Test Generation with Genetic Algorithm methodologies and Template Scripting Test Language (TSLT)

Final Report for CS569: Static Analysis/Model Checking, Spring 2016

Nicholas Nelson

June 6, 2016

1 Strategies

My approach toward this project has significantly changed direction and focus throughout the course of this term. I had originally hoped to take advantage of the abstractions provided by the Template Scripting Test Language (TSTL) tool in order to implement Adaptive Random Testing (ART) through a Random String Generation concept originally. As indicated in my original project proposal, I had hoped to integrate the *Random Border Centroidal Voronoi Tessellations (RBCVT)* [2] algorithm with TSTL.

For Milestone 2, I re-evaluated my approach and switched from the Adaptive Random Testing (ART) family of algorithms to a 2-Phase Coverage Annealing Random Testing approach which built upon both a random testing phase and a mutation phase. This solution was predicated upon the idea of increasing coverage on poorly covered statements by focusing the mutations upon actions that touched upon those lines. This solution had the effect of smoothing the differences in coverage across statements, but at the expensive of diversity in tests and a reduction in effective fault detection.

For the Final submission, I have radically redesigned my test generation algorithm. Simplifying the internal structure by removing the multi-phased infrastructure and class definitions, I have opted to implemented the generalized single-objective optimization with genetic algorithm using the DEAP library [1]. The DEAP library has been extensively researched and developed by Félix-Antoine Fortin et al. at Université Laval - Québec from 2012 to 2014. The machine learning focus of this team has provided a comprehensive library for the implementation of genetic algorithms, evolution strategies, multi-objective optimizations, co-evolution, and parallelizable frameworks. I believe the use of DEAP provides a robust solution for test generation using TSTL.

2 Methodology

By adhering as closely as possible to the "One Max" problem and solution provided Félix-Antoine Fortin et al. on their website, I have developed a version that uses TSTL's awareness of enabled actions to generate populations of valid test sequence sets which can be evaluated using a fitness function that is based upon the branch coverage for each test sequence set. When ranking across multiple test sequence sets, my algorithm prefers the best coverage over any other metric. After developing a population of such tests, I then perform crossovers and mutations upon selections of tests within the population based upon random probabilities of occurences. This ensures a competition between tests through the genetic algorithm will eventually result in the highest available coverage from within that set.

Since the initial test sets are generated with an element of random chance to them, they can and do produce test sets with low overall coverage (even after several generations of crossover and mutation). To combat this stale data, I iterate through additional sets of newly generated populations; culling only the best possible tests from the final versions of each of the generations and populations.

Although slow at first, my algorithm appears to reach peak coverage within a reasonable amount of time and with a decent diversity that allows for higher chances of encountering faults within the Software Under Test (SUT). Further work could be done to seed additional sets of tests into an already generated population instead of throwing each population out and generating an entirely new population, but the current performance is already promising.

3 Overall

My algorithm is the combination of the DEAP [1] genetic algorithm library with the Template Scripting Test Language (TSTL) in a manner that allows iterative generation of populations, improvement upon those populations through crossover, mutation, and genetic algorithm cycles, and targeted coverage evaluations as the fitness metric.

Brief testing of the algorithm has shown that it can locate the combination fault within the sample AVLTree implementation, but is not guaranteed to do so on every run. This is likely based upon whether there is enough alotted time provided for the algorithm to run, and whether starting populations were sufficiently diverse to allow for beneficial genetic mutations and mating.

References

[1] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, July 2012.

[2] Ali Shahbazi, Andrew F. Tappenden, and James Miller. Centroidal voronoi tes-

sellations – a new approach to random testing. *IEEE Transactions on Software Engineering*, 39(2):163–183, February 2013.