EuroSTAR’99

Testing the Temporal Behavior of Real-Time Tasks using Extended Evolutionary Algorithms

Joachim Wegener and Harmen Sthamer
DaimlerChrysler, Research and Technology, Alt-Moabit 96a, 10559 Berlin, Germany
phone: +49 30 39982 232, fax: 107
Joachim.Wegener@DaimlerChrysler.com

Hartmut Pohlheim
Brodberg 38, 14532 Kleinmachnow, Germany

Abstract

Many industrial products are based on the use of embedded computer systems. Usually, these systems have to fulfill real-time requirements. Correct system functionality depends on their logical correctness as well as on their temporal correctness. Therefore, the developed systems have to be tested not only with regard to their functional behavior but they also have to be thoroughly tested in order to detect existing 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. Within the past couple of years DaimlerChrysler Research and Technology has developed a new testing procedure for testing the temporal behavior of real-time systems: namely the evolutionary test. The test is interpreted as a problem of optimization, and employs evolutionary computation to find the test data with extreme execution times.

Evolutionary operators used for the evolutionary test have been constantly improved and upgraded during this period of time. During the initial work phases test data generation was still based on genetic algorithms. Meanwhile, extended evolutionary algorithms have been established specifically as adequate operators for testing temporal behavior.

This paper introduces approved extended evolutionary algorithms and illustrates their application for testing the temporal behavior of a bubblesort procedure. Further, an account will be given of the first industrial application of the evolutionary test during the development of a motor control system.

0 Introduction

Many industrial products are based on the use of embedded computer systems. Usually, these systems have to fulfill real-time requirements, correct system functionality depending on their logical correctness as well as on their temporal correctness. Therefore, the developed systems have to be tested not only with regard to their functional behavior but they also have to be thoroughly tested in order to detect existing 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 task of the tester 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. Evolutionary testing has already proved through various experiments to be a promising way for testing the temporal behavior of real-time systems ([5], [10], [13],[17]).

In earlier works genetic algorithms have been used for test data generation ([19], [20]), whereas this paper uses extended evolutionary algorithms for the evolutionary test. This includes the use of multiple subpopulations of test data, each using a different search strategy. Competition for limited resources between these subpopulations, is introduced to provide an efficient distribution of resources. The extended evolutionary algorithms upon which this work is based overcome problems by using a bit-coding of input data specific to genetic algorithms, e.g. a combination of evolutionary algorithms with neighborhood search techniques proves unnecessary. This increases the efficiency of evolutionary testing.

This paper introduces extended evolutionary algorithms that have shown especially good results within a series of tests of the temporal behavior of real-time systems. An exemplary application for testing the temporal behavior of a bubblesort procedure will be carried out. Further, an account will be given of the first industrial application of the evolutionary test that is being developed for testing the temporal behavior of diverse tasks of a motor control system. The results of evolutionary testing of the motor control system are compared to the maximum execution times determined by the developers of the tasks with systematic testing.

The first section gives a brief introduction to testing real-time systems. Section 2 contains an overview of evolutionary testing and describes how extended evolutionary algorithms are applied to examine the temporal behavior of real-time systems. The results of testing the bubblesort procedure and the motor control system will be described and discussed in Section 3. Section 4 gives concluding remarks and a brief outlook on future work.

1 Testing Real-Time Systems

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 life, and they cover a spectrum from the very simple to the very complex. Examples of current real-time systems include applications from the field of aerospace, automotive, and railway electronics. These are often safety-relevant systems which, in case of error, can cause considerable material damage or can 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 (cf.[14]).

Testing is one of the most complex and time-consuming activities within the development of real-time systems [6]. It typically consumes 50 % of the overall development effort and budget since embedded systems are much more difficult to test than conventional software systems ([2], [3], [7], [16]). The examination of additional requirements like timeliness, simultaneity, and predictability make the test costly. In addition, testing is complicated by technical characteristics such as the development in host-target environments, strong interaction with the system environment, frequent use of parallelism, distribution and fault-tolerance mechanisms, as well as the utilization of simulators.

Nevertheless, systematic testing is an inevitable part of the verification and validation process for software-based systems. Testing is the only method that allows a thorough examination of the test object's run-time behavior in the actual application environment. Dynamic aspects like the duration of computations, the memory actually needed during program execution, or the synchronization of parallel processes are especially important for the correct functioning of real-time systems.

Testing is aimed at finding errors in the systems and giving confidence in their correct behavior by executing the test object with selected inputs. For testing real-time systems, the examination of the logical system behavior alone is not sufficient. Additionally, real-time systems must be tested for compliance with their timing constraints. An investigation of existing software test methods shows that a number of proven functional and structural test methods are available for examining logical correctness but for examining temporal correctness there are no specialized test methods available which are suitable for industrial use. For this reason, we have developed and examined evolutionary testing – a new approach to testing temporal behavior which is based on the use of evolutionary algorithms [17].

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 input situations exist in such a manner that their computation violates the specified timing constraints. In general, this means that outputs are produced too early or their computation takes too long. The task of the tester therefore is to find the input situations with the shortest or longest execution times to check whether they produce a temporal error.

Evolutionary testing enables a totally automated search for extreme execution times. The search for the shortest and longest execution times is regarded as an optimization problem to which evolutionary algorithms are applied. 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.

2.1 Investigation of Search Space

With the exception of very simple real-time systems, the temporal behavior always forms a very complex multi-dimensional search space with many plateaus and discontinuities. This is due to the fact that the execution times for several input data executing the same program path could be identical; whereas the execution of different program paths leads to irregular changes of the execution times. Two examples are provided in Figure 1.

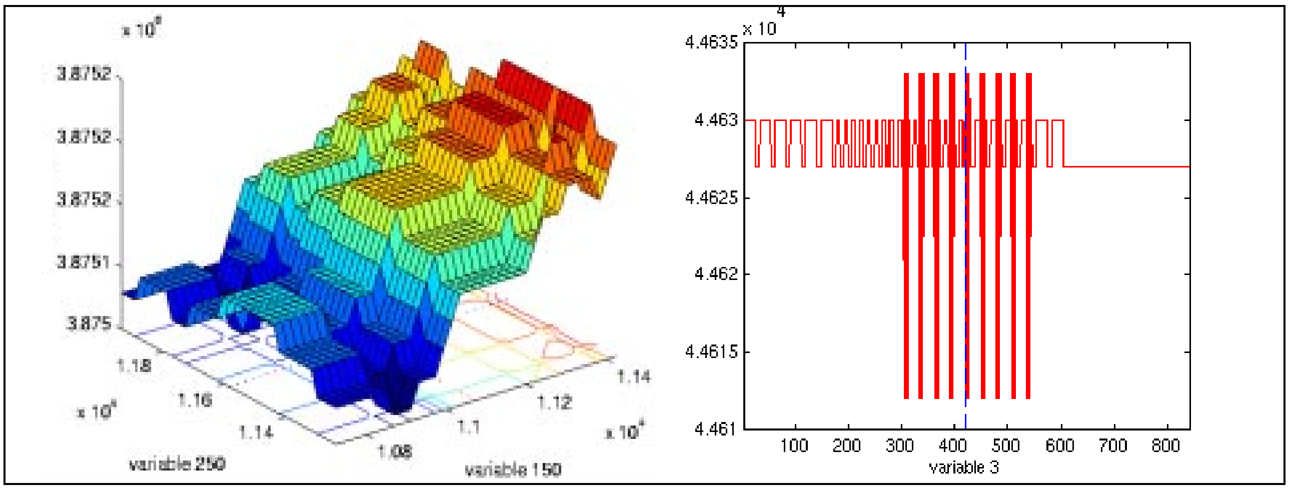

Figure 1: Visualization of execution times for a very small area of the search space of two software systems; left: Bubblesort procedure - variation of two variables (out of 500 variables), right: feature extraction - variation of just one variable (out of more than 800 variables)

Due to the complexity of the temporal behavior, it is not astonishing that some researchers in control theory have drawn the conclusion that computer systems are inherently probabilistic in terms of their timing behavior [15]. Correspondingly difficult is the testing of temporal behavior with conventional black-box and white-box methods. The evolutionary test therefore grounds on a stochastic procedure: evolutionary algorithms.

2.2 Evolutionary Algorithms

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 [17].

Evolutionary algorithms represent a class of adaptive search techniques and procedures based on the processes of natural genetics and Darwin’s theory of evolution. They are characterized by an iterative procedure and work in parallel on a number of potential solutions, the population of individuals. In every individual, permissible solution values for the variables of the optimization problem are coded.

The evolutionary search and optimization process is based on three fundamental principles: selection, recombination and mutation. The concept of evolutionary algorithms is to evolve successive generations of increasingly better combinations of those parameters which significantly affect the overall performance of a design. Starting with a selection of good individuals, the evolutionary algorithm achieves the optimum solution by the random exchange of information between these increasingly fit samples (recombination) and the introduction of a probability of independent random change (mutation). The adaptation of the evolutionary algorithm is achieved by the selection and reinsertion procedures since these are based on fitness.

2.3 Testing Temporal System Behavior using Extended 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. Usually, the initial population is generated at random. In principle, if test data has been obtained by a systematic test, these could also be used as an initial population [19]. Thus, the evolutionary test 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 (rank-based fitness assignment overcomes some problems which apply to proportional fitness assignment, and behaves in a more robust manner than the proportional fitness assignment [18]). In this way, the execution time for each test datum determines its fitness value. 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.

Members of the population are selected with regard to their fitness and subjected to recombination and mutation to generate new test data. By means of selection, it is decided which test data are chosen for reproduction. In order to retain the diversity of the population, and to avoid a rapid convergence towards a local optimum, a moderate selective pressure was applied. In this work we use stochastic universal sampling [1] as selection method.

For the recombination of test data, two recombination methods are applied:

- discrete recombination [21] and
- multi-point recombination (an extension of double-point crossover).

For mutation we also employ multiple methods:

- integer mutation using different range and precision parameters (an integer version of the mutation operator of [21]) and
- swap mutation using different ranges (exchange/swap of variables inside one individual, the range defines how far away from each other the variables can be to be exchanged).

The probability of mutating variables of an individual is set to be inversely proportional to its number of variables. The more dimensions one individual posesses, the smaller the mutation probability for each single variable. For each variable, the mutation probability is 1/number_of_variables, i.e. one mutation within one individual.

Afterwards it has to be checked if the generated test data are in the input domain of the test object. Invalid data are readjusted. The individuals produced are then evaluated by executing the test object with them and measuring the execution times.

The new individuals are united with the previous generation to form a new population according to the reinsertion procedures laid down. A reinsertion strategy with a generation gap of 90% is applied, i.e. only 90% offspring are produced. The next generation therefore contains 90% offspring and 10% parents. The best parents from the previous generation survive.

We also introduce an extended population model to our evolutionary algorithms. The entire population will be divided into a number of subpopulation and each subpopulation employs different evolutionary algorithms in order to follow a different search strategy, e.g. local searches or global searches. In order to combine local and global search strategies a migration of the best individuals takes place between the subpopulation at regular intervals. Additionally, every other subpopulation of one population compete compete with each other. Strong subpopulation receive more individuals, other subpopulation diminish in size. This results in an automatic distribution of resources.

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 bounds. 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.

Figure 2 shows the structure of the extended evolutionary algorithm employed.

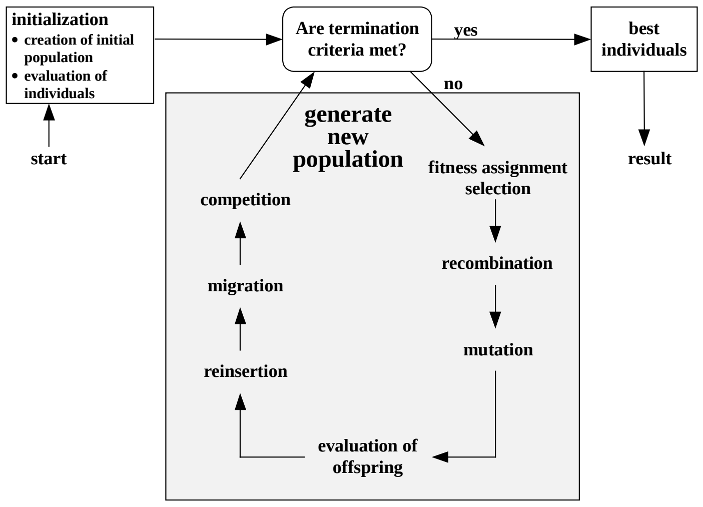

Figure 2: Structure of the extended evolutionary algorithm used

For a detailed discussion of the evolutionary algorithms mentioned and the extensions introduced consider one of the many available publications, e.g. [9] or [12]. The Genetic and Evolutionary Algorithm Toolbox (GEATbx [11]) for use with Matlab [8] contains an implementation of these methods and was used for our applications of evolutionary testing.

3 Application of Evolutionary Testing

Over the last three years several experiments involving evolutionary testing were carried out at Daimler-Chrysler Research and Technology to test its appropriateness for testing the temporal behavior or real-time systems, e.g. [5], [10], [17]. Promising results have been achieved; In comparison to the random test, the evolutionary test has achieved better results in all experiments. It also exhibits a number of advantages compared to static analyses and systematic tests.

In this paper we use evolutionary testing to determine the extreme execution times for two applications:

- The first test object is a bubblesort algorithm with 30 lines of code and 500 input parameters. This example is used for examining the optimization algorithms and for finding appropriate evolutionary parameters. The tests are carried out in a host environment using a SPARCStation 20 running with 200 MHz under Solaris 2.5. The performance measurement tool Quantify [22] is used to measure the execution times. The duration of executions is measured in processor cycles to rule out overheads by the operating system. Thus, the execution times reported are the same for repeated runs with identical parameters. Because of the algorithmic simplicity of the system, the minimum and maximum execution times can be easily calculated: the fully sorted list and the list sorted in reverse order.

- The second application field is a new motor control system for six- and eight-cylinder blocks, that is currently under development. The motor control system contains several tasks which have to fulfill timing constraints. Each task is a test object and is tested for its worst-case execution time by the developers using systematic testing. Evolutionary testing is used to verify the results from the developers' tests. The longest execution times determined by evolutionary testing are compared to the maximum execution times determined by the developers. The tests are performed on the target processor later used in the vehicles. The execution times are determined using hardware timers of the target environment with a resolution of 400 ns. A hardware timer is questioned directly before and after the execution of the tested task to determine the execution times. For the motor software modules the maximum execution times are not known for certain. This application happens to be the first industrial application of the evolutionary test.

Figure 3 gives an overview of the integration of evolutionary algorithm and the time measurement equipment. The scheme is similar for both methods described. Figure 3 depicts the second variant using a hardware timer of the target system.

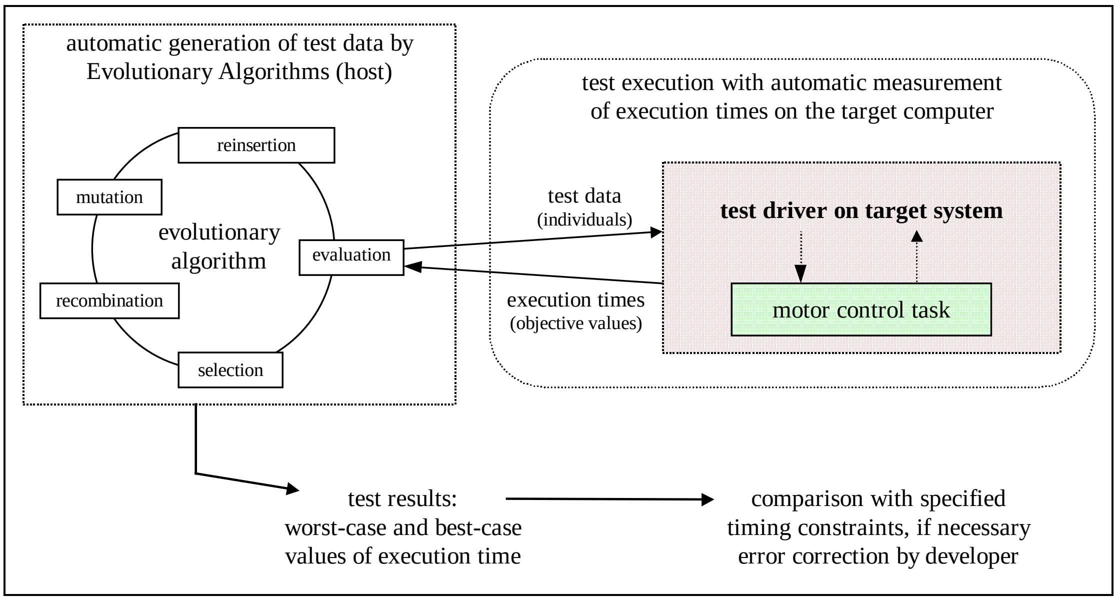
automatic generation of test data by Evolutionary Algorithms (host)
reinsertion
mutation
evolutionary algorithm
evaluation
recombination
selection
test execution with automatic measurement of execution times on the target computer
test data (individuals)
execution times (objective values)
test driver on target system
motor control task
test results: worst-case and best-case values of execution time
comparison with specified timing constraints, if necessary error correction by developer

Figure 3: Scheme of evolutionary testing for temporal correctness

In both experiments evolutionary testing is stopped after a predefined number of generations, specified according to the complexity of the test objects in respect to the number of input parameters and lines of code.

3.1 Testing Bubblesort Algorithm

In order to find the longest and shortest execution times, the example system bubblesort may be solved as parameter optimization problem or as an ordering problem. A combination of both methods is possible as well as using subpopulation with multiple strategies. Which of the algorithm variants or which combination is best suited for the solution of this problem was derived by the results of different optimization runs using multiple strategies and competition between these strategies.

The following operators were used to find the extreme execution times of the bubblesort procedure:

- 300 individuals (6 subpopulation with 50 individuals each) over 1000 generations. Normally we would use more individuals for a problem with 500 variables. However, this would extend the optimization times, due to the slow time registration in the UNIX environment, to more than one day,
- Selection: stochastic universal sampling, generation gap of 0.9,
- Recombination: double point and discrete recombination,
- Mutation: integer mutation and exchange/swap mutation, each with a mutation range of [0.2, 0.02, 0.002] and a mutation precision of 16,
- Migration: 20% every 20 generations,

- Competition (resource distribution) between subpopulation every 5 generations, competition rate of max. 5% of the worst individuals of the subpopulation, however subpopulation minimum of 17 individuals.

The recombination operators are equally well suited for this problem. The largest differences are constituted by the mutation operators. The integer mutation changes the values of the variables at their actual position within the individual. Only value of this variable is changed. By using different mutation ranges the size of these mutation steps can be adjusted. This outlines the kind of search: a large mutation range produces large mutation steps and leads to a rough search, whereas a small mutation range leads to a fine neighborhood search. The mutation precision defines the distribution of the mutation steps inside the mutation range.

The exchange or swap mutation exchanges the values of two variables inside one individual. Thus, the variable values are not changed, only the position within the individual varies. The mutation range predetermines how large the maximum distance between the variables to be exchanged is; similar to the integer mutation. The exchange mutation is a simple mutation operator for ordering problems that seemed appropriate for the bubblesort problem.

The following diagrams show the results of one test run of the bubblesort system. A search for the minimum execution time was carried out.

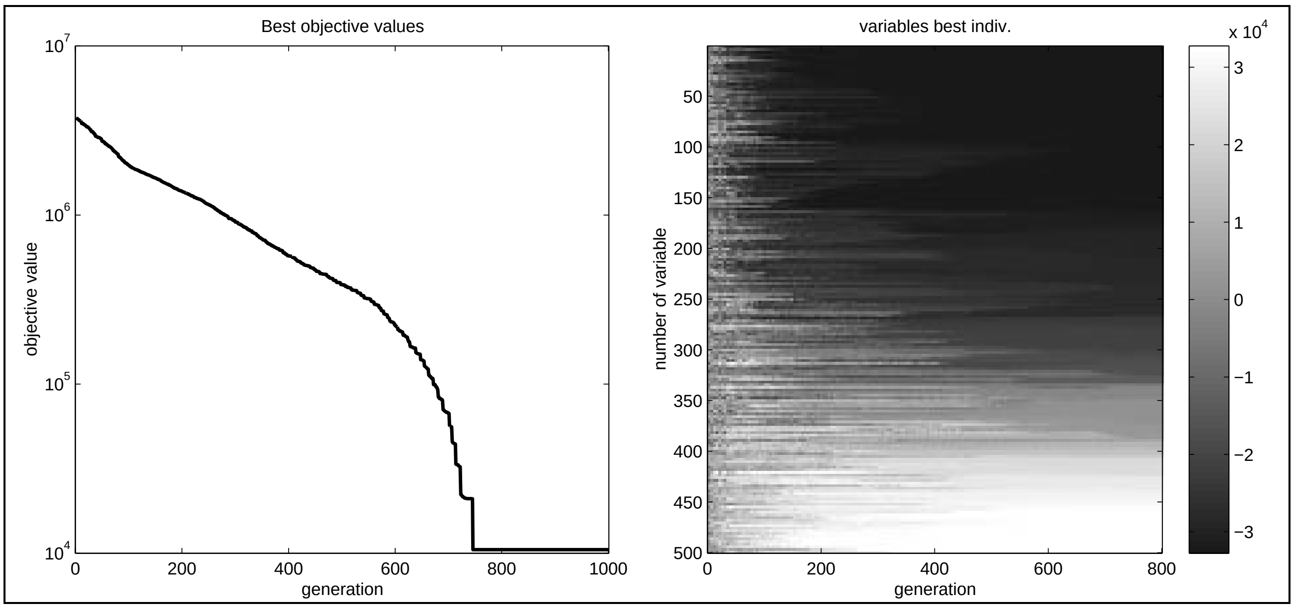

Figure 4: Results of bubblesort optimization; left: objective value of the best individual during all generations (convergence diagram); right: variable values of the best individual over all generations

Figure 4 shows the course of the objective values of the best individuals during all generations (left diagram). The logarithmic scaling shows clearly, that in generation 760 the optimum was reached. A sorted list with 500 entries was derived, controlled only by the execution time. Afterwards, the variables of the best individuals (see right-hand diagram) show only small changes which have no effect on the execution time – differently sorted lists have the same execution time.

In this example we used multiple strategies and competition between these strategies. During the analysis of the results we wanted to answer the following question: Which of the used mutation operators and the accompanying evolutionary parameters were well-suited for this problem? The analysis uses the two diagrams in figure 5, ranking of the subpopulation (left diagram) and relative size of the subpopulation (right diagram).

When looking at the ranking of the subpopulations (figure 5 left), a low rank characterizes a successful subpopulation (and thus a successful strategy). During the shown run, subpopulation 4 and 6 were equally successful until generation 350. At a later stage, subpopulation 4 was more successful than all other subpopulations. Beginning with generation 760, subpopulation 6 was more successful, but at this point the run had already reached the optimum. The ranking of the subpopulation corresponds with the size of the subpopulation. Until generation 350, subpopulation 4 and 6 include approximately the same amount of individuals. From then on the size of subpopulation 4 grows up to the possible maximum.

From these results it could be derived that the search strategies of subpopulation 4 and 6 are best suited for the solution of the bubblesort system. These two subpopulations use swap mutation with middle and small mutation steps (mutation range of 0.02 and 0.002) respectively.

In a further optimization run we employed the above result. Only two subpopulation with the above strategies (swap mutation with middle and small mutation range) were used. The number of individuals per subpopulation was set to 75, thus only 150 individuals were used. The optimum was found after 1050 generations. However, as we used fewer individuals, the overall number of objective function calls was much lower: 140.000 (1050 generations x 135 individuals per generation gap 90%) function calls for the second test run compared with 200.000 (760 x 270) objective function calls for the first test run with 6 subpopulation. Accordingly, the overall computation time was much lower: 16 hours compared to 24 hours before, 99% of this time were used by the objective function evaluations.

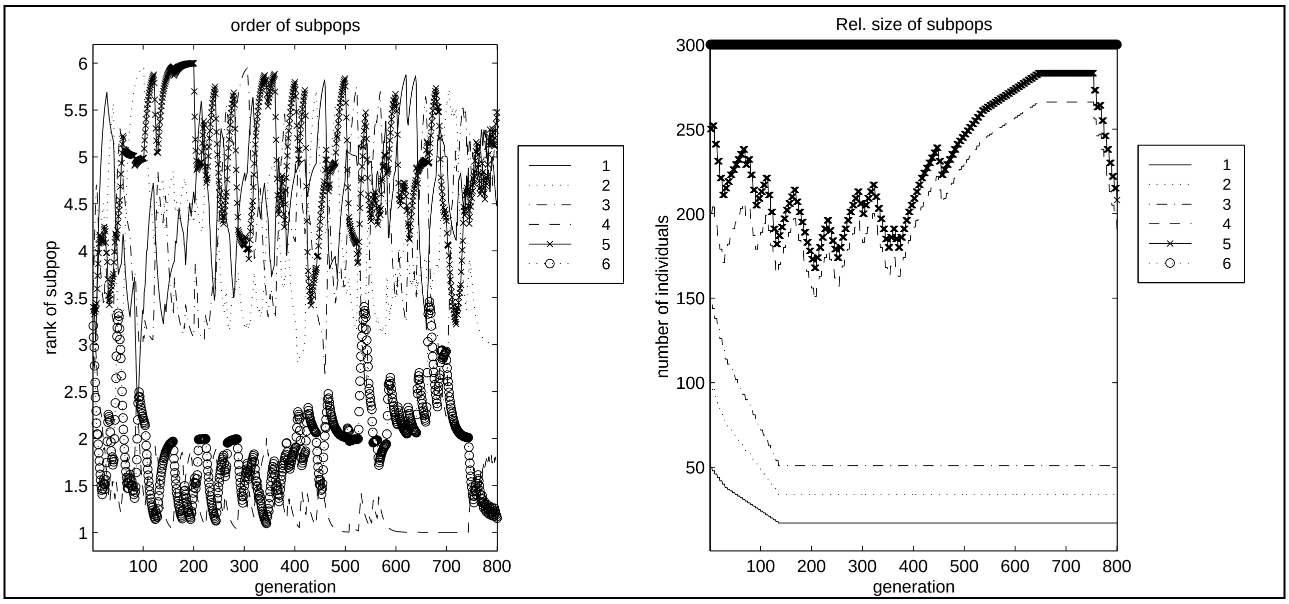

Figure 5: Results of bubblesort optimization, comparison between multiple strategies; left: ranking of subpopulations, right: size of subpopulation

3.2 Test of the Motor Control System

For the test of the motor control system an evolutionary test is employed in order to check the execution times that have been determined by the developers as the longest task execution times. As it is usual for many systems only the worst-case execution times of the tasks are of importance for the correct functioning of the motor control system, i.e. there are no tests for the shortest execution times. The test cases for testing the temporal behavior, defined by the developers are based on the functional specification of the system as well as on the internal structures of the tasks.

The test of the tasks of the motor control system is carried out on the actual target system. Thus, the time needed for the objective function call is much shorter: 30-50 seconds per generation, nearly independent of the number of individuals to be evaluated. Because of the lower number of input parameters (see table 1), an evolutionary test run needs just 50 to 100 generations with 150 individuals each to reach a good result, so that no test of a task takes longer than 1,5 hours. The test run is fully automatic.

For the application of the results derived from the test runs with the bubblesort system, we again used multiple subpopulation and employed different strategies and competition between subpopulation. Since the tasks of the motor control system are parameter optimization problems (and not ordering problems) we used parameters best suited for parameter optimization (discrete recombination and integer mutation with multiple ranges). The other parameters are similar to those used for the solution of the bubblesort system.

The results for the tasks are shown in Table 1. The table shows the maximum execution times determined by the developers with systematic testing (developer test) in comparison to the results achieved by evolutionary testing (evolutionary testing).

The comparison of the results between developer test and evolutionary test shows that evolutionary testing found longer execution times for all the given modules. This is especially astonishing, because evolutionary testing treats the software as black boxes whereas the developers are familiar with function and structure of their system. An explanation might be the use of system calls of which the effects on the temporal behavior can only be rated with difficulty by the developers.

Results from the tasks of the motor control system confirm the preliminary results from the comparison of systematic and evolutionary testing [5], these have already demonstrated that evolutionary tests are likely to achieve better results testing the temporal behavior of complex systems compared to systematic tests.

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

Table 1: Maximum execution times of motor control tasks determined by evolutionary and systematic testing

With regard to the motor control system execution times determined with evolutionary testing do not exceed the temporal constraints of a task in any of the cases. The irregularities of the test results therefore have not been disquieting. The intensive testing of the temporal behavior with systematic and evolutionary tests has strengthened the developers' confidence in a correct temporal behavior of the system.

4 Concluding Remarks and Future Work

Temporal correctness is crucial to the flawless functioning of real-time systems. Testing is the most important analytical method to assure the quality of real-time systems. Our work investigated the use of extended evolutionary algorithms to validate the temporal correctness of embedded systems. The illustrated extensions to evolutionary testing allow the combination of multiple strategies, e.g. global and local searches, and the automatic distribution of resources in accordance with the success of the strategies.

Even in comparison with developer tests good results were achieved in the experiments performed. Thus, 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. However, more work is still needed to find the most appropriate and robust optimization parameters for evolutionary testing, so that the extreme execution times can be detected efficiently and with a high degree of certainty in practical operation.

Since no search strategy can guarantee that extreme execution times will be found, the use of evolutionary testing alone is not sufficient for a thorough and comprehensive test of real-time systems. A combination with static analysis techniques and existing test methods is necessary. An efficient testing strategy for real-time systems should at the very least include systematic testing and evolutionary testing [17]. If instead of a randomly generated population, the evolutionary test used 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 and structure. In order to automate this test strategy an integration of the evolutionary test into the test system TESSY [16] is being developed. This will fully automate the test of the temporal behavior because TESSY automatically detects the interface of the test object and generates a test driver for the test execution.

In future, it is also intended to examine the combination with static analyses more closely [10]. 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 time. This means, developers of real-time systems would gain an efficient tool, to rate exactly the minimum and maximum execution times for their systems.

References

[1] *Baker, J.E.*: Reducing Bias and Inefficiency in the Selection Algorithm. Proceedings of the Second International Conference on Genetic Algorithms and their Application, Cambridge, USA, 1987.

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

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

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

[8] *Mathworks, The*: Matlab - UserGuide. Natick, Mass.: The Mathworks, Inc., 1994-1999. http://www.mathworks.com/

[9] *Mitchell, M.*: An Introduction to Genetic Algorithms. Cambridge, Massachusetts: MIT Press, 1996.

[10] *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.

[11] *Pohlheim, H.*: Genetic and Evolutionary Algorithm Toolbox for use with Matlab - Documentation. Technical Report, http://www.geatbx.com/, 1994-1999.

[12] *Pohlheim, H.*: Entwicklung und systemtechnische Anwendung Evolutionärer Algorithmen. Aachen, Germany: Shaker Verlag, 1998. (Development and Engineering Application of Evolutionary Algorithms. Ph.D. thesis, in german)

[13] *Puschner, P., and Nossal, R.*: Testing the Results of Static Worst-Case Execution-Time Analysis. ???, pp. 134-143, 1998.

[14] *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.

[15] *Törngren, M.*: Fundamentals of Implementing Real-Time Control Applications in Distributed Computer Systems. In Wikander, J., and Svensson, B. (Eds): Real-Time Systems in Mechatronic Applications, Kluwer Academic Publishers, Boston, USA, 1998.

[16] *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.

[17] *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.

[18] *Whitley, D.*: The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, California, USA, pp. 116-121, 1989.

[19] *Wegener, J., Grimm, K., Grochtmann, M., Sthamer, H., and Jones, B.*: Systematic Testing of Real-Time Systems. Proceedings of the Fourth European International Conference on Software Testing, Analysis & Review EuroSTAR '96, 1996.

[20] *Wegener, J., Sthamer, H., Jones, B., and Eyres, D.*: Testing Real-time Systems using Genetic Algorithms. Software Quality Journal, Volume 6, Number 2, June 1997, Chapman & Hall.

[21] *Mühlenbein, H. and Schlierkamp-Voosen, D.*: Predictive Models for the Breeder Genetic Algorithm: I. Continuous Parameter Optimization. Evolutionary Computation. Vol. 1, No. 1, pp. 25-49, 1993.

[22] *Rational Software Corporation*: Quantify. Rational Software Corporation, 1997.
http://www.rational.com/products/quantify/