*EuroSTAR '96*

# *Systematic Testing of Real-Time Systems*

*Joachim Wegener, Klaus Grimm, Matthias Grochtmann*
*and*
*Harmen Sthamer, Bryan Jones*

*Daimler-Benz AG*
*Research and Technology*
*Alt-Moabit 96 a*
*D-10559 Berlin*
*Germany*

*Tel.: +49 (0) 30 39982-232/-226/-229*
*FAX: +49 (0) 30 39982-107*
*Email: wegener@DBresearch-berlin.de*
*grimm@DBresearch-berlin.de*
*grochtm@DBresearch-berlin.de*

*Department of Computer Studies*
*University of Glamorgan*
*Pontypridd,*
*CF37 1DL,*
*UK*

*+ 44 (0) 1443 48-2730*
*+ 44 (0) 1443 48-2715*
*hsthamer@glam.ac.uk*
*bfjones@glam.ac.uk*

## *Abstract*

*The development of embedded systems is an essential industrial activity whose importance is increasing. Commonly, embedded software systems have to fulfil real-time requirements. The most important analytical method to assure the quality of real-time systems is dynamic testing. Testing is the only method which examines the actual run-time behaviour of embedded software systems, based on an execution in the real application environment. Dynamic aspects like the duration of computations, the memory actually needed, or the synchronisation of parallel processes are of major importance for the correct function of real-time systems, and have to be tested.*

*The most important prerequisite for a thorough test is the design of relevant test cases. The classification-tree method was developed to overcome the shortcomings of current functional testing practice. It turns functional test case design into a process comprising several structured and systematised parts - making it easy to handle, to understand, and also to document.*

*Nevertheless, the classification-tree method alone is not sufficient for a comprehensive test of embedded systems. It is not well-suited for an examination of temporal correctness which is additionally essential to real-time systems. Already very small systems show a large variety of different execution times. Therefore, a new approach - which concentrates on the temporal system behaviour - was developed to complement the systematic test with the classification-tree method. It is based on the use of genetic algorithms and has been successfully tried out. The idea is to use genetic algorithms to search for input situations with very long or very short execution times, and to check whether they produce a temporal error. If all the times found correspond with the timing constraints, confidence in the temporal correctness is substantiated. Otherwise the real-time system has failed and an error is detected.*

## Introduction

*A large number of industrial products is based on the use of embedded computer systems. The importance of software in such systems is permanently increasing due to the need of a higher system flexibility. Usually, embedded computer systems have to fulfil real-time requirements. A faultless function of the systems does not depend only on their logical correctness but also on their temporal correctness. Dynamic aspects like the duration of computations, the memory actually needed during program execution, or the synchronisation of parallel processes are of major importance for the correct function of real-time systems.*

*Consequently, dynamic testing is the most important analytical method for the quality assurance of embedded computer systems, since it is the only method which allows a thorough examination of the actual run-time behaviour of such systems in their application environment. Therefore, it has to be decided whether or not the system is in compliance with the system specification and the timing constraints.*

*Testing is one of the most complex and time-consuming activities within the development of real-time systems [Heath, 1991]. Often more than 50 % of the overall development budget is spent on testing. Testing embedded systems is much more difficult than it is for conventional software systems. The examination of additional requirements like timeliness, simultaneity, and predictability make the test costly. In addition, technical characteristics complicate the test, for example, the development in host-target environments, the frequent use of parallelism, distribution and mechanisms for fault tolerance, the strong connection with the system environment, as well as the utilisation of simulators.*

*This paper suggests a test strategy for the examination of embedded systems with regard to functional and temporal correctness. The idea is to perform a functional test with the classification-tree method [Grochtmann and Grimm, 1993], and afterwards to complement it with a test of the temporal program behaviour. This test is based on the use of genetic algorithms.*

*The first section contains a brief description of our approach. The second section describes test case determination by means of the classification-tree method. The next two sections explain the basics of genetic algorithms and their application to testing temporal system behaviour. The fifth section presents our test strategy for real-time systems. After some concluding remarks the paper closes with a short outlook on future work.*

## 1. Our Approach

*The systematic test is an inevitable part of the verification and validation process for embedded computer systems. The most important prerequisite for a thorough test is the design of relevant test cases, since they determine the kind and scope and hence the quality of the test. A test case defines a certain situation to be tested; it comprises a set of input data. A test case abstracts from a concrete test datum and defines it, only in so far as it is required for the intended test.*

*The functional test is the most important approach to test case design because it provides a system application oriented test. Only by means of test cases derived from the system specification it can be* *found out if specified requirements or functions have been omitted (e.g. simply forgotten) during the subsequent activities of the software development process. Most approaches to functional testing are based on the idea of partition testing [Jeng and Weyuker, 1989]. During partition testing the input domain of the system under test is divided into subsets according to various criteria and then one test datum is selected from each of these subsets. Popular approaches are equivalence partitioning, cause-effect graphing [Myers, 1979] and category-partition testing [Ostrand and Balcer, 1988]. Shortcomings of these approaches are that the test case design procedure is unsystematic or the handling of the method is awkward and complicated. To overcome these shortcomings, the classification-tree method was developed. It turns functional test case design into a process comprising several structured and systematised steps - making it easy to handle, to understand, and also to document. The classification-tree method is described in the following section. It is widely used within the Daimler-Benz Group and has been applied successfully to a large variety of different software-based systems, for example business applications, as well as applications from aerospace and automotive electronics.*

*The application of the classification-tree method to embedded computer systems revealed areas for further improvement of the test case design process. An investigation within Daimler-Benz Aerospace divisions showed three areas where additional research work is necessary:*

- *testing the temporal behaviour of real-time systems,*
- *testing systems with different sequences of input data in special consideration of their history of usage and with respect to their current data state, and*
- *white-box criteria suitable to measure the coverage of event-triggered, parallel, and distributed systems.*

*This paper deals with testing of the temporal behaviour of real-time systems. Current research work also focusses on methodological improvements for testing with sequences of test data. Further investigations about structural testing of complex systems are planned for the future.*

*Our approach for testing real-time systems thoroughly comprises a functional test with the classification-tree method and also a test of the temporal system behaviour. At first, the functional test is performed to ensure the logical correctness of the test object. Afterwards, the test cases, determined during this test, are used as basis for a genetic search for the shortest and longest execution times of the system under test in order to find temporal errors. In general, structural test methods are an important complement to functional testing [Grimm, 1996]. However, it has to be considered that an instrumentation of the test object causes probe effects - the run-time behaviour of the real-time system changes.*

## 2. Classification-Tree Method

*The classification-tree method uses partly and improves ideas from the category-partition method defined by Ostrand and Balcer [1988]. The basic idea is to partition separately the input domain of the test object under different aspects, assessed as relevant for the test, and then to recombine the different partitions to form test cases.*

*First, the tester is required to determine various aspects relevant to the test. Each aspect should allow a narrow, and thus clear, differentiation of the possible inputs of the test object. The next step is to* *partition the input domain under each aspect. Each partition is a classification in the mathematical sense, i.e. the input domain of the test object is divided completely and disjointly into subsets, the so-called classes. The partition should be complete in order not to miss some parts of the set to be examined from the beginning; and it should be disjoint in order to achieve a precise differentiation according to the aspect under test.*

*It is often appropriate to establish classifications which do not partition the entire input domain, but only one class of another classification. By means of restricting the classifications to individual classes, potential logical incompatibilities can be prevented. The recursive application of classifications to classes results in a tree of classifications and classes.*

*Test cases are then obtained by combining classes of different classifications, thereby selecting exactly one class from each classification. Therefore, a test case is the set achieved by constructing the intersection of the respective selected classes. Logical compatibility must be observed for the combination of classes, i.e. the intersection must not be empty. The tester selects as many test cases as needed for a thorough test. This means that the test cases have to cover all aspects, both individually and in combination.*

*The method provides a descriptive graphical representation of the stepwise refinement of the partition in the form of a so-called classification tree. To determine the test cases, this tree is subsequently used as the head of a combination table in which combinations of classes are marked.*

*The number of test cases defined during a classification-tree test depends on the combinations of classes chosen. An appropriate criterion for a minimal number of test cases is to require each class to be used in at least one test case. The number of test cases required to meet this minimal criterion can easily be computed from the classification tree. It is given by the maximal number of different classes belonging to one classification. A theoretical criterion for a maximal number of test cases is to require each possible combination of classes of different classifications as a test case. The number of test cases required to meet this maximal criterion can also be derived from the classification tree. However, this theoretical maximum does not include the case of class combinations which are logically incompatible or which cannot occur due to specified restrictions. To take into account these constraints the maximal criterion can be restricted to those combinations of classes which are actually possible.*

*It should be noted that the tester's experience and creativity are still needed at certain points in the classification-tree method. However, the method guides the tester and provides a structured and systematised approach to test case determination. It provides a clear and comprehensive documentation, and it is easy to use.*

*Example*

*The use of the classification-tree method will be explained with a simple example. The test object is a Computer Vision System which should determine the size of different objects (Figure 1). The possible inputs are various building blocks. Appropriate aspects in this particular case would be, for example, the size, colour and shape of a block.*

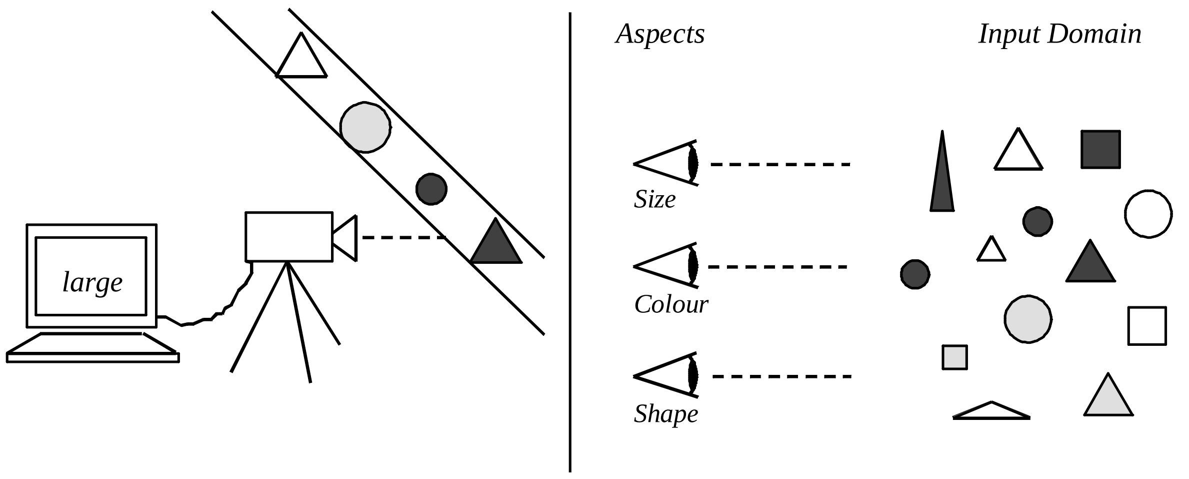

Figure 1: Computer Vision System with Aspects for Classification

*The classification based on the aspect colour leads, for example, to a partition of the input domain into red, green and blue blocks, the classification based on shape produces a partition into circular, triangular and square blocks. An additional aspect is introduced for the triangle class: the shape of triangle. It distinguishes equilateral, isosceles, and scalene triangles. The various classifications and classes are noted as classification tree (Figure 2).*

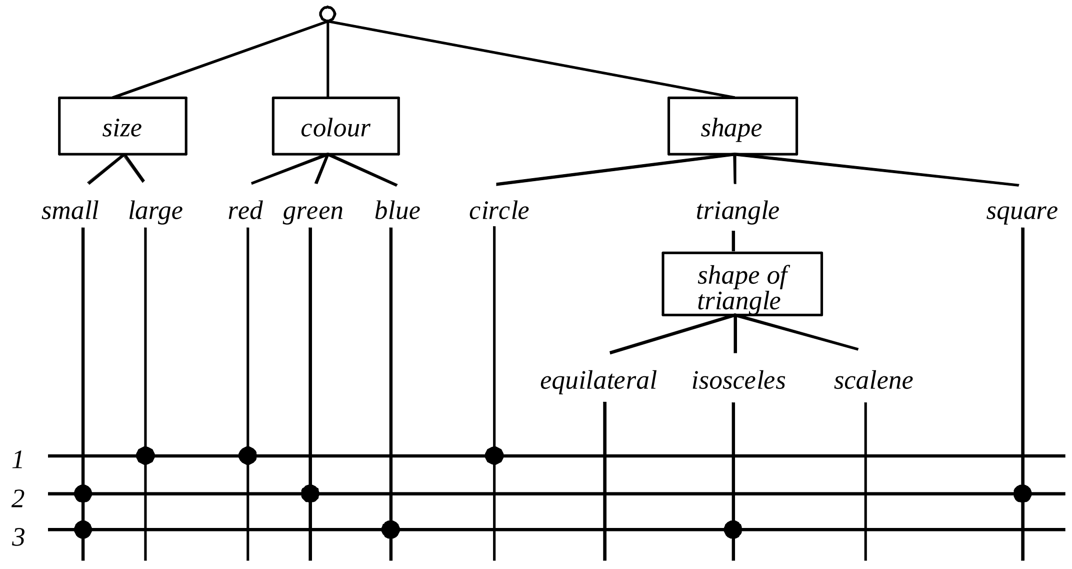

Figure 2: Classification Tree

*In the combination table associated with the tree, three possible test cases are given as examples. Test case three, for instance, describes the test with a small blue isosceles triangle.*

*A comprehensive description of the classification-tree method and related works were given by Grochtmann and Grimm [1993].*

*For the test of real-time systems aspects relevant to the temporal behaviour of systems may be introduced into the classification-tree, for example the complexity of input and output data, the complexity of functions or function sequences to be performed, or complex and time-consuming system states.*

## 3. Genetic Algorithms

*Many different optimisation techniques have been developed. However, these techniques often have problems with functions that are not continuous or differentiable everywhere, multi-modal (multiple peaks) and noisy functions. Therefore, more robust optimisation techniques are under development which may be capable of handling such problems. In the past biological and physical approaches have become of increasing interest to solve optimisation problems, including for the former neural networks, genetic algorithms and evolution strategies and for the second simulated annealing.*

*Genetic algorithms represent a class of adaptive search techniques and procedures based on the processes of natural genetics and Darwin's principle of the survival of the fittest. They model natural processes, such as selection, recombination, mutation, migration, locality, and neighbourhood [Pohlheim, 1996]. The fundamental concept of genetic algorithms is to evolve successive generations of increasingly better combinations of those parameters which significantly affect the overall performance of a design. The genetic algorithm achieves the optimum solution by the random exchange of information between increasingly fit samples and the introduction of a probability of independent random change. The strength of genetic algorithms is derived from their ability to exploit information in a highly efficient manner about a large number of individuals. The adaptation of the genetic algorithm is achieved by the natural selection and survive procedures, since these are based on fitness. The fitness-value expresses the performance of an individual with regard to the current optimum.*

*Genetic algorithms are iterative procedures that produce new populations of individuals at each step. A new population is created from an existing population by means of performance evaluation, selection, recombination and survival, see Figure 3. These processes repeat themselves until the population locates an optimum solution or some other stopping condition is reached. The selection of the starting generation has a significant effect on the performance of the next generation. Usually, it is generated randomly or heuristically.*

*Once the initial population has been created the evaluation phase begins. The genetic algorithms require that members of a population can be differentiated according to their fitness. The relative fitness of each individual is determined from a model of the system under test. An individual that is near an optimum solution gets a higher fitness value than an individual which is far away. The members that are fitter are given a higher probability of participating during the selection and reproduction phases and the others are more likely to be discarded. The fitness is the only feedback facility which maintains sufficient selective differences between competing individuals in a population.*

*In the selection phase individuals of the population may be chosen for the reproduction phase in several ways: for example, they may be chosen at random, or preference may be given to the fitter members.*

*During the reproduction phase, a complete new population of offspring is formed. Respectively, two members are chosen from the parent generation. The evolutionary process is then based on the genetic operators, for example crossover and mutation, which are applied to them to produce two new members for the next generation. The crossover operator couples the items of two parents to generate two offspring, which are created by swapping corresponding substrings of its parents. The*

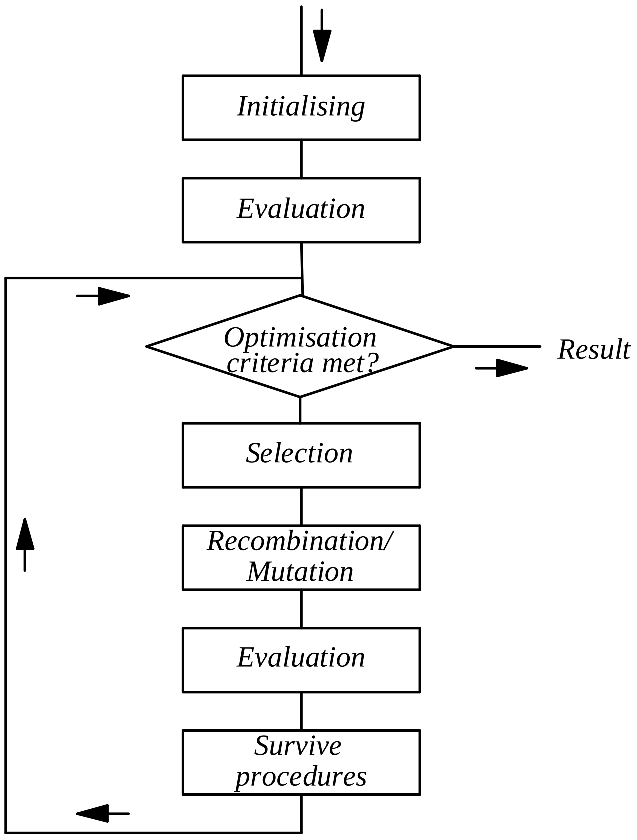

*Figure 3: Block Diagram of Genetic Algorithms*

*idea of crossover is to create better individuals by combining genetic material of fitter parents. The mutation operator alters one or more genetic cells of a selected individual with a low probability. This ensures a certain diversity in the population over long periods of time and prevents stagnation near a local optimum. A predefined survival strategy determines which of the parent and offspring survive.*

*The whole process is repeated generation by generation until a global optimum is found or some other stopping condition is reached [Sthamer, 1996].*

## 4. Testing Temporal System Behaviour with Genetic Algorithms

*The major objective of testing is to find errors. The temporal behaviour of real-time systems is defective when input situations cause the computation to violate 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 is, therefore, to find the input situations with the shortest or longest execution times (worst-case runtime) to check whether they produce a temporal error.*

*An investigation of existing software test methods showed that they mostly concentrate on testing for logical correctness. They are not suited for an examination of temporal correctness which is also essential to real-time systems. Nevertheless, in practice they are used to test the temporal behaviour since specialised methods are not available.*

*We conducted several experiments showing the complexity of testing the temporal system behaviour. A statistical test, for example, of a simple C-function with 4603 randomly generated sets of* *test data resulted in 298 different execution times from 5.27* msec *to 26.27* msec*. One program branch was not executed. Altogether, the function has only 16 branches and 30 statements and just uses the ‘.umul ’system call from the Solaris operating system. Figure 4 shows its control graph. A systematic test with the classification-tree method led to 49 test cases with 43 different execution times. Full branch coverage was achieved. These statistics illustrate the necessity of specialised approaches for testing the run-time behaviour of real-time systems.*

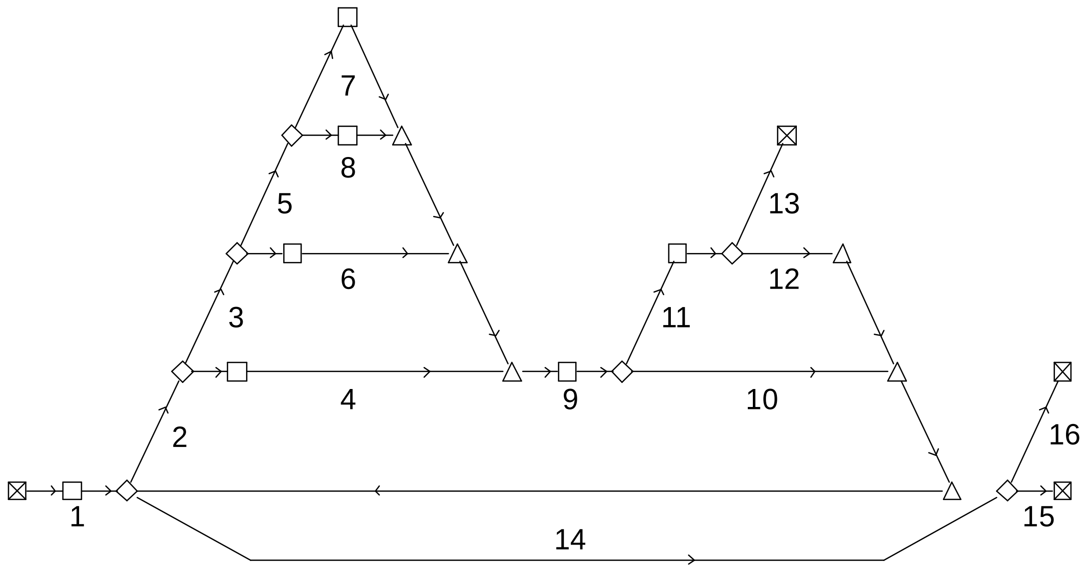

*Figure 4: Control Graph*

*When genetic algorithms are used to solve optimisation problems, good results are obtained surprisingly quickly [Sthamer, 1996]. Even for large, extremely non-linear, complex, and poorly understood search spaces genetic algorithms have been used successfully. In the context of software testing, the basic idea is to search the domain for input data which satisfy the goal of testing. Several papers exist which describe the application of genetic algorithms to structural testing, for example [Jones et al., 1995], [Roper, 1996], [Xanthakis et al., 1992].*

*However, our idea is to use genetic algorithms to find the shortest and longest execution times of a system. Populations of individuals, are generated applying the principle of survival of the fittest to produce better and better test data with regard to their fitnesses. The system is executed with the generated individuals. The execution time for each individual determines its fitness. For each generation, a new set of solutions is created by selecting individuals according to their level of fitness and breeding them together using operators borrowed from natural genetics like recombination and mutation. If all the times found meet the specified timing constraints for the system under test, confidence in the temporal correctness of the system is substantiated. Otherwise, an error is found.*

*Genetic algorithms seem to be an appropriate solution to this optimisation problem. They enable a totally automated closed-loop optimisation of execution times in sense of a search for the shortest and longest execution times (Figure 5). The execution time for each individual can be used as its fitness-value which is a very simple and easy way of assessment. Besides, genetic algorithms are particularly suited to problems involving large numbers of variables and complex input domains. They are even effective when the search or optimizing spaces are not smooth or continuous. Since genetic algorithms search from a population of points rather than from a single point the probability* *of being stuck at local optima is significantly reduced in comparison with more traditional optimisation strategies, like hill climbing. The use of mutation and subpopulations can further reduce the chance of being stuck. Furthermore, genetic algorithms are suitable for parallel processing so increasing the overall performance.*

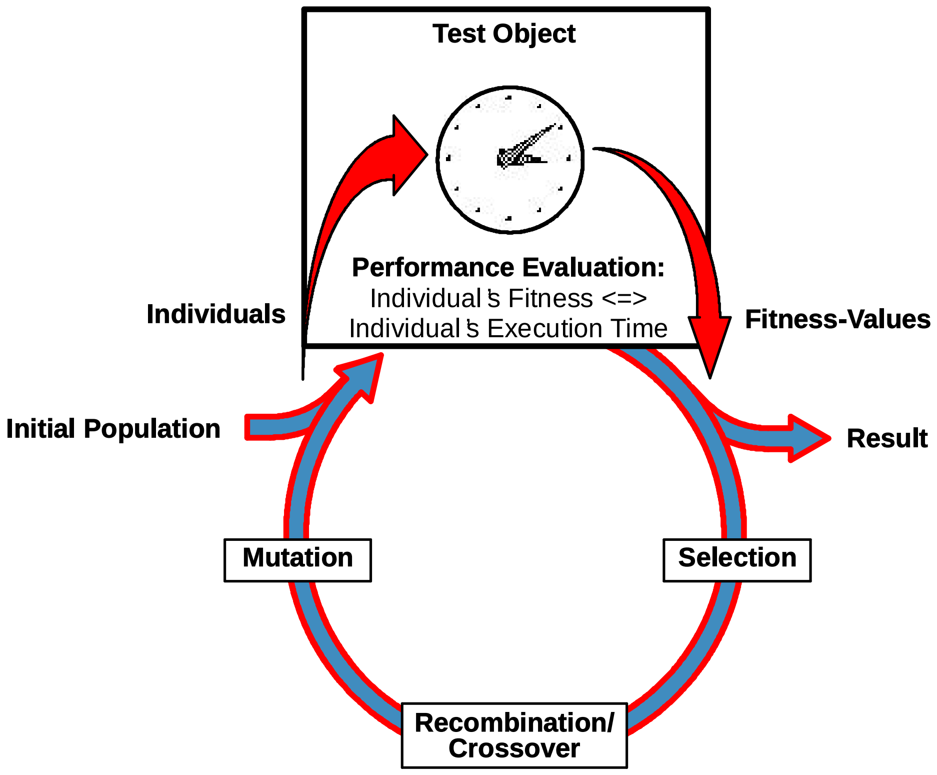

*Figure 5: Closed-Loop Optimisation of Execution Times*

*We used genetic algorithms in several experiments to determine the shortest and longest execution times of systems. The results achieved were promising. For the simple C-function already mentioned, the longest execution time (26.27 msec) was found very fast, in less than 20 generations. Regarding the shortest execution time the results were even more impressive. We found a new execution time (5.07 msec) which is shorter than the shortest time determined so far by statistical and systematic testing. For this, only 60 % of test runs performed for statistical testing was needed. The results seem to be plausible. Figure 6 shows the corresponding control-flows for the shortest and the longest execution time of our sample function. For the shortest execution time only the path 1-2-4-9-11-13 is executed, for the longest execution time all branches are executed except the empty ones numbered 10, 13, and 15.*

*Naturally, it is much more difficult to verify the results for large test objects with complicated control-flow graphs and complex temporal behaviour. In most experiments the genetic algorithms found shorter as well as longer execution times than by statistical testing. DeJong [1993] showed, that genetic algorithms rapidly locate the area in which a global optimum could be found, but need much more time to pinpoint it. Accordingly, it is difficult to ensure that the test data and so the execution times generated in our experiments, always represent the global optimum of the system under test. A detailed analysis of the assembler code and additional systematic tests are necessary to*

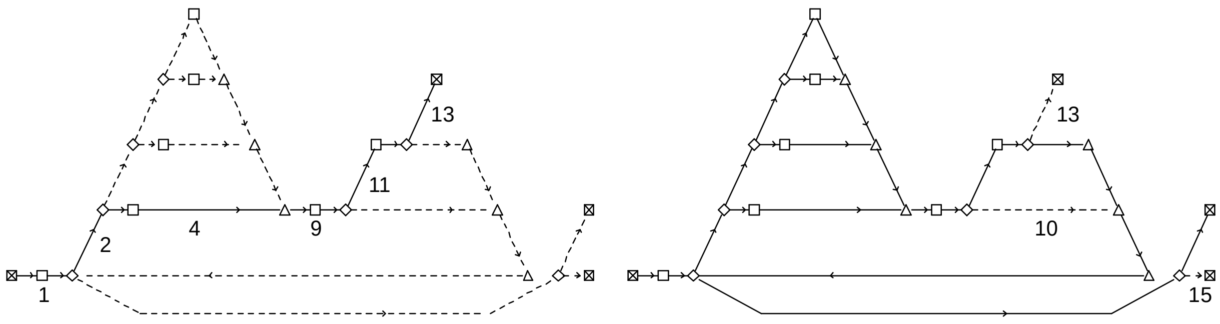

*Figure 6: Control Flow Graphs for the Shortest (5.07* msec) *and the Longest (26.27* msec) *Execution Time*

*verify this completely. Nevertheless, our results are promising and we intend to intensify our research in this area.*

## 5. Test Strategy for Real-Time Systems

*Since genetic algorithms try to achieve the optimum solution by the random exchange of information between increasingly fit samples (recombination) and the introduction of independent random change (mutation), they share a problem with random and statistical testing: it is not predictable if and when certain input situations will be found, which might be especially important for the runtime behaviour of the test object. On the other hand the existing approaches to systematic testing are not sufficient to examine the temporal behaviour of systems thoroughly. Therefore, an effective test strategy for embedded systems with real-time requirements should contain systematic testing as well as genetic optimisation.*

*We propose a combination of the classification-tree method with genetic algorithms (Figure 7). At first, the tester uses the classification-tree method for the systematic design of black-box test cases. The tester also adds aspects assessed as relevant to the temporal system behaviour, for example the complexity of input data or time-consuming system states. However, test cases determined with the classification-tree method mainly focus on the examination of logical correctness. Afterwards, the second step of our test strategy concentrates on the examination of temporal correctness. The test data specified for the systematic test is used as initial population for the optimisation of execution times by means of genetic algorithms - as described in chapter 4. Thus, the genetic search for the shortest and longest execution times benefits from the tester*'s *experience and his domain knowledge. The second step allows a detailed analysis of the temporal behaviour of the system under test.*

## 6. Conclusion and Future Work

*The classification-tree method was developed to overcome the shortcomings of the current functional testing practice. It was employed successfully on a wide range of real-world applications. Significant improvements concerning error detection and test efficiency have been observed. The use of the classification-tree method also for embedded computer systems revealed areas for further improvement. One area is the comprehensive test of the temporal behaviour of real-time systems.*

*An investigation of existing testing approaches showed a lack of methods suitable for an examination of temporal system behaviour. The temporal behaviour of real-time systems is unacceptable*

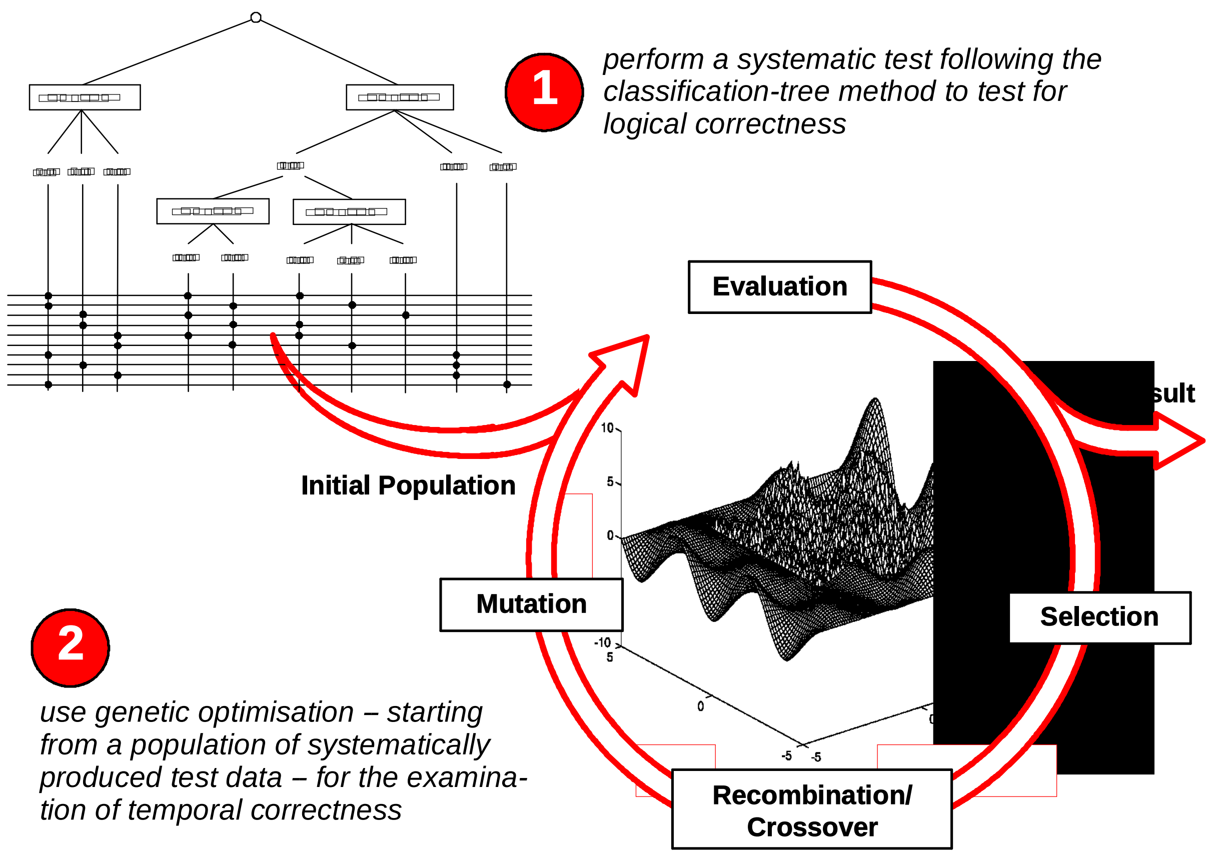
1
perform a systematic test following the classification-tree method to test for logical correctness
Evaluation
sult
Initial Population
Mutation
Selection
Recombination/ Crossover
2
use genetic optimisation – starting from a population of systematically produced test data – for the examination of temporal correctness
10
5
0
-10
5
0
-5
-5
0

Figure 7: Test Strategy Proposed for Real-Time Systems

*when inputs exist in such a manner that their computation violates the specified timing constraints. We propose to use genetic algorithms to find the shortest and longest execution times of a system. If they do not meet the specified timing constraints, an error is found.*

*First experiments achieved promising results. In comparison with statistical testing the genetic algorithms always obtained better results. The disadvantage of a random method which is that no step builds upon another is avoided by using genetic algorithms. Genetic algorithms take advantage of the old knowledge held in a parent population to generate new solutions with improved performance. Their iterations are based on the experience which has been gained from previous trials.*

*However, further research work has to be done to substantiate the general applicability of this approach and to consolidate its theoretical fundamentals. Many more experiments have to be performed to determine which genetic operators are best-suited to this optimisation problem. Furthermore, it could be possible to combine genetic algorithms with simulated annealing to increase the efficiency of the search. Another idea for further improvement is to combine our approach with structural testing. The fitness-function could be expanded in such a way that individuals which execute a new program branch get a high fitness-value to ensure their survival in the next generation. Thus the diversity of the population is not only maintained with respect to the temporal behaviour of individuals but also in consideration of the test object's internal structure.*

*Additionally, current research work concentrates on the development of extensions to the classification-tree method, to improve the method with regard to tests with sequences of data.*

## References

*DeJong, K.A. (1993). Genetic Algorithms are not Function Optimizers. In Whitley, L. (Ed.). Foundations of Genetic Algorithms, Morgan Kaufmann Publishers, Inc., California, pp. 5-17.*

*Grimm, K. (1996). Systematic Testing of Software-Based Systems. Proceedings of the 2nd Annual ENCRESS Conference, Paris, France, June 1996.*

*Grochtmann, M., and Grimm, K. (1993). Classification-Trees for Partition Testing. Journal of Software Testing, Verification and Reliability, Vol. 3, No. 2, pp. 63-82.*

*Heath, W.S. (1991). Real-Time Software Techniques. Van Nostrand Reinhold, New York.*

*Jeng, B., and Weyuker, E.J. (1989). Some Observations on Partition Testing. In Kemmerer, R.A. (Ed.). Proceedings of the ACM SIGSOFT '89 Third Symposium on Software Testing, Analysis, and Verification (TAV3), Key West, Florida, December 1989, pp. 38-47.*

*Jones, B.F., Sthamer, H.-H., Eyres, D.E. (1995). Genetic Algorithms: A New Way of Evolving Test Sets. Conference Papers of EuroSTAR '95, London, UK, pp. 24/1-24/10.*

*Myers, G.J. (1979). The Art of Software Testing. Wiley, New York.*

*Ostrand, T., and Balcer, M. (1988). The Category-Partition Method for Specifying and Generating Functional Tests. Communications of the ACM, 31 (6), pp. 676-686.*

*Pohlheim, H. (1996). GEATbx: Genetic and Evolutionary Algorithm Toolbox for Use with Matlab – Documentation. Technical Report, Technical University Ilmenau.*

*Roper, M. (1996). CAST with GAs – Automatic Test Data Generation via Evolutionary Computation. Computer Aided Software Testing (CAST) Tools, IEE Colloquium C6, Digest No. 96/096.*

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

*Xanthakis, S., Ellis, C., Skourlas, C., LeGall, A., Katsikas, S. (1992). Application of Genetic Algorithms to Software Testing. 5th International Conference on Software Engineering, Toulouse, France.*