# Adaptive Random Testing through Random String Generation in TSTL

**Project Proposal for CS569: Static Analysis/Model Checking, Spring 2016**

Nicholas Nelson

April 19, 2016

## 1 Background

Black-box testing methods, such as Random Testing (RT) and Adaptive Random Testing (ART) generate test cases without considering the program's source code [2]. Therefore, these methods are independent from the language of the source code. As a result, black-box testing methods are very general; all that is needed is the structure of the inputs and the outputs of the program under test.

RT is interesting since it has a low computational cost and is easy to implement. However, RT is not very effective at fault detection. According to various empirical studies, e.g., [1] [6] [12] [3] [8], faults usually occur in continuous regions within the input domain. This means that faults are often clustered in the input space [5]. Accordingly, a diverse set of test cases that has a better coverage of the input domain has a greater chance of detecting a fault. As a result, RT's failure detection performance can be improved if test cases are distributed more diversely in the input space.

ART approaches [4, 7, 11] were developed to enhance the performance of RT. ART approaches generate more effective test cases by producing diverse, but still random, test cases across the input domain. Therefore, the probability of fault detection is improved [11].

## 2 Proposal Description

The recent work of Shahbazi and Miller has seen the introduction of a new approach, namely *Random Border Centroidal Voronoi Tessellations (RBCVT)* [10]. However, RBCVT is limited to generating numerical test cases. Beside numerical inputs, many programs accept strings as their input. To be more precise, a string input in this context, and all the strings that are generated in the evaluations refer to unstructured ASCII strings.

In a follow-up paper to their work on RBCVT, Shahbazi and Miller [9] turned their focus toward developing methods to automatically generating string test cases. They

focus upon generating effective sets of tests cases where each test case is a string. And, since it is believed that a diverse set of test cases is more likely to produce more effective test cases [1] [6] [12] [3] [8], they have defined a fitness function that measures the diversity of a test set. This allows an optimization technique to be employed to generate test cases based upon the fitness function. To construct a fitness function to measure the diversity, they utilized distance functions between strings.

Since Shahbazi and Miller have already done the work of evaluating several different string distance functions in terms of performance metrics based upon the effectiveness of the generated test cases and runtime, I would like to reimplement their best performing algorithm within TSTL. To do so, I will treat the combination of the actions and parameter inputs of the software under test (SUT) as strings and using that set, apply the overall best string distance functions using Shahbazi and Miller's *Multi-Objective Genetic Algorithm (MOGA)* [9].

The results from Shahbazi and Miller's work [9] show that the runtime order complexity of *MOGA (NSGA-II)* is $O_{MOGA} = N^2 \times (O_{FL} + O_{FD})$, where $N$ is the population size, $FL$ is the *Length-Control Fitness Function* and $FD$ is the *Diversity-based Fitness Function*. Combined with the real-world performance evaluation tests that they conducted to evaluate their method, which determined that using a multi-objective optimization technique would result in superior test case production, I hope to at least match the performance of RT within TSTL.

## References


[1] Paul E. Ammann and John C. Knight. Data diversity: An approach to software fault tolerance. *IEEE Transactions on Computing*, 37(4):418–425, April 1988.

[2] Saswat Anand, Edmund K. Burke, Tsong Yueh Chen, John Clark, Myra B. Cohen, Wolfgang Grieskamp, Mark Harman, Mary Jean Harrold, and Phil Mcminn. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*, 86(8):1978–2001, August 2013.

[3] P.G. Bishop, Coborn House, and E Da. The variation of software survival time for different operational input profiles (or why you can wait a long time for a big bug to fail). In *Proceedings of FTCS-23*, pages 98–107, 1993.

[4] T.Y. Chen, F.C. Kuo, H. Liu, and W.E. Wong. Code coverage of adaptive random testing. *IEEE Transactions on Reliability*, 62(1):226–237, March 2013.

[5] T.Y. Chen, T.H. Tse, and Y.T. Yu. Proportional sampling strategy: a compendium and some insights. *Journal of Systems and Software*, 58(1):65–81, 2001.

[6] George B. Finelli. Nasa software failure characterization experiments. *Reliability Engineering & System Safety*, 32(1):155–169, 1991.

[7] J. Lv, H. Hu, K. Y. Cai, and T. Y. Chen. Adaptive and random partition software testing. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(12):1649–1664, December 2014.

[8] Christoph Schneckenburger and Johannes Mayer. Towards the determination of typical failure patterns. In *Fourth International Workshop on Software Quality Assurance: In Conjunction with the 6th ESEC/FSE Joint Meeting*, SOQUA '07, pages 90–93, New York, NY, USA, 2007. ACM.

[9] Ali Shahbazi and James Miller. Black-box string test case generation through a multi-objective optimization. *IEEE Transactions on Software Engineering*, 42(4):361–378, April 2016.

[10] Ali Shahbazi, Andrew F. Tappenden, and James Miller. Centroidal voronoi tessellations – a new approach to random testing. *IEEE Transactions on Software Engineering*, 39(2):163–183, February 2013.

[11] A. F. Tappenden and J. Miller. A novel evolutionary approach for adaptive random testing. *IEEE Transactions on Reliability*, 58(4):619–633, December 2009.

[12] L.J. White and E.I. Cohen. A domain strategy for computer program testing. *IEEE Transactions on Software Engineering*, 6(3):247–257, 1980.