# Testing the Temporal Behavior of Real-Time Engine Control Software Modules using Extended Evolutionary Algorithms

**H. Pohlheim**, **J. Wegener**, **H. Sthamer**, Berlin

Many industrial products use embedded computer systems. Usually these systems have to fulfill real–time requirements, and correct system functionality depends on their logical correctness as well as on their temporal correctness. Therefore the developed systems have to be thoroughly tested in order to detect deficiencies in temporal behavior, as well as to strengthen the confidence in temporal correctness.
Existing test methods are not specialized in the examination of temporal correctness. For this reason new test methods are required which concentrate on determining whether the system violates its specified timing constraints. Normally, a violation means that outputs are produced too early or their computation takes too long. The tester's task therefore is to find the input situations with the longest or shortest execution times in order to check whether they produce a temporal error. It is virtually impossible to find such inputs by analyzing and testing the temporal behavior of complex systems manually. However, if the search for such inputs is interpreted as a problem of optimization, evolutionary computation can be used to find the inputs with the longest or shortest execution times. This search for accurate test data by means of evolutionary computation is called Evolutionary Testing.
Experiments using Evolutionary Testing on a number of real world programs have successfully identified new longer and shorter execution times than had been found using other testing techniques. Evolutionary Testing, therefore, seems to be a promising approach for the verification of timing constraints.
For the optimization we employed extended Evolutionary Algorithms. This includes the use of multiple subpopulations, each using a different search strategy. Additionally, competition for limited resources between these subpopulation have taken place, providing an efficient distribution of the resources.

## 1 Introduction

In real–time computing the correctness of the system does not only depend on the logical result of the computation but also on the time when the results are produced. Real–time systems play an important role in our lives and they cover a spectrum from the very simple to the very complex. Examples of current real–time systems include applications in the fields of aerospace, automotive, and railway electronics. These are often safety–relevant systems which, in case of error, may cause considerable material damage or may even endanger human lives. At present there is still a lack of suitable methods and tools for the development of real–time systems which permit a coherent treatment of correctness, timeliness, and fault tolerance in large scale distributed computations. Consequently, real–time system design is mostly done unsystematically, and timing constraints are verified with ad hoc techniques, or with costly and extensive simulations (cf. [15]).

Testing is one of the most complex and time-consuming activities within the development of real-time systems [6]. Typically, it consumes 50 % of the overall development effort and budget since embedded systems are much more difficult to test than conventional software systems ([2], [8], [1], [18]). The examination of additional requirements like timeliness, simultaneity, and predictability make the test costly. In addition, testing is complicated by technical characteristics like the development in host-target environments, the strong connection with the system environment, the frequent use of parallelism, distribution and fault-tolerance mechanisms, as well as the utilization of simulators.

For examining temporal correctness no specialized test methods are available which are suited for industrial use [19]. For this reason we have developed and examined a new approach for testing temporal behavior. This approach is based on the use of Evolutionary Algorithms, therefore we call it Evolutionary Testing. Our work investigates the effectiveness of Evolutionary Algorithms to validate the temporal correctness of embedded systems by establishing the maximum and minimum execution times.

Section 2 contains an overview of Evolutionary Testing and describes how it is applied to examine the temporal behavior of real-time systems. The results of several experiments will be described and discussed in Section 3. Section 4 gives concluding remarks and a short outlook on current and future work.

Pohlheim, H., Wegener, J., Sthamer, H.: Testing the Temporal Behavior of Real-Time Engine Control Software Modules using Extended Evolutionary Algorithms. VDI-Berichte 1526, Düsseldorf: VDI-Verlag, pp. 61-65, 2000.
GMA/GI Fachtagung: Computational Intelligence im industriellen Einsatz, Baden-Baden, 11./12. Mai 2000

# 2 Evolutionary Testing

The major objective of testing is to find errors. Real-time systems are tested for logical correctness by standard testing techniques such as the classification-tree method [4]. A common definition of a real-time system is that it must deliver the result within a specified time interval and this adds an extra dimension to the validation of such systems, namely that their temporal correctness must be checked.

The temporal behavior of real-time systems is defective when the computation of input situations violate specified timing constraints. In general, this means that outputs are produced too early or their computation takes too long. The tester's task therefore is to find the input situations with the shortest or longest execution times to check whether they produce a temporal error.

The search for the shortest and longest execution times can be regarded as an optimization problem to which Evolutionary Algorithms seem an appropriate solution.

## 2.1 Investigating search space

In contrast to other optimization methods, Evolutionary Algorithms are particularly suited for problems involving large numbers of variables and complex input domains. Even for non-linear and poorly understood search spaces Evolutionary Algorithms have been used successfully [19].

With the exception of simple real-time systems the temporal behavior always forms a very complex multi-dimensional search space with many plateaus and discontinuities. On the one hand the execution times for several input data executing the same program path can be identical. On the other hand the execution of different program paths leads to irregular changes of the execution times. Two examples are provided in figure 1.

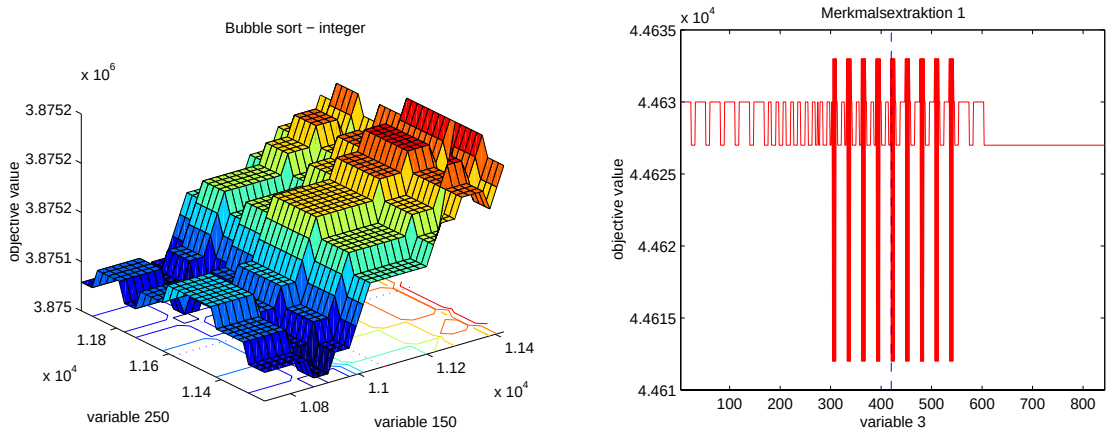


Figure 1: Visualization of execution times for a very small area of the search space of two software systems; left: Bubble-sort algorithm - variation of two variables (out of 500 variables), right: feature extraction - variation of just one variable (out of more than 800 variables)

## 2.2 Application of Evolutionary Algorithms

When using Evolutionary Testing for determining the shortest and longest execution times of test objects, each individual of the population represents a test datum with which the system under test is executed. In our experiments the initial population is generated at random or obtained by systematic testing (and randomized). Thus, the evolutionary approach benefits from the tester's knowledge of the system under test. For every test datum, the execution time is measured. Afterwards, the population is sorted according to the execution times determined. The fitness assigned to each individual depends only on its position in the individuals rank and not on the actual execution time. If one searches for the worst-case execution time, test data with long execution times obtain high fitness values. Conversely, when searching for the best-case execution time, individuals with short execution times obtain high fitness values.

For the optimization we employed extended Evolutionary Algorithms. This includes the use of multiple subpopulations, each using a different search strategy. Additionally, competition for limited resources between these subpopulations have taken place. For a detailed discussion of Evolutionary Algorithms and

the used extensions see [13]. The Genetic and Evolutionary Algorithm Toolbox for use with Matlab - GEATbx [12] contains an implementation of these methods and was used for the experiments.

We use an extended population model inside our evolutionary algorithms. Besides using multiple subpopulation (regional population model or migration model) and migration, every subpopulation employed different evolutionary parameters (multiple strategies). Competition between the subpopulation takes place every couple of generations. Successful subpopulation receive more individuals, other subpopulation shrink in size. Thus, an automatic distribution of resources takes place.

The evolutionary process repeats itself, starting with selection, until a given stopping condition is reached, e.g. a certain number of generations is reached, or an execution time is found which is outside the specified timing constraints. In this case, a temporal error is detected and the test is successful. If all the times found meet the timing constraints specified for the system under test, confidence in the temporal correctness of the system is substantiated. However, in this case it becomes considerably more difficult to decide when the test should be terminated in order to establish sufficient confidence in the temporal correctness of the test object. Therefore, DaimlerChrysler Research and Technology and the Stanford University are currently working on developing suitable stopping criteria [17].

Evolutionary Testing enables a totally automated search for extreme execution times. The test especially benefits from the fact that test evaluation concerning temporal behavior is usually trivial. Contrary to logical behavior, the same timing constraints apply to large numbers of input situations.

In the experiments presented in the following section, evolutionary testing was stopped after a predefined number of generations, specified according to the complexity of the test objects with respect to number of input parameters and lines of code.

## 2.3 Testing temporal system behavior

We have used Evolutionary Algorithms in several experiments to determine the shortest and longest execution times of different systems.

For the measurement of the execution times we used hardware timers of the target system. The hardware timer is started before executing the test datum and returns the time value directly after the execution finishes.

In Figure 1 an overview of the integration between the Evolutionary Algorithm and the time measurement equipment is given.

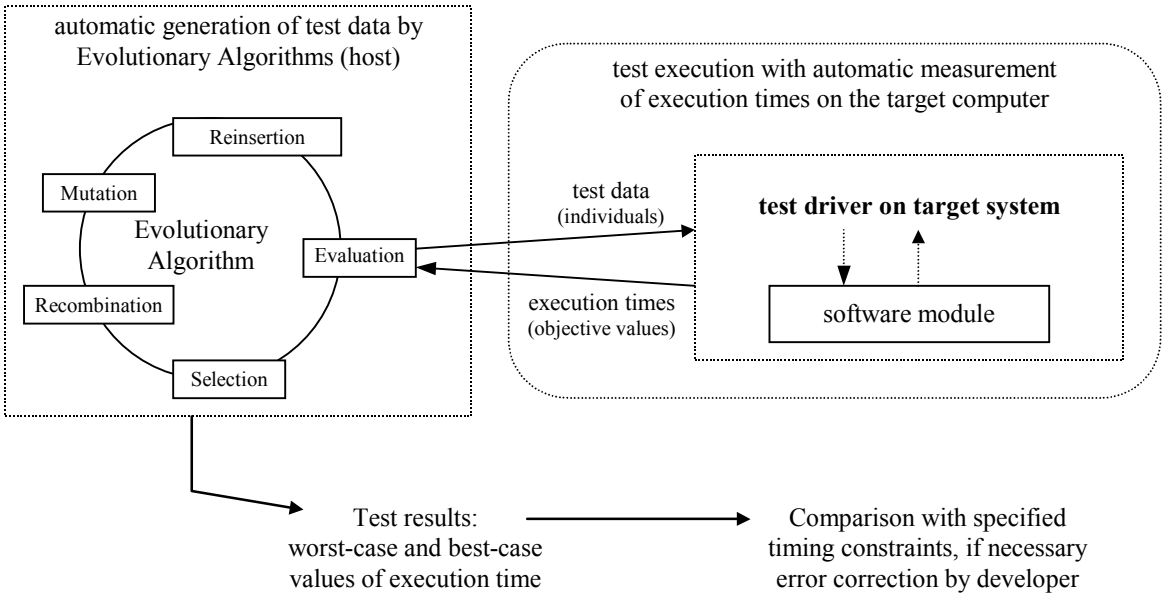


Figure 1: Scheme of Evolutionary Testing for temporal correctness

# 3 Evolutionary Testing in Practice

Over the last three years many experiments involving Evolutionary Testing were carried out at Daimler-Chrysler, Research and Technology. In this paper we describe results from one application field: an engine control system. The system contains several modules which have to fulfill timing constraints.

## 3.1 Testing engine control software modules

The test of the engine software modules was carried out on the target system (target processor later used in the vehicles). The execution times were measured by means of hardware timers. Thus, the time needed for the objective function call was low (30-50 seconds per generation, nearly independent of the number of individuals to evaluate). Because of the lower number of module parameters (variables of the Evolutionary Algorithm), one optimization run needed not more than 50 to 100 generations to reach a good result.

Since the software modules of the engine control system are parameter optimization problems we used parameters best suited for parameter optimization (discrete recombination and integer mutation with multiple ranges). For an extensive description of Evolutionary Algorithms suited for parameter optimization see [13], especially chapter 7.

| | max. exec. time in µs | | | |
|---|---|---|---|---|
| module name | Evolutionary Testing | developer test | lines of code | para-meters |
| zr2 | 69,6 µs | 67,2 µs | 41 | 18 |
| t1 | 120,8 µs | 108,4 µs | 119 | 18 |
| mc1 | 112,0 µs | 108,4 µs | 98 | 17 |
| mr1 | 68,8 µs | 64,0 µs | 81 | 32 |
| k1 | 59,6 µs | 57,6 µs | 39 | 14 |
| zk1 | 58,4 µs | 54,0 µs | 56 | 9 |

Table 1: Maximum execution times of engine control software modules determined by Evolutionary, systematic, and random testing

The results of some of our optimizations are shown in table 1. For these systems we only list the found maximum execution times because only the worst-case execution time is of importance for the correct functioning of the engine control system.

For the engine software modules the maximum execution times are uncertain. Therefore, the results from Evolutionary Testing were compared to the maximum execution times determined by the developers through systematic testing. These values are shown in table 1 in the third column labeled developer test.

A comparison of the results between developer test and evolutionary test showed that Evolutionary Testing found longer execution times for all the given modules. This is especially astonishing because the Evolutionary Algorithm treats the software modules as black boxes.

If these promising results can be established during further optimizations developers of real-time software modules would gain an efficient tool for the determination of minimum and maximum execution times for their software modules. A lot of former testing by hand could be carried out automatically with the Evolutionary Testing. Results should be as good or better.

# 4 Concluding remarks

In various experiments, Evolutionary Testing has been successfully applied in search of the longest and shortest execution times of real–time programs in order to check whether they violate specified timing constraints.

Evolutionary Algorithms show considerable promise in testing and validating the temporal correctness of real–time systems, and further research work in this area should prove fruitful. Yet, more work is still needed to find the most appropriate and robust parameters for Evolutionary Testing, so that the extreme execution times can be detected efficiently and with a high degree of certainty in practical operation.

However, the use of Evolutionary Algorithms alone is not sufficient for a thorough and comprehensive test of real–time systems. A combination with existing test procedures is necessary to develop an effective test strategy for embedded systems. The combination of systematic testing and Evolutionary Testing is promising [19].

If the evolutionary search does not start with a randomly generated population, but with a set of test data systematically determined by the tester, the disadvantage of Evolutionary Algorithms, namely that they

might not find certain test relevant value combinations, can be compensated for. Moreover, Evolutionary Testing benefits from the tester's knowledge of the program function or structure.

In future it is also intended to examine the combination with static analyses more closely [11]. By combining both approaches the area in which one finds the extreme execution time of the system can be closely defined, e.g. static analyses give an upper estimate for the maximum execution time and testing gives a lower estimate for the maximum execution times.

# References


[1] *Beizer, B.*: Software Testing Techniques. New York: Van Nostrand Reinhold, 1990.

[2] *Davis, C.G.*: Testing Large, Real–Time Software Systems. Software Testing, Infotech State of the Art Report, Vol. 2, 1979, pp. 85–105, 1979.

[3] *Grochtmann, M., and Grimm, K.*: Classification Trees for Partition Testing. Software Testing, Verification & Reliability, Vol. 3, No. 2, pp. 63-82, 1993.

[4] *Grochtmann, M., and Wegener, J.*: Test Case Design Using Classification Trees and the Classification-Tree Editor CTE. Proceedings of Quality Week '95, San Francisco, USA, 1995.

[5] *Grochtmann, M., and Wegener, J.*: Evolutionary Testing of Temporal Correctness. Proceedings of Quality Week Europe '98, Brussels, Belgium, 1998.

[6] *Heath, W.S.*: Real-Time Software Techniques. Van Nostrand Reinhold, New York, USA, 1991.

[7] *Hennell, M.A., Hedley, D., and Riddell, I.J.*: Automated Testing Techniques for Real-Time Embedded Software. Proceedings of the European Software Engineering Conference ESEC '87. Strasbourg, France, 1987.

[8] *Hetzel, B.*: The Complete Guide to Software Testing. Wellesley, MA: QED Information Sciences, 1988.

[9] *Jones, B.F., Eyres, D.E., and Sthamer, H.-H.*: A Strategy for using Genetic Algorithms to Automate Branch and Fault-based Testing. The Computer Journal. Vol. 41, No. 2, pp. 98-107, 1998.

[10] *Mathworks, The*: Matlab - UserGuide. Natick, Mass.: The Mathworks, Inc., 1994-2000. http://www.mathworks.com/

[11] *Mueller, F. and Wegener, J.*: A Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints. Proceedings of the IEEE Real-Time Technology and Applications Symposium RTAS '98, pp. 144-154, 1998.

[12] *Pohlheim, H.*: GEATbx - Genetic and Evolutionary Algorithm Toolbox for use with Matlab. http://www.geatbx.com/, 1994-2000.

[13] *Pohlheim, H.*: Evolutionäre Algorithmen - Verfahren, Operatoren, Hinweise aus der Praxis. Berlin, Heidelberg: Springer-Verlag, 1999. http://www.pohlheim.com/eavoh/index.html

[14] *Schultz, A.C., Grefenstette, J.J., and De Jong, K.A.*: Test and Evaluation by Genetic Algorithms. IEEE Expert. Vol. 8, No. 5, pp. 9-14, 1993.

[15] *Stankovic, J.A.*: Misconceptions about Real–Time Computing - A Serious Problem for Next–Generation Systems. IEEE Computer, Vol. 21, No. 10, pp. 10–19, 1988.

[16] *Sthamer, H.-H.*: The Automatic Generation of Software Test Data Using Genetic Algorithms. PhD Thesis, Department of Electronics and Information Technology, University of Glamorgan, Wales, UK, 1996.

[17] *Sullivan, M. O', Vössner, S. and Wegener, J.*: Testing Temporal Correctness of Real-Time Systems - A New Approach Using Genetic Algorithms and Cluster Analysis. Proceedings of EuroSTAR'98, pp. 397-418, 1998.

[18] *Wegener, J. and Pitschinetz, R.*: TESSY - Yet Another Computer–Aided Software Testing Tool? Proceedings of the Second European International Conference on Software Testing, Analysis & Review EuroSTAR '94, 1994.

[19] *Wegener, J., and Grochtmann, M.*: Verifying Timing Constraints of Real-Time Systems by Means of Evolutionary Testing. Real-Time Systems, 15, pp. 275-298, 1998.