Chapter 3
The Application of Genetic Algorithms to the Evaluation of Software Reliability

Angel Fernando Kuri-Morales
Instituto Tecnológico Autónomo de México, México

ABSTRACT

The evaluation of software reliability depends on a) The definition of an adequate measure of correctness and b) A practical tool that allows such measurement. Once the proper metric has been defined it is needed to estimate whether a given software system reaches its optimum value or how far away this software is from it. Typically, the choice of a given metric is limited by the ability to optimize it: mathematical considerations traditionally curtail such choice. However, modern optimization techniques (such as Genetic Algorithms [GAs]) do not exhibit the limitations of classical methods and, therefore, do not limit such choice. In this work the authors describe GAs, the typical limitations for measurement of software reliability (MSR) and the way GAs may help to overcome them.

1. INTRODUCTION

The IEEE defines reliability as “The ability of a system or component to perform its required functions under stated conditions for a specified period of time.” Rosenberg, L., Hammer, T., Shaw, J. (1998) wrote that to most project and software development managers, reliability is equated to correctness. That is, they look to testing and, thereafter, finding and fixing the “bugs” thus yielding the “correct” software. Typically we are not satisfied with merely “correct” software. Rather we also seek software which also displays a certain amount of quality. Intuitively, one of the basic elements of software’s quality is its correctness, i.e. a program’s characteristic of outputting adequately (correctly) for all possible inputs. On first look it would seem that correctness is the least one could ask of reliable software. Unfortunately, one of the basic theoretical consequences of the analysis by Turing (1936) is that there is no algorithm (or program) which will, in all cases, be able to determine whether another algorithm (or program) will behave correctly. In other words, correctness is incomputable. Therefore,
the assessment of software’s correctness turns out to be formally undecidable. In practice, therefore, one has to acquiesce with the fact that practical “correctness” results from conscientious testing, and practically finding and (hopefully) fixing bugs.

Since finding and fixing bugs discovered in testing is necessary to assure reliability, a methodology leading to the development of a robust, high quality product through all of the stages of the software lifecycle is most desirable. That is, the reliability of the delivered code is related to the quality of all of the processes and products of software development; the requirements documentation, the code, test plans, and testing.

Software reliability is not a well defined concept, but there are important efforts striving to identify and apply metrics to software products that promote and assess reliability. Reliability is a by-product of quality, and software quality can be measured. Quality metrics assist in the evaluation of software reliability.

ISO 9126 (1991) defines six quality characteristics, one of which is reliability. Gillies (1992) indicates that “A software reliability management program requires the establishment of a balanced set of user quality objectives, and identification of intermediate quality objectives that will assist in achieving the user quality objectives.” Since reliability is an attribute of quality, it can be concluded that software reliability depends on high quality software.

1.1 Optimizing from Quality Criteria

If we are able to objectively define the elements on which software’s quality depends then we will be able to define an objective measure of it. Furthermore, we will also be able to compare different software systems purportedly solving the same set of problems and establish a solid figure of relative merit. Once having defined the most adequate metrics the assessment of software quality becomes one of optimization: namely, the maximization of the figure of merit. The problem of how to achieve efficient optimization has led to the development of computer intensive methods which differ from the classical ones in various respects. The classical optimization techniques are useful in finding the optimum solution or unconstrained maxima or minima of continuous and differentiable functions. These are analytical methods and make use of differential calculus in locating the optimum solution. The classical methods have limited scope in practical applications as some of them involve objective functions which are not continuous and/or differentiable. These methods typically assume that the function is twice differentiable with respect to the design variables and that the derivatives are continuous. The quality criteria related to software quality which are to be optimized do not satisfy the above requirements. In fact, they do not usually even correspond to a closed form function. Therefore a new approach to the optimization is needed, one which is not dependent on the mathematical properties just mentioned. Several such optimization techniques have been advanced and analyzed during the past years. They were inspired by natural evolution and are collectively called “Evolutionary Computation” (EC).

1.2 Evolutionary Computation

Throughout the years since their inception there have been several attempts to mimic (albeit partially) the process of biological evolution. The primary motivation arises from the fact that living beings are seen as advanced solutions to the problem of adaptation to a dynamic but (relatively) stable environment; they are the by-product of optimal adaptation to dynamic habitats. Some of the evolutionary methods are more mathematically demanding while others are more intuitive. In a work of this kind it is impossible to even skim over all of them. It is enough, in this context, to say that examples of EC are Evolution Strategies, Evolutionary Programming, Genetic Programming, Learning Classifier Systems and Genetic Algorithms (GAs). Among these the most popular
Related Content

Evaluation of Genetic Algorithm as Learning System in Rigid Space Interpretation
[www.igi-global.com/chapter/evaluation-of-genetic-algorithm-as-learning-system-in-rigid-space-interpretation/82702?camid=4v1a](www.igi-global.com/chapter/evaluation-of-genetic-algorithm-as-learning-system-in-rigid-space-interpretation/82702?camid=4v1a)

BDD-Based Synthesis of Reversible Logic
[www.igi-global.com/article/bdd-based-synthesis-reversible-logic/51676?camid=4v1a](www.igi-global.com/article/bdd-based-synthesis-reversible-logic/51676?camid=4v1a)

Relevance of Mesh Dimension Optimization, Geometry Simplification and Discretization Accuracy in the Study of Mechanical Behaviour of Bare Metal Stents
[www.igi-global.com/chapter/relevance-mesh-dimension-optimization-geometry/65163?camid=4v1a](www.igi-global.com/chapter/relevance-mesh-dimension-optimization-geometry/65163?camid=4v1a)

Subspace Discovery for Disease Management: A Case Study in Metabolic Syndrome
[www.igi-global.com/article/subspace-discovery-disease-management/53720?camid=4v1a](www.igi-global.com/article/subspace-discovery-disease-management/53720?camid=4v1a)