Volume 3, Issue 6, June 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Genetic Algorithm Approach for Automatic Patch Prediction

P. Maragathavalli , S. Kanmani
Department of Information Technology
Pondicherry Engineering College
Puducherry-14, India

Abstract—Software system should be reliable and available failing which huge losses may incur. To achieve these objectives a thorough testing is required. In recent years, there has been a dramatic rise in the number of languages used in mainstream projects. In particular, languages which run on the JVM or Common Language Runtime (CLR) have become quite popular in creating a global development framework. Naturally, such languages prefer to inter-operate with other languages built on these core platforms, particularly Java and C#. The problem is that such efforts are crippled by one fundamental limitation: circular dependencies-a relation between two or more modules which either directly or indirectly depend on each other to function properly. This paper presents the study of patch prediction using a common testing software to minimize cost of testing and optimization of software testing techniques by using Genetic Algorithms (GAs) and Sequential Quadratic Programming Algorithm(SQPA) to provide features like memory management, security, garbage collection, thread management and exception handling.

Keywords—Optimization techniques, Genetic algorithm, SQP algorithm, Patch prediction.

I. Introduction

In the thirty past years, Genetic algorithms have been successfully applied to a wide range of problems such as Natural Systems Modeling (e.g. Artificial Life environments, immune system modeling [2] [3], Machine Learning systems, and optimization). The co-evolutionary approach, Genetic Programming (GP) has been applied for automatic software repair in most of the domains [5] [6]. They use *delta debugging* to efficiently compute a one minimal subset of changes from the primary repair. Delta debugging is conceptually similar to binary search, but it returns a set instead of a single number. GAs handle a population of individuals (chromosomes) often modeled by vector of binary genes. Each one encodes a potential solution to the problem and so-called fitness value, which is directly correlated to how good it is to solve the problem. In general, the basic approaches are to test software consists of using formal specifications to design an application. This approach is very strict but unfortunately not often used because the breadth of formal specification methods does not encompass all the functionality needed in today's complex applications.

The second approach consists of doing test as part of the traditional engineering models (e.g. waterfall, spiral, prototyping) that have a specific phase for testing generally occurring after the application has been implemented. The modifications to these traditional models have being incorporating testing in every phase of the software development with methodologies such as extreme programming [4] used in the implementation of Windows XP. Despite all the claims, the truth here is that current approaches are insufficient to test software appropriately, thus causing the status of the field, which clearly seems to be losing the battle of providing users with reliable software. It has been noticed that complete reliability is hard to achieve in empirical approaches to complete testing is impossible.

This paper is organized into three parts: part I describes the functionality of GA and SQPA, part II presents the usage of these algorithms in detection of errors to the alternatives of existing error detection techniques, part III discusses the implementation of GAs using .net in Microsoft Visual Studio for error detection followed by the result analysis and conclusion.

II. Optimization Using Genetic Algorithms

Genetic algorithm is a search technique used to find exact or approximate solutions to optimization and search problems. GAs represents a class of adaptive search techniques & procedures based on the processes of natural genetics & Darwin's principal of the survival of the fittest. There is a randomized exchange of structured information among a population of artificial chromosomes. A problem is defined as maximization of a function of the kind $f(x_1, x_2 \ldots x_m)$ where $(x_1, x_2 \ldots x_m)$ are variables which have to be adjusted towards a global optimum. Three basic operators are used in GA, are selection, crossover & mutation. Crossover performs recombination of different solutions to ensure that the genetic information of a child life is made up of the genes from each parent. GAs differentiated from other conventional techniques due to: (i) GA a representation for the sample population must be derived; (ii) GAs manipulate directly the encoded representation of variables, rather than manipulation of the variables themselves; (iii) GAs use stochastic rather than deterministic operators; (iv) GAs search blindly by sampling & ignoring all information except the outcome of the sample; (v) GAs search from a population of points rather than from a single point; thus reducing the probability of being stuck at a local optimum, which make them suitable for parallel processing. An attempt of evolutionary algorithm GA can be taken for finding patches in the software.

In paper [1], presents a new procedure for the miniaturization of rectangular microstrip patches based on genetic algorithms. The procedure starts with a typical rectangular patch, which is divided into an adequate number of rectangular cells. Some cells are then removed to provide reduction in the patch resonance frequency. It is a complete and versatile GA code with advanced options like niching, uniform or simple crossover, jump and creep mutation, elitism, or the possibility of choosing one or two children per couple of parents. The genetic algorithm concepts for patch finding are learnt from this paper. In paper [2], many of the human-competitive results were produced using runs of genetic algorithms and genetic programming that employed a developmental process. They explained the way of using genetic programming in 'machine intelligence' to make the systems understandable one; so called *intelligent machines.* The survey has been made with genetic programming used in game playing, finite algebras, photonic systems, image recognition, optical lens systems, cellular automata rules, bioinformatics, sorting networks, robotics, assembly code generation, software repair, scheduling, communication protocols, symbolic regression, reverse engineering, and empirical model discovery. From this, automatic software repair techniques are useful.

Paper [3] proposes a novel patch generation approach, Pattern-based Automatic program Repair (PAR), using fix patterns learned from existing human-written patches. They manually inspected more than 60,000 human-written patches and found there are several common fix patterns. They approach leverages these fix patterns to generate program patches automatically. The generation and evaluation of patches for different data sets are learnt from this paper. Another evolutionary approach, Genetic Programming is used commonly in automated software repair [5] [6] [7]. In paper [9], genetic algorithm is used to find optimization sequences that generate small object codes. The solutions generated by this algorithm are compared to solutions found using a fixed optimization sequence and solutions found by testing random optimization sequences. Based on the results found by the genetic algorithm, a new fixed sequence is developed to reduce code size. They use a compiler framework with C and FORTRAN to compare the results of code sizes with no optimization, fixed optimization sequences, and optimizing using sequences found by the genetic algorithm. The concept of compiler code optimization using GA is learnt from this paper [9].

III. Design Of Common Language Runtime Compiler

The Common Language Runtime (CLR) is a core component of Microsoft's .NET initiative. The CLR is the part of the .NET Framework that executes and manages any .NET Application. It is Microsoft's implementation of the Common Language Infrastructure (CLI) standard, which defines an execution environment for program code. In the CLR, code is expressed in a form of bytecode called the Common Intermediate Language. Developers using the CLR write code in a language such as C# or VB.NET. At compile time, a .NET compiler converts such code into CIL code. At runtime, the CLR's just-in-time compiler converts the CIL code into code native to the operating system. Alternatively, the CIL code can be compiled to native code in a separate step prior to runtime. The data flow among modules in CLR is shown in Fig.1.

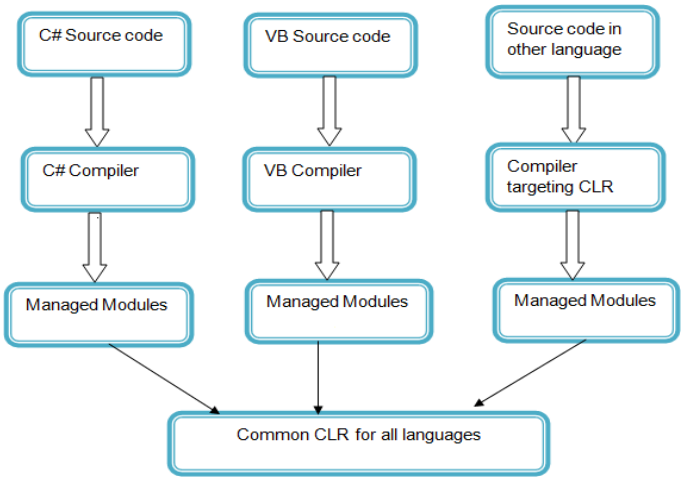


Fig. 1 Data flow diagram

The following are the services provided by the common language runtime compiler:

1) Memory Management: Automatic memory management of applications is done using a garbage collection mechanism. This guarantees that objects that have no reference in your application will be freed, and you as the developer will not worry about releasing memory for objects. This guarantees faster performance and, at the same time, gives you the time to focus on the development issues of your applications.
2) Security: Guarantees that your .NET Applications will run in a safe context by using a mechanism called Code-Access Security. There is another security mechanism called Role-Based Security which controls the access to system resources. Using .NET, security is managed by CLR.
3) Exception Handling: Handling of application errors is performed by throwing an exception to the CLR.
4) Thread Management: Threads are commonly used for processing; the multiple threads management is taken care by CLR itself.
5) Garbage Collection: Free blocks are combined together for managing memory spaces within a CLR.

IV. Implementation

The implementation of automatically finding patches using genetic algorithm has been dealt in Microsoft Visual Studio. Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It can be used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services in both native code together with managed code for all platforms supported by Microsoft Windows .NET Framework, and .NET Compact Framework. Visual Studio supports different programming languages by means of language services, which allow the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C/C++ (via Visual C++), VB.NET (via Visual Basic .NET), C# (via Visual C#), and F# (as of Visual Studio 2010). Support for other languages such as M, Python, and Ruby among others is available via language services installed separately. Individual language-specific versions of Visual Studio also exist which provide more limited language services to the user: Microsoft Visual Basic, Visual J#, Visual C#, and Visual C++.

The .NET Framework is a software framework for Microsoft Windows operating systems. It includes a large library, and it supports several programming languages which allow language interoperability (each language can use code written in other languages). The .NET library is available to all the programming languages that .NET supports. Our implementation uses .net as a platform to develop common testing software for all languages supported by .net.

The implementation is organized into five processes namely Authentication, Optimization, Estimation, Correction, and Performance evaluation; these things are explained below:

1) *Authentication*: User authentication is required to check the .net code for errors and their correction. The input of software is the .net code to be tested. C# J# or any language under .net can be the input. The common language runtime makes it easy to design components and applications whose objects interact across languages. Objects written in different languages can communicate with each other, and their behaviors can be combined.
2) *Optimization:* The optimization model is implemented in a prototype tool, which receives the input folder consisting of all files related to the project as input and spits the required executable file from the population of all files using Genetic Algorithm. The approach is to improve the accuracy of the prediction abilities. The program chosen consists of almost 10,000 lines of Dot net code.
3) *Error Estimation*: In this module we compute an estimate of the defects in an ongoing testing process. This could be used to improve the plan for developing the test cases. System testing is one of the last phases (if not the last), the time to release can be better assessed; the estimated remaining defects can be used to predict the required level of customer support. The implemented tool uses the SQP algorithm for the exact solution and a genetic algorithm for the heuristic solution, useful for solving multiple application problems.
4) *Error Correction*: Error correction techniques are used to improve the estimates and, consequently, reduce the convergence time. After error has been estimated user is provided with the specified line number of the error along with a tree view of the code for ease of correction. It is used to correct the current estimate; the corrected value is saved in the code where the source code located in a system.
   The output screen looks like as shown in Fig.2:

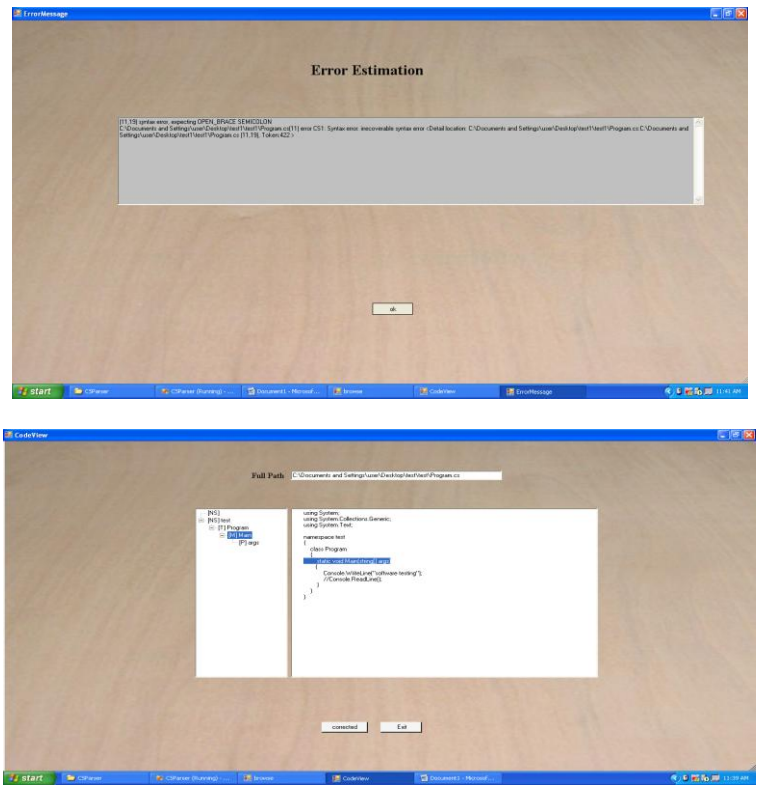


Fig. 2 Error estimation and correction

5) Performance Evaluation: Fitness function calculates the number of lines of dead code eliminated. Higher the fitness means that higher the level of optimization performed. Software was tested according to the predicted times and the actual achieved reliability was compared with the predicted reliability. We also evaluated the effect of the error in the parameter estimations on the predictive accuracy. During the testing process of a critical application, part of testing resources could be reserved to tune the application performance in order to fulfill performance requirements. This is especially true for real-time systems.

V. Result Analysis

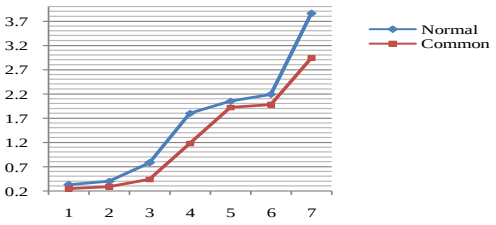
The software is tested with different types of codes injected with different errors. The error detection time of common testing software is compared with normal .net compiler for performance analysis. Codes are tested for different errors displaying the error definition with the corresponding token number, line number and column number. The compile time required for Normal, Common compilers and Code optimization results ARE TABULATED IN TABLE 1.

TABLE I

RESULTS OF JAVA PROGRAMS IN REDUCED COMPILE TIME AND LINES OF CODE

S. No.	Sample programs	Lines of Code			Compile time (in secs)	
		Initial	Without using GA	With using GA	Normal compiler	Common compiler
1.	Travelling salesman	55	51	40	0.33	0.25
2.	Website	80	75	60	0.40	0.29
3.	Estate	180	170	136	0.78	0.44
4.	Elevator	160	152	140	1.80	1.18
5.	N-queens problem	240	228	210	2.05	1.92
6.	Report generation	300	284	252	2.19	1.98
7.	Chess playing	310	293	278	3.86	2.94

Compile time required for common compiler is reduced significantly when compared to the normal compiler. Initially, simple programs are tested using genetic algorithm with LOC of 50-100 and then with 150-310. For small programs both the code and compile time are less, so the difference in reduction is also less; whereas in large programs the source code as well as the compile time using common compiler, the reduction results are significant and shown in Fig. 3. For example, the *travelling salesman* problem compile time is reduced from 0.33 to 0.25s; but, in *report generation* and *chess playing* the lines of code is reduced to 252 & 278 and compile time difference is in secs; i.e., completed 1secs before by using common compiler.



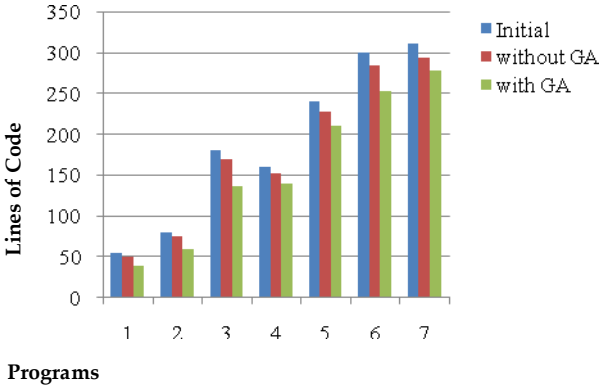


Fig. 3 Reduction in compile time and lines of code

VI. CONCLUSION

Genetic algorithms produce better results in optimization of many software engineering problems like resource allocation, inductive concept learning, scheduling, and game playing. In this paper, genetic algorithm is used for automatic error detection and correction with the help of newly developed compiler; Common compiler has been developed based on common language runtime in order to reduce the size of source code as well as testing time. The experimental results are satisfactory and reduction is achieved almost 89%. More languages can be added into the common compiler in addition to java and c#.

REFERENCES

[1] Ridhi Gupta, Sanjay Gurjar, and Ashish Kumar, "Using Genetic Algorithms Reduction of Rectangular Microstrip Patches," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 2, iss. 4, pp. 1643-1646, Apr. 2013.

[2] John R. Koza, "Human-competitive results produced by genetic programming," *Genetic Programming Evolvable Machines*, Springer, vol. 11, pp. 215- 284, May 2010.

[3] Dongsun Kim, Jaechang Nam, Jaewoo Song, and Sunghun Kim, "Automatic Patch Generation Learned from Human-Written Patches," *35th International Conference on Software Engineering*, San Francisco, CA, May 18-26, 2013.

[4] Kulvinder Singh and Rakesh Kumar, "Optimization of Functional Testing using Genetic Algorithms," *International Journal of Innovation, Management and Technology*, vol. 1, no. 1, Apr. 2010.

[5] Stephanie Forrest, ThanhVu Nguyen, Westley Weimer and Claire Le Goues, " A Genetic Programming Approach to Automated Software Repair," *Genetic and Evolutionary Computation Conference (GECCO)*, Canada, pp. 947-954, Jul. 8–12, 2009.

[6] Westley Weimer, Stephanie Forrest, Claire Le Goues, and ThanhVu Nguyen, "Automatic Program Repair with evolutionary computation," *Communications of the ACM*, vol. 53, no. 5, May 2010.

[7] A. Arcuri, and X. Yao, "A novel co-evolutionary approach to automatic software bug fixing," *IEEE Congress on Evolutionary Computation*, pp. 162–168, 2008.

[8] Roberto Pietrantuono, Stefano Russo, and Kishor S. Trivedi, "Software Reliability and Testing Time Allocation: An Architecture-Based Approach," *IEEE Trans. on Software Eng.*, vol. 36, no. 3, May/Jun. 2010.

[9] Keith D. Cooper, Philip J. Schielke, and Devika Subramanian, "Optimizing for Reduced Code Space using Genetic Algorithms," Department of Computer Science, Rice University, Houston, Texas, USA, 2009.

[10] K. D. Cooper, D. Subramanian, and L. Torczon, "Adaptive optimizing compilers for the 21st Century," *In Proceedings of the 2001 LACSI Symposium*, Los Alamos Computer Science Institute, Oct. 2001.

[11] A. Eiben and J. Smith, "Introduction to Evolutionary Computing," Springer, 2003.

[12] Andy P Nisbet, "GAPS: A compiler framework for genetic algorithm (GA) optimised parallelisation," *International Conference and Exhibition on High-Performance Computing and Networking*, HPCN Europe '98, 1998.

