AUTOMATIC SOFTWARE TESTING BY GENETIC ALGORITHM OPTIMIZATION, A CASE SUDY

JARMO T. ALANDER AND TIMO MANTERE

ABSTRACT. In this work we have studied the feasibility of program testing automation by using optimization via genetic algorithm. The main objective of the study is to find potentially problematic situations by maximizing the response times of a rather large real-time embedded system. The results show that the genetic optimization is able to outperform pure random testing. The identification of the problematic input helps the programmer to fix performance bottlenecks.

1. INTRODUCTION

Performance and reliability are among the most important quality criterion in the planning, realization and use of computer systems. The objective is to guarantee at least the minimum performance in any case. For example in some real-time systems absolute response time constraints are set. In general speed is an important competition factor in software markets. Unfortunately it is difficult to reach good performance in practice.

Requirements for precision and reliability of embedded software are usually high, e.g. a failure to meet a response time constraints might have severe consequences both in economics and human terms. It has been estimated that about two thirds of the total costs of product development projects of embedded systems consist of software development work, of which about half is testing costs [17]. Thus software testing will cause up to one third of the total product development costs. Testing is laborious, because it tend to increase exponentially with the code complexity. Manual testing is slow and thus expensive. Software grows and becomes more and more complicated bringing with it problems related to its performance.

It has been said that when testing software, one should examine only small sections at a time, otherwise it easily results that somewhere is a fault, but it is extremely difficult to locate. However in this work the goal was to test the whole software as one piece. In larger software the performance problems are usually emphasized. There can be millions of lines of source code produced by hundreds of programmers. Even if there was no performance problem in the code written by the individual programmer, such will easily appear in the integration phase. Testing is a vital part of software development and automation makes it faster, more reliable and cost efficient.

1.1. Research problem.

The object of this study is software testing automation; how to realize it using a combinatorial optimization method called genetic algorithm (GA). A real-time embedded system must usually react within a given response time. This can be seen as an optimization problem: find those i.e. problem cases, causing the longest response times.

The object of this study is a communication module of a rather large embedded software, consisting of about 11 Mb of source code written in C. We performed automatic black-box type stress testing, by using a GA, that optimizes input parameters, stimuli from the environment to

Key words and phrases. Software engineering, software testing, combinatorial optimisation, embedded systems, genetic algorithms, simulation.

the software. The objective was to find the extreme response times, in order to reveal bottlenecks, which must be corrected. A simulation environment was used mainly for cost reasons.

Related work. In the field of software testing GA has been applied mainly to the automatic test data generation [13, 14, 16, 18, 19, 20, 22, 25, 26, 27]. These studies have concentrated on the optimization of branch coverage; in other words finding test sets that covers all possible paths of the program. Especially reference [13] is interesting because in it several problems are dealt with both a GA and random method. For more references on GA in software testing see the bibliography [2]. A closely related application area of GAs is VLSI testing [3].

2. Genetic algorithms

The genetic algorithms have steadily reached growing popularity during the last twenty or so years as a general method to solve difficult optimization problems [2]. The method is based on the recombination of genes in the same way as it takes place in nature. This combined to selection leads to the growth of the average fitness of the population. GA is not perhaps the best possible method in a given task but it is robust and more or less suitable for most complex optimization tasks.

A GA forms a sort of artificial simple electric ecosystem, where digital beings (bit-strings, that represent parameter values) struggle for survival and reproducing possibilities. GA is a method especially suited for high power digital computing. It is at its best when facing problems that can't be solved with exact methods or at least within a given processing time. GA is often applicable when the number of parameters or their variations is considerable i.e. when combinatorial explosion prevents us from testing all possibilities. The reason for the growing popularity of genetic algorithms is probably the fact that GA does not set any *a priori* restrictions on the target function. It does not need to be smooth, meaning it does not need to be derivative or even be continuous and it can have several local optimum. The target function doesn't even need to be expressible in a mathematical form, if one can e.g. measure fitness somehow.

The study of genetic algorithms started in 1975, when John Holland developed the first GA at the University of Michigan [11]. He utilized evolutionism to number sequences that would live, reproduce and die like living organisms. He got the idea after having been convinced that living organisms can solve an optimization problem like adaptation to its natural environment better than even the most powerful supercomputers. In order to describe it, Holland borrowed terms from the glossary of genetics. However, all the concepts can be interpreted exactly and are easy to implement.

Benefits of GA include that it can easily be run on parallel processors and it can easily be adapted to different problems. It doesn't provide much programming and is usually relatively efficient in difficult problems. GAs belong to the so called soft computing methods, which include such methods as artificial neural nets and fuzzy logic.

3. Software testing

Software that has been compiled is syntactically correct, but it usually contains an unknown number of semantic errors. These errors are searched by code validation or by running the program using some testing method [12, 17, 24]. The amount of testing is not necessarily equivalent to the effectiveness of testing: a few carefully planned test cases can lead to better results than an intensive random experiment. However, one should not forget that even the best tests cannot save a badly designed program.

A comprehensive test would require the testing with at least every acceptable input. The test result depends not only on input but also on the state of the program. Thus input should be tested with all combinations of different states. Unfortunately comprehensive testing is impossible in practice. This does not mean that investment to testing would not be worthwhile, but that the functionality of the program should not be trusted too much.

3.1. **Automation of testing.** It has been estimated that software testing takes 30-50% of the working time of a software project [9]. In terms of manpower testing is expensive and susceptible to errors. Already partial testing automation will bring significant savings and improves program quality. Software testing automation can benefit from the following [7]:

- The testing can be prepared even before all the necessary tools are available; this was also partly the case in this study.
- It accelerates the testing essentially.
- The tests can be done at least partly unmanned.
- The repeatability helps to locate also infrequent error situations.
- It reduces the amount of routine work.
- Opportunity for remote testing.

There are also some drawbacks:

- The planning of tests and analysis of results may be laborious.
- Requires better trained testing tool developers and users.

Software can be so complex that one cannot easily deduce what response results from a given input. Thus determining the proper test data can be difficult. Software may be so non-deterministic that the response time is due to more the non-deterministic nature of the software and testing system, than the given input.

4. Embedded systems

Embedded system refers to a computer system that has been integrated into an electro-mechanical device. The electronics part of the embedded system controls electric and mechanical functions of the device. The processor of the system executes the so-called embedded software, which is to read impulses from the environment and to process responses to them. Usually the embedded software is real-time, this means that it has to respond within given time constraints.

4.1. **Real-time software.** The operation of any sequential program has a well defined beginning and end, while a real-time software is in an infinite loop constantly interacting with its environment. An impulse originating from the environment starts a series of events, which results in some sort of response. Real-time software is usually required to meet some response time constraints.

The operation of the real-time software is often presented with the help of the so-called state-machine model. In a simple state-machine only the state, during which the impulse occurs affects the response. The actual software is seldom in accordance with this simple model, however. The actual software can process several impulses overlapping causing also the responses to have tendcy to overlap, resulting in sometimes unexpected and unforeseeable interactions. The load variations can cause dramatic behavioral changes. Thus it is not usually easy to predict the behaviour of the system.

5. Experimental setup

The object embedded system has been designed to carry out many functions as a power distribution protection device (relay) control unit. It is responsible for the primary protection functions

of the device and auxiliary functions including controlling functions, measuring functions and surveillance procedures. The device contains actually several processing and interface units, which communicate with each other and other protection devices. The system have two separate serial ports, which support several communication protocols.

On the whole the system hosts a quite demanding software, which should operate fast and reliably in all circumstances. It was still under development during our project.

of

5.1. **Simulation environment.** The testing was done using a simulation environment, which was chosen originally because of cost reasons. Another obvious reason was that the hardware was not yet completed when the project started.

The simulation environment was ESIM, a program development tool for the simulation and testing of the embedded systems. It's a product of a Finnish company called Prosoft. ESIM is designed to be able to simulate any given embedded system, provided that the software is written in C or C++ language [23]. linked ESIM provides opportunities to monitor program execution. Furthermore it provides simulation functions for the interfaces (ports, buses, user interface), the operating system, registers, tasks, semaphores, etc.

whole

5.2. **Field buses.** The communication between the modules is done via field buses. The testing concentrated on the load testing of the message-processing unit. There were two main field buses in use, CAN (controller area network) and LON (local operating network).

The tested target was the communication unit, which operated under the main CPU. The object software was constructed in such a way that it uses a CAN (controller area network) bus for all data transfer between the main CPU and other electronic modules, including I/O-cards, key panel, and LCD-display. The same processing unit also handles LON bus messages. Communication interface, CAN and LON field buses are also the natural interface to the software. In our test CAN bus data transfer was simulated and tested, while software behaviour was monitored. The main objective was to study if lower priority LON message processing has some effect on the processing of higher priority CAN messages.

CAN is a serial bus which has been developed for the data transfer of advanced real-time and decentralized control system devices, in which for example the sensors and regulating units communicate directly with each other without the help of any intervening control unit. The CAN bus is a so-called multi-master, in other words several nodes can access the channel simultaneously. The CAN protocol is relatively simple which makes the programming of applications fairly easy. The high reliability of the data transfer has also been stated as one of its advantages. The CAN message contains 0 to 8 bytes of information. The data transfer rate can be chosen in the range 125kb-1Mb/s, data transfer distance is dependent on the chosen speed. One CAN bus can contain 2032 objects according to CAN 2.0A specifications [8].

LON is originally a field bus developed mainly for building automation [10]. In the LON bus the processor and memory reside in each control unit, meaning that the net does not need a centralized control. The most important application areas iclude real estate, industrial and process automation, vehicles etc small systems. The devices of different systems can be connected to the same LON bus. The data transfer channel is open and the devices of several different manufacturers can be connected to it.

5.3. **The GA used.** GA simulates the messages that the field bus message handler receives from the I/O-cards and display/interface cards when communicating through the CAN bus. When communicating via the LON bus, it simulates other similar units in the net. Furthermore, it

	Random	*GA*
`Count`	10,000	10,000
`Mean` [ms]	155.47	214.73
`Std. deviation` [ms]	65.52	101.89
`Minimum` [ms]	19.00	22.00
`Maximum` [ms]	444.00	576.00
`Range` [ms]	425.00	554.00
`Median` [ms]	157.00	194.00

TABLE 1. Descriptive statistics of response times of test cases created by the GA and the pure random method.

simulates the operation of the digital signal processing unit and writes information directly in the databases located in a common memory. A test case record contains among others the delay between two CAN messages, an optional LON load message, send times, data fields, and sender addresses. The fitness value is the time from the sending of the CAN message to the receiving of the CAN answer message.

The first generation in GA is created using the Windows random number generator. The genetic operators used to create new generations were single point and uniform crossovers together with mutation. We re-evaluate the individuals selected from the previous generation, because the system is rather non-deterministic. The most important parameters of the GA used in this study have been: single point and uniform crossover of equal rate, the total crossover rate being 70%, mutation rate 8%, population size 100 of which 30% of the best are selected for the next generation (elitism).

6. RESULTS

In this work both the genetic algorithm and the tested software were run under the ESIM simulator in the same workstation, which was a PC equipped with a Pentium 200 mmx processor and Windows 95 operating system. Observe that the processing times should not be directly compared to those of the target environment, in which the object software runs considerably faster.

In order to compare pure random and a GA based method, 10,000 test cases were generated and tested by both methods. Table 1 represents the corresponding descriptive statistics. ¿From it we can see that the average response time of test cases created by GA is much longer (38%) than that of the random method. This implies that GA is able to find and favor some test cases that lead to longer response times.

Figure 1 shows the distribution of response times for both methods. As we can see the histograms are quite similar, except that the GA method has an additional peak between 300 and 400 ms. Obviously GA has been able to identify some input parameter combinations resulting to longer response times. In order to find those parameters that cause the difference we calculated the correlations between all input variables and response time. Almost all of these correlations were very small, of magnitude 0.1 or less. The highest correlation (0.45) was found between the response time and in which state of the CAN message processing the LON message was sent.

To demonstrate how strong the effect of LON message sending is on the response time we draw figure 2, where test cases were divided into two groups “no LON messages” and “LON message sent during the test case”. It is clear that the additional peak at the longer response time end was almost totally created from test cases, where the LON message was sent. So it is obvious that

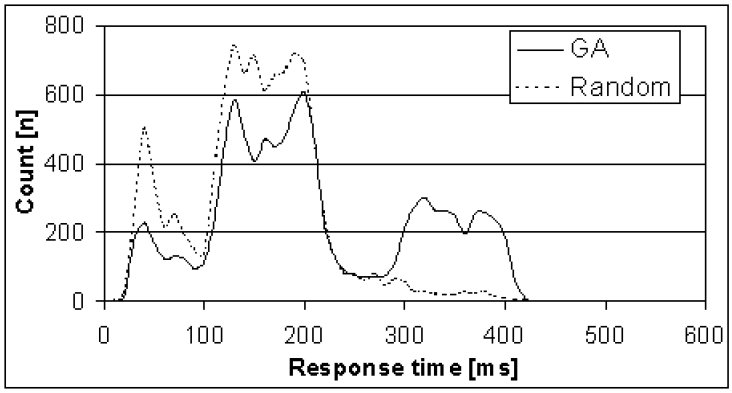

FIGURE 1. Response times distribution for the GA and random methods.

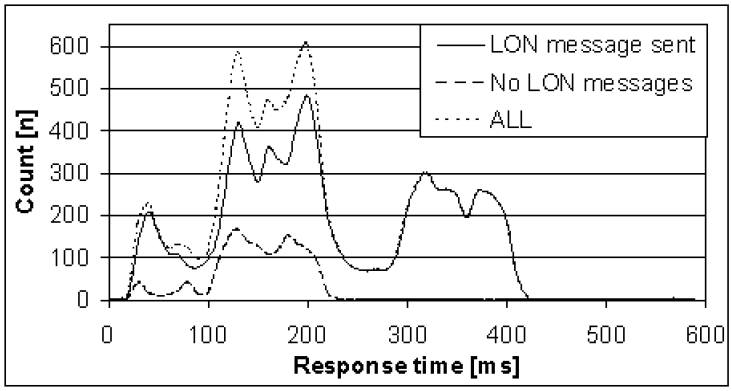

FIGURE 2. Response time dependensy on LON sending.

the GA based method favors the LON message sent during the test case more than the random method. We identify six different states of CAN message processing when a LON message can be sent (figure 3). It is obvious that the longest response times can be expected if the LON message is sent shortly before the CAN message. This is probably due to the fact, that in that case the program starts to process the received LON message, before the higher priority CAN message arrives causing the interrupt handler to start the processing of the CAN message. In figure 3 both the GA and random methods perform similarly in all six classes, while the overall response time is much higher with the GA method.

Furthermore we divided all test cases into the same six classes given in figure 3 and figure 4. From the latter it is obvious that GA learns the problematic test parameters and creates many more test cases for the most problematic class B than the random method. The time segments in the six classes are not equal, which explains that neither method creates evenly distributed test cases.

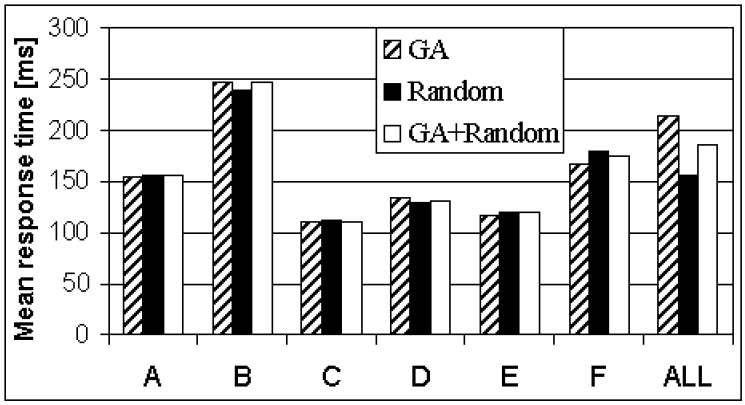

FIGURE 3. The six CAN message processing states, when LON load message can be sent. A=no LON message, B=LON message shortly before the CAN message, C=LON and CAN message simultaneously, D=LON message same time as program processes CAN message, E=LON message simultaneously as tested program starts to generate replay to the CAN message, F=LON message after E, but before replay CAN message comes back from the program.

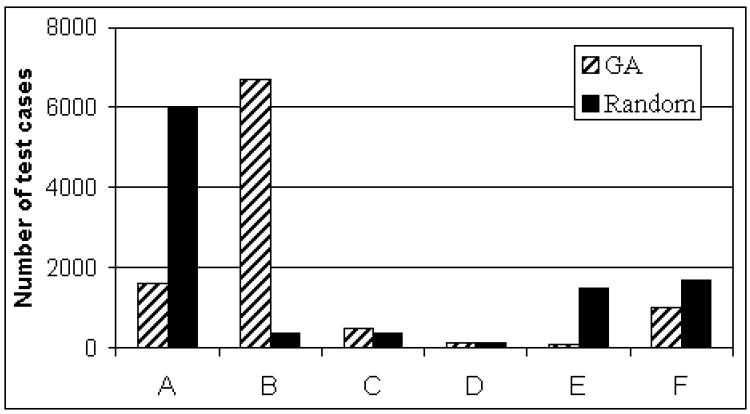

FIGURE 4. Distribution of test cases produced by the GA and random methods. For the notations used see figure 3.

6.1. **Comments and discussion.** It was discovered that at the beginning of the testing the first couple of test cases caused the longest response time. The tested software is a state-machine. Thus we will get rid of this problem if we succeed to move the state-machine between the tests to a more random state.

Because of the non-determinism every test case should be evaluated several times, the average of which could be used as the fitness value for each test case.

The genetic algorithm parameters effect on the test results was also studied briefly, but this optimization problem was not especially sensitive to the GA parameters. This is in good agreement with our and other studies on the sensitivity of GA based optimization [1, 4].

7. Conclusions

This study shows that the genetic algorithm is applicable in pure black box type software testing. It was significantly more efficient in finding the suspect input parameter sets than random testing. When analyzing the results it turned out that the simple statistical analysis combined with some graphics is most useful.[21]

We find the results of this study encouraging and recommend that GA based optimization can be seriously considered when developing software testing automation. However, more research on the possibilities of different kind of program testing with GA should be done in order to evaluate its full potential.

Acknowledgements. This work is a part of a larger project applying GAs in industrial optimization problems and has been supported by the Finnish Technology Development Center (Tekes) and ABB Corporate Research.

Further information on this work can be found in our anonymous ftp server (`ftp.uwasa.fi`) in directory `cs/report99-2` and also raports [5, 6]. The work was based on the thesis done by one of the authors (T. M.) [15] under the guidance of J. A. The project and especially the analysis of results is also reported in [21]. Lilian Grahn is acknowledged for her help with the proofreading of this paper.

References

1. Jarmo T. Alander, *An indexed bibliography of genetic algorithms*, Practical Handbook of Genetic Algorithms: New Frontiers (Lance D. Chambers, ed.), New Frontiers, vol. III, CRC Press, Inc., Boca Raton, FL, 1999, pp. 503–572.
2. ———, *Indexed bibliography of genetic algorithms in computer science*, Report 94-1-CS, University of Vaasa, Department of Information Technology and Production Economics, 1999.
3. ———, *Indexed bibliography of genetic algorithms in electronics and VLSI design and testing*, Report 94-1-VLSI, University of Vaasa, Department of Information Technology and Production Economics, 1999, (`ftp.uwasa.fics/report94-1/gaVLSIbib.ps.Z`).
4. ———, *Population size, building blocks, fitness landscape and genetic algorithm search efficiency in combinatorial optimization: An empirical study*, Practical Handbook of Genetic Algorithms: New Frontiers (Lance D. Chambers, ed.), New Frontiers, vol. III, CRC Press, Inc., Boca Raton, FL, 1999, pp. 459–485.
5. Jarmo T. Alander, Timo Mantere, Ghodrat Moghadampour, and Jukka Matila, *Searching protection relay response time extremes using genetic algorithm-software quality by optimization*, Electric power systems research **46** (1998), 229–233.
6. Jarmo T. Alander, Timo Mantere, and Pekka Turunen, *Genetic algorithm based software testing*, Artificial Neural Nets and Genetic Algorithms, Proceedings of International Conference (ICANNGA97) (Norwich (UK)) (George D. Smith, ed.), Springer-Verlag, Wien, April 1997, pp. 325–328.
7. Antti Auer and Jukka Korhonen, *State testing of embedded software*, EuroSTAR '95, November 1995, Olympia confrence centre, London, England (1995).
8. Bosch, *CAN specification, version 2.0*, Robert Bosch GmbH, Posttach 50, D-7000 Stuttgart 1, 1991.
9. M. Chaufer and T. Mosser, *Testing embedded systems*, Touloise 89: Proceedings of the Second International Workshop on Software Engineering and Its Applications. Touloise, France, 4-8 December 1989. Nanterre, France. (1989).
10. Echelon, *LON talk protocol specification. version 3.0, number 19550*, Echelon Corporation, Palo Alto, CA, 1994.
11. John Holland, *Adaptation in natural and artificial systems*, University of Michigan Press. Reissued by MIT Press, 1992, Ann Arbor, MI, 1975.
12. Hannu Honka, *A simulation-based approach to testing embedded software, vtt publications 124*, VTT, Technical research centre of Finland, Espoo, Finland, 1992.
13. B. F. Jones, D. E. Eyres, and H.-H. Sthamer, *A strategy for using genetic algorithms to automate branch and fault-based testing*, The Computer Journal **41** (1998), no. 2, 98–107.

14. B. F. Jones, H.-H. Sthamer, X. Yang, and D. E. Eyres, *The automatic generation of software test data sets using adaptive search techniques*, Proceedings of the Software Quality Management 3 (Seville, Spain), vol. 2, Computational Mechanics Publications, Ltd., Southhampton, UK, 3.-5. April 1995, pp. 435–444.
15. Timo Mantere, *Automaattinen ohjelmien testaus geneettisten algoritmien avulla [Automatic program testing by optimizing with genetic algorithms]*, University of Vaasa, Vaasa, Finland, 1999.
16. Gary E. McGraw, Christoph C. Michael, and Michael A. Schatz, *Generating software test data by evolution*, Technical Report RSTR-018-97-01, RST Corporation, 1998.
17. G. J. Meyers, *Art of software testing*, John Wiley&Sons, New York, 1979.
18. C. C. Michael, G. E. McGraw, M. A. Schatz, and C. C. Walton, *Genetic algorithms for dynamic test data generation*, Proceedings of the 12th IEEE International Conference Automated Software Engineering (Incline Village, NV (USA)), vol. ?, IEEE Computer Society Press, Los Alamitos, CA, 1.-5. November 1997, pp. 307–308.
19. Christoph C. Michael and Gary E. McGraw, *Opportunism and diversity in automated software test data generation*, Technical Report RSTR-003-97-13, RST Corporation, 1997.
20. Christoph C. Michael, Gary E. McGraw, Michael A. Schatz, and Curtis C. Walton, *Genetic algorithms for dynamic test data generation*, Technical Report RSTR-003-97-11, RST Corporation, 1997.
21. Ghodrat Moghadampour, *Using genetic algorithms to testing a protection relay software - a statistical analysis*, University of Vaasa, Vaasa, Finland, 1999.
22. Min Pei, Erik D. Goodman, Zongyi Gao, and Kaixiang Zhong, *Automated software test data generation using a genetic algorithm*, Technical Report 6/2/94, Beijing University of Aeronautics and Astronautic, 1994.
23. Prosoft, *Esim - testing environment for embedded software, user's guide version 2.1 for Windows NT*, Prosoft Oy, Oulu, Finland, 1995.
24. R. J. Martin R. A. DeMillo, W. M. McCracken and J. F. Passafiume, *Software testing and evaluation*, Benjamin/Cummings Publishing, Menlo Park, California, 1997.
25. Marc Roper, Iain Maclean, Andrew Brooks, James Miller, and Murray Wood, *Genetic algorithms and the automatic generation of test data*, Research report RR/95/195 [EFoCS-19-95], University of Strathclyde, Department of Computer Science, 1995.
26. Alison Lachut Watkins, *The automatic-generation of test data using genetic algorithms*, Proceedings of the 4th Software Quality Conference (Dundee (UK)) (I. M. Marshall, W. B. Samson, and D. G. Edgar-Nevill, eds.), vol. 2, University of Abertay Dundee, Scotland, 4.-5. July 1995, pp. 300–309.
27. S. Xanthakis, C. Ellis, C. Skourlas, A. Le Gall, S. Katsikas, and K. Karapoulios, *Application of genetic algorithms to software testing [Application des algorithmes génétiques au test des logiciels]*, Proceedings of the 5th International Conference on Software Engineering (Toulouse, France), 7.-11. December 1992, pp. 625–636.

Department of Information Technology and Industrial Economics, University of Vaasa, PO Box 700, FIN-65101 Vaasa, Finland

E-mail address: `firstname.lastname@uwasa.fi`