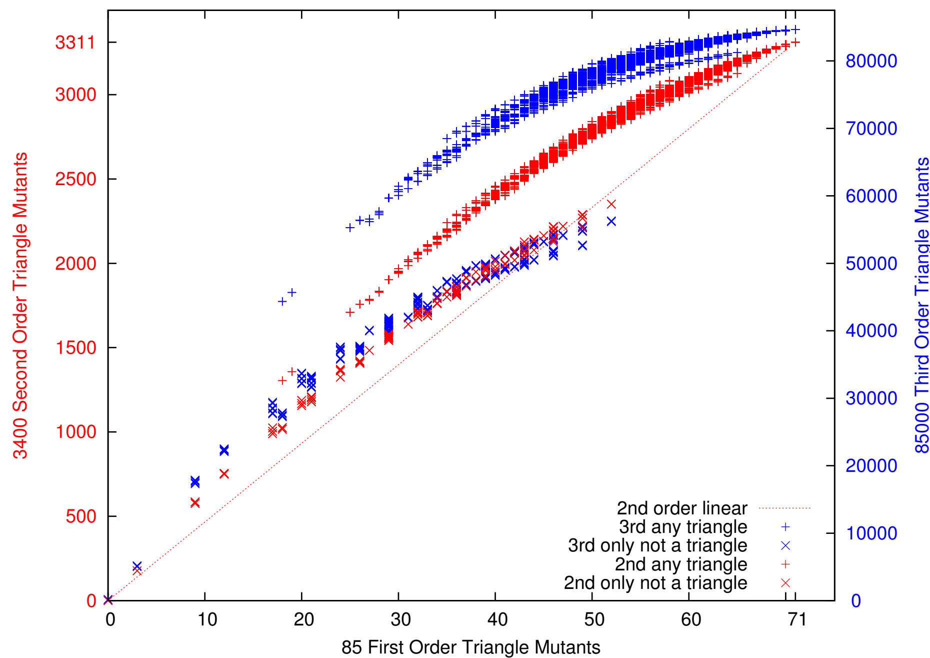

Figure 2: 16 383 test suites were generated by selectively including parts of the triangle test suite. Test suites that kill more first order mutants tend to kill more second order mutants (red) and third order mutants (blue). There is a near linear relationship, but most test suites are proportionately more effective against higher order triangle mutants. However test suites where all tests have an expected output of "not a triangle" (×) are closer to the proportionate response. From [15]

freely available at `http://www.gp-field-guide.org.uk` (2008) (With contributions by J. R. Koza).

[3] Le Goues, C., et al.: Current challenges in automatic software repair. Software Quality Journal **21** (2013) 421–443

[4] Landsborough, J., et al.: Removing the kitchen sink from software. In Langdon, W.B., et al., eds.: Genetic Improvement 2015 Workshop, ACM 833–838

[5] Langdon, W.B., Harman, M.: Optimising existing software with genetic programming. IEEE Transactions on Evolutionary Computation **19**(1) (2015) 118–135

[6] Langdon, W.B., Harman, M.: Grow and graft a better CUDA pknotsRG for RNA pseudoknot free energy calculation. In Genetic Improvement 2015 Workshop, 805–810

[7] Bruce, B.R., et al.: Reducing energy consumption using genetic improvement. In Silva, S., et al., eds.: GECCO '15: Genetic and Evolutionary Computation Conference, Madrid, Spain, ACM, ACM 1327–1334

[8] Wu, Fan, et al.: Deep parameter optimisation. In Silva, S., et al., eds.: GECCO '15: Proceedings of the 2015 on Genetic and Evolutionary Computation Conference, Madrid, ACM (2015) 1375–1382

[9] Harman, M., et al.: Babel pidgin: SBSE can grow and graft entirely new functionality into a real world system. In Le Goues, C., Yoo, S., eds.: SSBSE 2014. LNCS 8636, 247–252 Winner SSBSE 2014 Challange Track.

[10] Marginean, A., et al.: Automated transplantation of call graph and layout features into Kate. In Labiche, Y., Barros, M., eds.: SSBSE 2015. LNCS 9275, 262–268

[11] Schulte, E., et al.: Software mutational robustness. Genetic Programming and Evolvable Machines **15**(3) (2014) 281–312

```
function notme($ip) {
  return $ip!="93.4.203.5"     &&
         $ip!="93.4.201.238"   &&
         $ip!="93.4.201.15"    &&
         $ip!="93.4.201.196"   &&
         $ip!="93.4.201.125"   &&
         $ip!="93.4.201.43"    &&
         $ip!="93.4.203.196"   &&
         $ip!="93.4.201.38"    &&
//
//probably me?
         $ip!="215.88.203.93"  &&
         $ip!="215.88.151.5"   &&

         $ip!="93.4.201.5"     &&
         $ip!="93.4.151.5"     &&
         $ip!="109.142.203.5"  &&
         $ip!="109.142.54.196" &&
         $ip!="109.142.203.93" &&


         $ip!="86.162.4.128"   &&
         $ip!="86.162.4.5"     &&
         $ip!="86.162.212.5";
}
```

```
function me($ip,$year) {
  if(     $ip=="93.4.203.5"     ||
          $ip=="93.4.201.238"   ||
          $ip=="93.4.201.15"    ||
          $ip=="93.4.201.196"   ||
          $ip=="93.4.201.125"   ||
          $ip=="93.4.201.43"    ||
          $ip=="93.4.203.196"   ||
          $ip=="93.4.201.38" )
    return during($year,2004,2006);
//probably me?
  if(     $ip=="215.88.203.93"  ||
          $ip=="215.88.151.5" )
    return during($year,2007,2007);
  if(     $ip=="93.4.201.5"     ||
          $ip=="93.4.151.5"   )
    return during($year,2006,2008);
  if(     $ip=="109.142.203.5"  ||
          $ip=="109.142.54.196" ||
          $ip=="109.142.203.93"   )
    return during($year,2008,2010);
  if(     $ip=="86.162.4.128"   )
    return during($year,2010,2015);
  if(     $ip=="86.162.4.5"     ||
          $ip=="86.162.212.5"  )
    return during($year,2010,2100);
    return false;
}
function during($year,$start,$end) {
  if($year<2000 || $year>2100) {
    echo "during($year,$start,$end) bad year"
  return ($year>=$start && $year<=$end);
}
```

Figure 3: Change designed to restrict exclusion of my use of the GP bibliography [19] from statistics according to when I was using each computer. Exclusion was originally based only on computers' internet address (ip). Left: fragment of original PHP code. In calling code `if(!notme($parts[1]))` replaced by `if(me($parts[1],$year))`. Right new code. Coded by hand. Could artificial intelligence have made this change? Anonymised training data available via http://www.cs.ucl.ac.uk/staff/W.Langdon/gggp/notme

[12] Schulte, E., et al.: Repairing COTS router firmware without access to source code or test suites: A case study in evolutionary software repair. In Genetic Improvement 2015 Workshop, 847–854 Best Paper.

[13] Orlov, M., Sipper, M.: Flight of the FINCH through the Java wilderness. IEEE Transactions on Evolutionary Computation **15**(2) (2011) 166–182

[14] Yeboah-Antwi, K., Baudry, B.: Embedding adaptivity in software systems using the ECSELR framework. In Langdon, W.B., et al., eds.: Genetic Improvement 2015 Workshop, Madrid, ACM (2015) 839–844

[15] Langdon, W.B., et al.: Efficient multi-objective higher order mutation testing with genetic programming. Journal of Systems and Software **83**(12) (2010) 2416–2430

[16] Yao, Xiangjuan, et al.: A study of equivalent and stubborn mutation operators using human analysis of equivalence. In: ICSE 2014, 919–930

[17] Langdon, W.B., Petke, J.: Software is not fragile. In Bourgine, P., Collet, P., eds.: Complex Systems Digital Campus E-conference, CS-DC'15. Proceedings in Complexity, Springer (2015) Paper ID: 356 Invited talk, Forthcoming.

[18] Li, Heng, Durbin, Richard: Fast and accurate long-read alignment with Burrows-Wheeler transform. Bioinformatics **26**(5) (2010) 589–595

[19] Langdon, W.B., Gustafson, S.M.: Genetic programming and evolvable machines: ten years of reviews. **11**(3/4) (2010) 321–338