Signal Generation for Search-Based Testing of Continuous Systems

Andreas Windisch
Technische Universität Berlin
Daimler Center for Automotive IT Innovations
Berlin, Germany
andreas.windisch@dcaiti.com

Noura Al Moubayed
Daimler AG
Research & Advanced Engineering
Software Technology, Specification and Test
Böblingen, Germany
noura.al_moubayed@daimler.com

Abstract

Test case generation constitutes a critical activity in software testing that is cost-intensive, time-consuming and error-prone when done manually. Hence, an automation of this process is required. One automation approach is search-based testing for which the task of generating test data is transformed into an optimization problem which is solved using metaheuristic search techniques. However, only little work has been done so far applying search-based testing techniques to systems that depend on continuous input signals.

This paper proposes two novel approaches to generating input signals from within search-based testing techniques for continuous systems. These approaches are then shown to be very effective when experimentally applied to the problem of approximating a set of realistic signals.

1 Introduction

The critical activity of testing is the systematic selection of test data since the accomplishment of an exhaustive test is typically infeasible in practice. Since the test data generation process is very cost-intensive, time-consuming and error-prone when done manually, the automation of this process is highly aspired. While various search-based automation approaches for certain testing objectives on the code level have been developed and proven to be of value in recent years [9, 5, 14], there have been only few efforts addressing the search-based test data generation for systems depending on input signals, i.e. sequences of data values [16, 2]. In order to apply the idea of search-based testing to the generation of input sequences for continuous systems, an efficient and robust way of generating and optimizing signal sequences is needed. Both white- and black-box testing techniques would be applicable for continuous systems with it.

This paper proposes two novel signal generation and optimization approaches and reports on our results from an empirical comparison when applied to the problem of approximating a set of six realistic signals. Approximating a signal and reaching a certain goal within search-based software testing are both based on fitness value feedback, thus being directly comparable. The results indicate that both approaches are, despite some minor issues, well-suited for signal generation and optimization when applied to software testing.

The paper is structured as follows: section 2 describes three signal generation approaches. One that is based on the calculation of trigonometric polynomials, one that is based on previous work making use of signal segments and one that extends the former by applying linear genetic programming. Section 3 includes the results of the experiments, and section 4 summarizes the paper and gives suggestions for future work.

2 Signal Generation Approaches

The creation of effective test scenarios for dynamic systems in general, i.e. the creation of system input signals by arrays of numbers, is hard to realize for real-world models as these often require the signals to have a specific minimum length (number of included time steps). In combination with a small sample rate, the resulting size of arrays representing the signals prevent its application to optimization engines. Accordingly, a discrimination between optimization and simulation sequences needs to be accomplished as shown in figure 1. The optimization sequence is used by an optimization engine and can be transformed into a simulation sequence which in turn is used as input for the execution of the model under test.

The following sub sections propose three approaches that are based on this principle. The first one uses a fourier series, which is known to be especially adequate for approximating arbitrary mathematical functions. This work

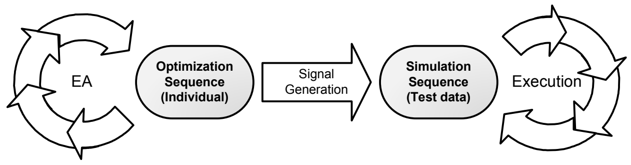

EA
Optimization Sequence (Individual)
Signal Generation
Simulation Sequence (Test data)
Execution

Figure 1. Signal representations split into optimization and simulation sequences

presents how it can be used for generating signal test data by applying an optimization technique. The second approach is based on previous work and uses a sequence of signal segments whose characteristics are specified by a number of parameters. These parameters are optimized by using genetic algorithms. In order to overcome a severe drawback of this signal generation approach we propose an extended version that makes use of the principles of linear genetic programming.

2.1 Fourier Series Approach

This approach is based on trigonometrical polynomials of variable degrees. Almost every mathematical function can be approximated by these finite linear combination of sine and cosine functions given a sufficiently large polynomial degree. Equation 1 shows the formula of a real trigonometric polynomial of degree k for each time step t.

$$T_k(t) = \frac{a_0}{2} + \sum_{n=1}^{k} \left(a_n \cdot \cos\left(n \cdot t\right) + b_n \cdot \sin\left(n \cdot t\right)\right) \quad (1)$$

Function $T_k(t)$ is used to calculate the signal values based on the parameters a_n and b_n. The user must specify the desired degree k which leads to a certain number of parameters: $2 \cdot k + 1$. In order to optimize this set of parameters, any advanced metaheuristic search technique can be used, such as for example evolutionary / genetic algorithms or particle swarm optimization.

2.1.1 Optimization and Simulation Sequences

Each signal is specified by the parameters a_n and b_n which constitute the optimization sequence. This set of parameters is optimized by the metaheuristic search engine.

The simulation sequence, which is the actual test data that is used for executing the system under test, is given by the signals that are calculated by equation 1 using the optimization sequence.

2.2 Segment Approach

The underlying idea of how to generate signals generally is based on the work of Baresel et al. [2]. It proposes a simple way of generating signals by stringing together a certain number of parameterized base signals. Therefore it makes use of further information about the input signals for the system under test that needs to be provided by the tester, who specifies the nature of the signals to be generated. This on the one hand includes general attributes, such as the length of the signals to be generated and their designated resolution. On the other hand it also contains separate specifications of the signals for each input of the system under test. It is thus possible to determine a list of base signals with which one particular input signal should be build up and to specify the signal amplitude boundaries to be kept. Figure 2 illustrates the general assembly of parameterized base signals (signal segments) into one composite signal. These signal segments can be specified using the three pa-

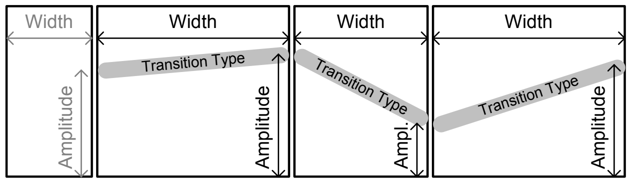


Figure 2. General signal assembly based on a certain number of parameterized base signals. Each base signal depends on three parameters: amplitude, transition and width.

rameters *Amplitude*, *Transition Type* and *Width*. The amplitude of each signal segment is supposed to be constrained to the signal boundaries specified for the appropriate input of the system under test. To be more precise, each signal segment depends on two amplitude values: one starting and one ending amplitude. The starting amplitude is taken from the amplitude value of the previous signal segment; the ending amplitude however depends on the parameterization of the signal segment itself. For this reason the first signal segment only provides the starting amplitude to the successive signal segment.

The transition type of the signal segments determines the way in which the starting amplitude is connected to the ending amplitude and thus highly influences the nature of the generated signal. Currently five transition types have been implemented which are described in section 2.2.1 in more detail. This approach allows for an easy extension of transition types if necessary.

Finally, the width of the signal segments is used to assure the variability of the generated signals. If the desired length of the final combined signal has been specified, the widths

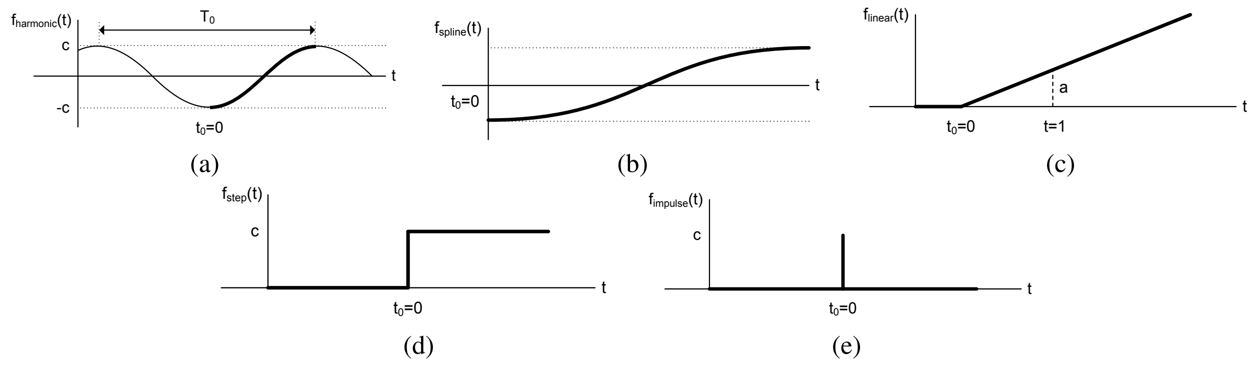


Figure 3. Five base signals used to assemble signal sequences: (a) harmonic signal, (b) spline signal, (c) linear signal, (d) step signal, (e) impulse signal

of all involved signal segments is scaled to the given overall signal length. Otherwise they are used absolutely, therefore leading to variable signal lengths.

2.2.1 Base Signals

This section describes some elementary signals, which are of utmost importance for control as well as systems engineering. Each of these elementary signals illustrated in figure 3 is used by the signal generator to build up signals, as described in section 2.2.2.

Harmonic Signal This signal segment is realized as a sine function whose first derivation - and thus its gradient - is zero at both the start and the end in order to ensure the monotony of the generated signals. This sine segment thus consists of a quarter sine wave. The harmonic signal can be used whenever a harmonic transition between two different amplitude values is accomplished.

Spline Signal The spline function is a polynomial of degree 3. Both periodic and non-periodic signal flows can be interpolated using splines. They thus allow for smooth gradients, i.e. no discontinuities occur up to the second derivation. The usability of spline segments conforms to the usability of sine segments.

Linear Signal The linear signal segment is very simple and intuitively important for expressing both ascending and descending signal flows. The frequently changing position of an acceleration or brake pedal of a car is one example where linear signal segments can be used to build up a simulation signal sequence.

Step Signal The step-like signal segment may be used wherever stepwise changes of the signal value appear.

Impulse Signal The impulse signal segment can be used whenever short signal spikes need to be generated. The pushing of a button or the triggering of a lever would, for example, be candidates for which impulses may be reasonable.

The impulse function as a signal segment is realized by holding the starting amplitude fixed until the last data value of the segment. The impulse is then released at this last data value by exhibiting the end amplitude of the segment. The following signal segment (used within the signal generation process) will proceed from the starting amplitude of the impulse segment.

2.2.2 Composition of Signal Segments

The signal generator builds up signals by lining up a certain number of signal segments. Each of these signal segments parameterizes an elementary signal (base signal); with these parameters given, the signal segments can be rendered and then joined together to one overall compound signal.

Figure 4 shows this lining up for a sample chromosome, i.e. an optimization sequence, consisting of nine signal segments, resulting in a signal which consists of eight rendered signal segments. The first signal segment goes from amplitude value 5.0 down to -4.0, using a step transition from where the second segment continues ending at amplitude value 2.0, using a sine transition and so on. The influence of the segment's widths can clearly be read from the resulting signal. The triplication of, for example, the width of segment 2 in relation to segment 3 within the optimization sequence also leads to a triplication of the associated signal segment width within the simulation sequence. In this example an overall signal was to be generated that exhibits a total length of 80 seconds.

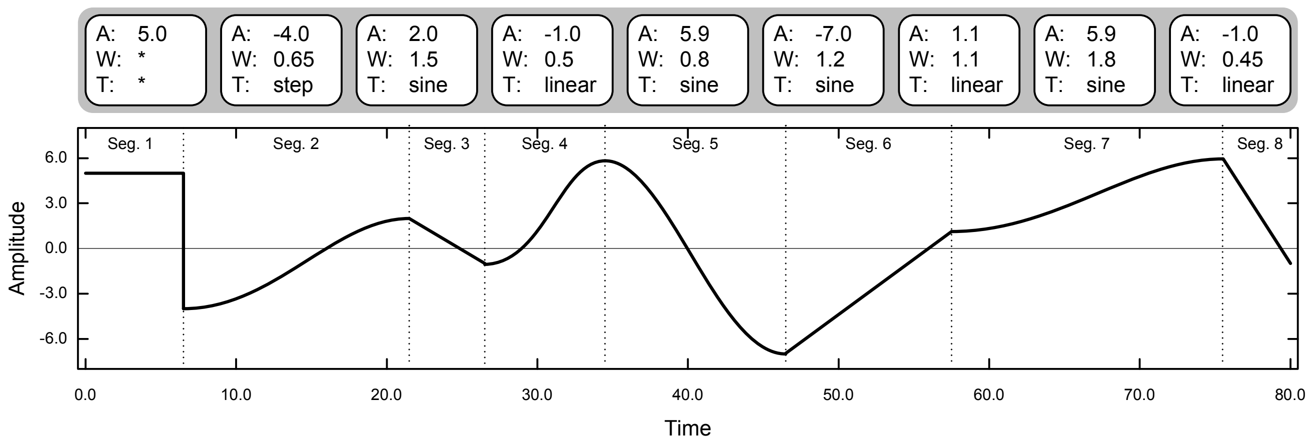


Figure 4. Construction of signals by lining up single signal segments. On top an examplary chromosome containing nine genes is shown whereas the signal hence generated is shown below.

2.2.3 Optimization and Simulation Sequence

Accordingly, a signal can be described using the parameters of the signal segments it is composed of, i.e. $3 \cdot n$ parameters with n being the number of signal segments. This number can be reduced if the user is able to provide knowledge about the signals to be generated. If for example only one transition type is allowed for one signal, the search engine does not need to optimize this parameter. As a result, the optimization sequence would reduce to $2 \cdot n$ parameters. The same applies to the width and the amplitude; either of them can be constrained to equal maximum and minimum values which leads to fixed lengths or fixed amplitudes of the signal segments, respectively. However, both cases separately would further reduce the length of the optimization sequence to n parameters, if this kind of knowledge is given.

In contrast, the simulation sequence, i.e. the test data, is obviously given by the signals that are generated by the underlying signal generator using the optimization sequences.

2.3 Linear Genetic Programming Approach

This approach extends the previously introduced segment approach by applying principles of linear genetic programming to automatically vary the signal segments in order to overcome the drawback of having to specify the number of signal segments.

2.3.1 Linear Genetic Programming

Genetic programming is a type of an evolutionary algorithm intended to evolve computer programs that fulfill a particular task. More precisely, its goal is the creation of optimal programs for calculating given input-output pairs. For doing so, genetic algorithms determine the optimal arrangement of function calls within a given structure. These structures can be tree structures or sequences, e.g. a series of assembler-commands, according to the given input and output. Whereas a candidate solution within traditional genetic algorithms consists of a fixed-size data structure, genetic programming allows generating and evolving data structures of variable size that represent program statements or computational operators, respectively. A tree representation is the most typically used data structure for genetic programming approaches. Therefore, a candidate solution is represented by a program tree that defines the sequence of execution by assembling multiple functions [13]. John Koza is a pioneer in this field, using genetic programming in the late 1980s for the creation of LISP2-programs encoded as simple tree structures [6].

In contrast, linear genetic programming uses linear data structures [1, 4, 3, 11, 7]. Instead of representing a candidate solution using a tree, they are represented using a linear list of instructions which more likely equates to the imperative nature of program execution present for common instruction-based languages. As can be seen in the next section the signal generation approach described in this paper is also based on linear data structures and thus is well-suited for the application of linear genetic programming.

2.3.2 Linear GP for Signal Generation

This section reports on the application of linear genetic programming to the general signal generation approach described in section 2.3.1 and with it describes the underlying data structure and the genetic operators applied.

Figure 5 illustrates the data structure the linear genetic

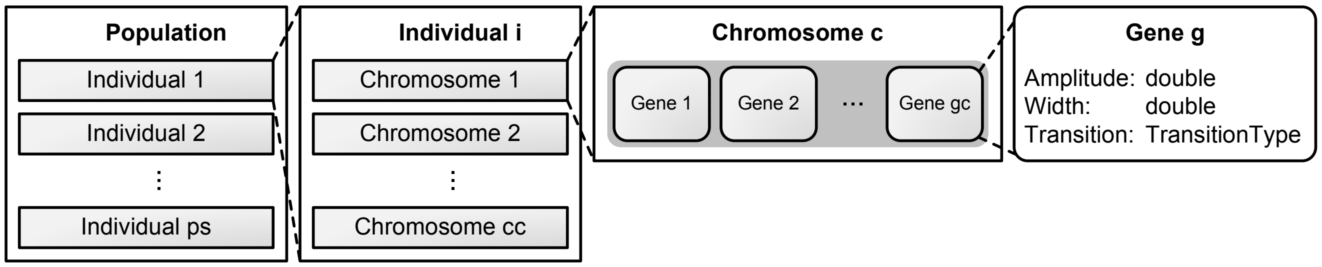


Figure 5. Encoding of the data structures used by linear genetic programming for signal generation. A population consists of ps individuals, each individual consists of cc chromosomes, which in turn consist of gc genes.

programming algorithm operates on directly. A population consists of a previously specified number of individuals. The number of individuals directly influences both effectiveness and efficiency of the metaheuristic search as follows. A bigger number of individuals within a population facilitates the diversity of the search und thus leads to a higher exploitation of the search space, eventually leading to an increase of its effectiveness. However, the efficiency of the search suffers at the same time because a larger number of individuals needs to be evaluated, some of which possibly cover similar areas of the search space.

Each *individual* is built up from a certain number of chromosomes, i.e. the particular representations of the inputs of the system under test. The number of chromosomes within each individual must be specified by the user, since this highly depends on the actual system under test.

A *chromosome* contains a certain number of genes. Each chromosome is the representation of valid signals for the associated input of the system under test. Hence, input-wide attributes, such as amplitude boundaries or the allowed transition types are set for the entire chromosome and thus apply to all genes. The chromosomes represent the linear data structures the linear genetic programming operators operate on.

Genes are representations of signal segments and accordingly feature three parameters: Amplitude, Transition Type and Width.

This data structure is evolved according to the principles of linear genetic programming using the genetic operators selection, crossover, mutation and reinsertion, implemented as described in the following sub section.

2.3.3 Genetic Operators

This section briefly describes the most important genetic operators, i.e. selection, crossover, mutation and reinsertion, which are used by genetic algorithms for evolving new generations.

Selection Selection accomplishes the assortment of individuals based on their fitness taking place at various points within the genetic algorithm. Selection for example is used for determining the individuals to be used to create the offsprings. The selection probability of an individual though depends on its fitness, though whereby the assortment of high quality potential solutions is favored. Several selection concepts exist, e.g. *roulette wheel selection*, *stochastic universal sampling* and *truncation selection*.

Crossover Crossover is the most important operator in genetic algorithms and is based on the metaphor of sexual reproduction. By exchanging and combining information, offspring is generated from at least two previously selected individuals. The *generation gap* thereby determines the number of offspring to be generated during one generation, which is given as the proportion of offspring relating to the overall population. Several crossover techniques exist which may be used depending on the data representation and the application area.

One of these techniques is homologous crossover, which combines the genome of two parental individuals [8, 10]. This technique is homologous in the sense that it only exchanges genes within a given range valid for both parents as illustrated in figure 6. Therefore the cut points are the same in both parents and the length of the offspring is the same as for the first parent. Another crossover technique is two-point crossover. For both parents two cut points are randomly determined and the genes within these boundaries are exchanged between both parents to form an offspring as illustrated in figure Since the cut points and thus the number of genes exchanged is different for both parents, the resulting offspring may also be longer or shorter than both of its parents. 7.

Referring to the data structure described in section 2.3.2, it is worth mentioning that for crossover of two individuals the crossover operator is performed on their chromosomes such that each chromosome of one parent is exclusively

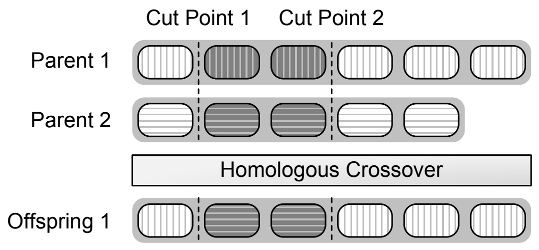


Figure 6. Homologous crossover: two cut points are randomly chosen within which the genes of both parents are exchanged to form an offspring.

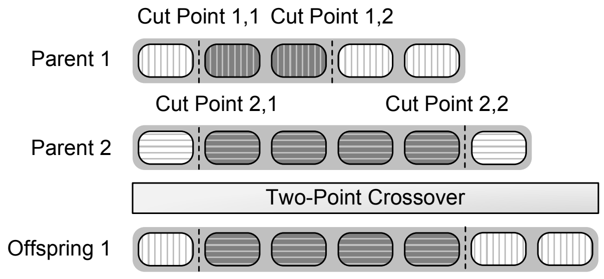


Figure 7. Two-point crossover: two cut points are randomly chosen for each parent within which their genes are exchanged to form an offspring.

combined with the respective chromosome of the other parent. This is necessary since each chromosome of an individual represents one particular input of the system under test and thus may exhibit e.g. different amplitude boundaries.

Mutation Mutation is usually performed with a low probability on newly generated individuals, i.e. they are slightly modified in order to prevent them from becoming too similar which helps maintaining the diversity of the entire population as well as counteracting premature convergence in local optima. The extend of the modifications is described as *mutation step* and its frequency as *mutation rate*.

For this approach, mutation is applied to the genes within the chromosomes of each individual as shown in figure 8. *Reducing mutation* randomly determines both the number of genes to be removed and the index from which they shall be removed, which leads to a length reduction of the mutant. For *multi point mutation*, a certain number of genes to be mutated is randomly chosen depending on the mutation rate. Mutation is carried out by adding or subtracting a small random value to or from these genes [12]. *Extending mutation* simply adds a randomly determined number of newly created genes to the end of the chromosome.

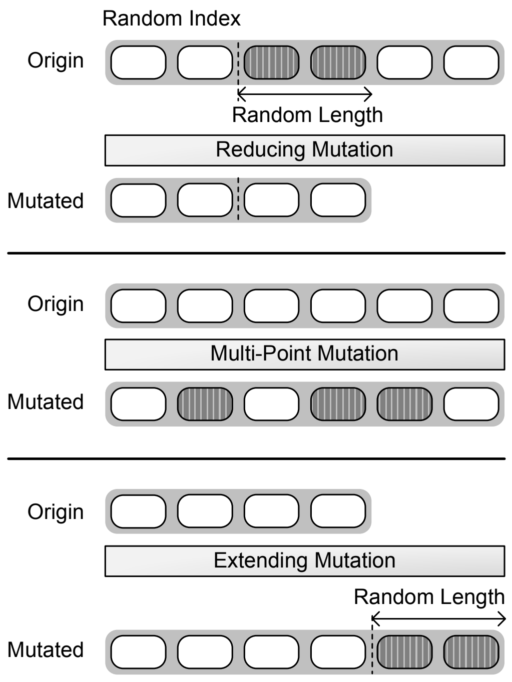


Figure 8. Three different mutation operators and their impact on example chromosomes.

The probability for each individual, each chromosome and each gene to be mutated depends on the attribute's *individual mutation rate* $P_{M_{Individual}}$, *chromosome mutation rate* $P_{M_{Chromosome}}$ and *gene mutation rate* $P_{M_{Gene}}$ respectively.

Reinsertion Reinsertion determines which individuals created by crossover and subsequent mutation are to be included in the primary population. The parameter *reinsertion rate* describes the maximum proportion of individuals of the original population to be replaced by offspring. Depending on both the generation gap and the reinsertion rate, different reinsertion strategies may be applied, such as *elitest reinsertion*, *pure reinsertion* or *union reinsertion* [12].

2.3.4 Optimization and Simulation Sequences

For this approach, the optimization sequence is supplied by the data structure the linear genetic programming engine directly operates on, i.e. the list of individuals which are themselves lists of chromosomes that are composed of various genes as illustrated in figure 5.

The simulation sequence again is generated from the underlying signal generator which interprets the optimization sequence in order to generate the signals that are directly used as inputs for the system under test.

3 Experiments

This section describes a small experimental series intending to reveal the ability of the proposed signal generation approaches to generating real-world signals and thus searching for suitable inputs for a system under test.

Applied to evolutionary testing, the underlying search technique, e.g. genetic algorithms or genetic programming, searches for signal sequences to be used as test stimuli for the system under test. This search is based on and guided by the applied objective function. By evaluating and assessing the generated inputs it directly influences the signal generation and optimization process. In other words it somehow defines the very signal which leads to a particular desired behavior of the system under test and calculates the difference to it for the respective input under evaluation. Hence, an easy and time-saving way of investigating the abilities of the presented approach to generating input signals capable of solving the problems of evolutionary testing is to use it for the approximation of a particular realistic signal.

The signals to be approximated here emanate from various divisions of Mercedes Benz Cars and thus represent industrial real-world examples of signals. They are described briefly in section 3.1, the results of the experimental approximation case study is presented in section 3.3.

3.1 Test Objects

Six realistic signal flows are available for this experimental approximation case study taken from the automotive domain. All these signals have been logged from the CAN bus directly and either show the output of certain sensors, such as a temperature sensor or similar internally calculated information sequences such as the distance to a preceding car calculated within an adaptive cruise control or the like. All signals have been recorded for a time period of 30 seconds and exhibit a sampling rate of 1 millisecond.

3.2 Experimental setup

For this experimental case study, signal generation and optimization is to be performed using the fourier series approach and the linear genetic programming approach described in sections 2.1 and 2.3 respectively. In addition, experiments using random data generation are to be carried out as a controlling baseline technique. This is realized by using the L-GP approach and repeatedly creating and evaluating initial populations. The segment approach is not used here since the only difference to the L-GP approach is the limitation of having to specify the number of signal segments in advance.

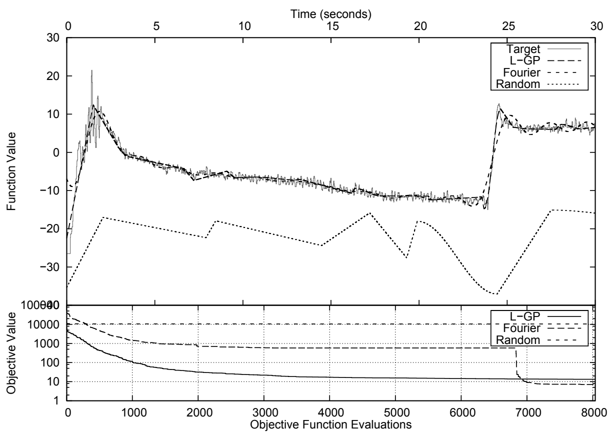
Particle Swarm Optimization has been used for the fourier series approach in order to optimize the optimization sequences. The configuration (gregarious particle swarm optimizer) used for this algorithm has been taken from Windisch [15], who already applied it to the task of structural testing successfully. We chose a degree of 15 for the fourier series, which leads to a search space with 31 dimensions. These parameters are initialized to values between -100 and 100. The optimization is carried out for 200 iterations, which is a total of 8.000 objective function evaluations.

Table 1 shows the settings used for the experiments that applied the linear genetic programming approach as signal generation technique. We configured the signal generator to use the sine, linear and step transitions using amplitude values between -150 and 150 since we pretend to have no knowledge about the signals, but still wanting to constrain the search space for reasons of efficiency.

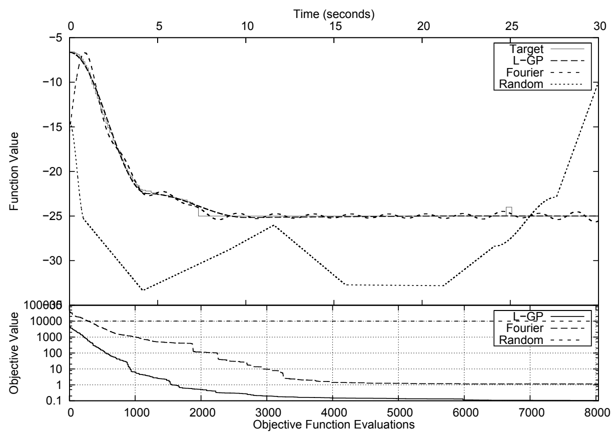
Initialization	Individuals	50
	Genes	$\in [60, 200]$
Selection	Type	Stochastic Universal Sampling, Tournament, pressure $\in [0.1, 0.5]$
	Generation Gap	1.0
Crossover	Type	Homologous, Two-Point
	Probability	0.75
Mutation	Type	Reducing (p=0.005), Multi-Point (p=0.495), Extending (p=0.005)
	$P_{M_{Individual}}$	0.95
	$P_{M_{Chromosome}}$	0.9
	$P_{M_{Gene}}$	0.05
	Range	0.1
	Precision	8
Reinsertion	Type	Elitest
	Rate	0.9
Termination	Evaluations	8.000
	Fitness	≤ 0

Table 1. Configuration of the linear genetic programming algorithm used for the experimental case studies.

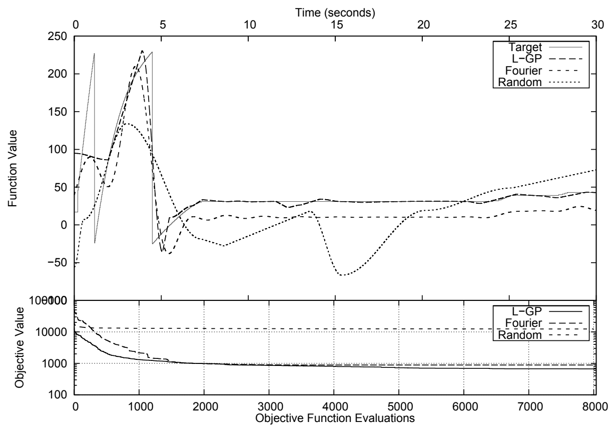
For each signal a total of ten complete optimization runs is supposed to be executed. The fitness of an individual is calculated as the mean squared error of the target signal and the signal generated from this individual. The results of the runs, i.e. the signals generated from the respective best individuals are stored for investigation and evaluation.



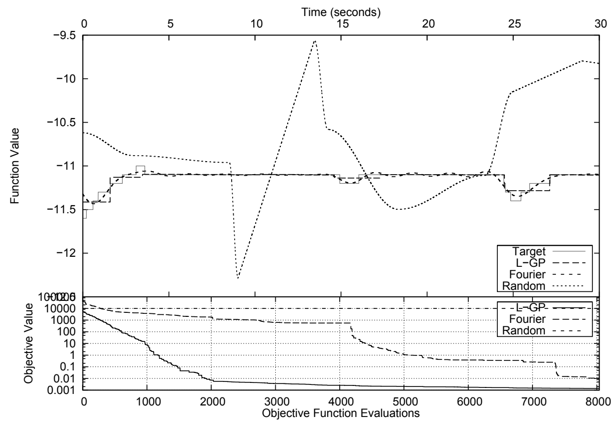
(a)



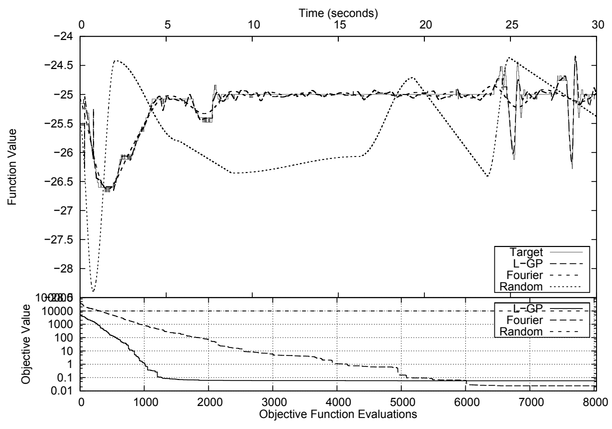
(b)



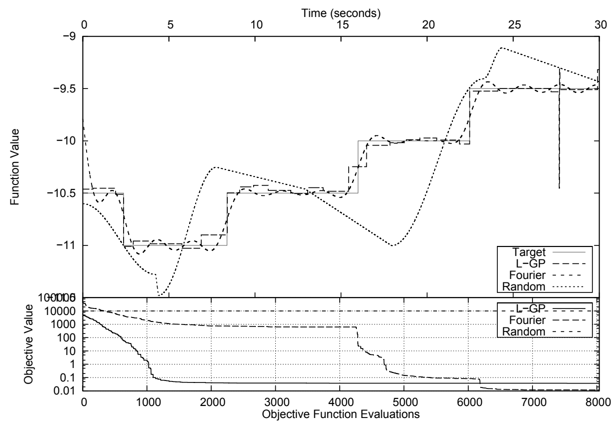
(c)



(d)



(e)



(f)

Figure 9. Results of the experimental case study aiming at approximating a total of six real-world signals using the proposed signal generation approaches based on fourier series, linear genetic programming and random data generation respectively.

3.3 Results

The results of the performed experiments are shown in figure 9. Each of these figures shows the target signal as a grey line, whereas the signals generated from the best individual constructed by the respective signal generators are drawn using black dashed lines. Additionally, a plot showing the objective value progress characteristics is given for each experiment.

Signal (a) is quite noisy and as expected both signal generators have not been able to generate a signal that reproduces a similar noisy behavior. In theory, the signal is reproducible by fourier series when using a sufficiently large degree k. The same applies to the L-GP approach when using linear segments given a sufficiently large number of signal segments. Since k has been set to 15 and the number of signal segments has been constrained to values in the range from 60 to 200, only a very rough approximation was possible and has been achieved by both approaches.

Signal (b) shows a relatively simple signal flow featuring a constant function value of 0 in 75% of its data values. In the beginning it contains two sine-like signal descends and near the end it exhibits a small impulse-like signal step. Overall, both signal generators were able to approximate the signal quite well except for the small signal step. Since the fitness function calculates the fitness depending on the entire signal, these kinds of small impulse-like steps do not carry much weight. Hence, it it is not surprising that the signal generators encounter difficulties. It is also worth mentioning that the fourier series approach was unable to generate a similar constant signal, for which a bigger degree k would have been necessary.

Signal (c), for the most part similar to the signal before, features a constant signal flow. The interesting part is located within the first five seconds, where a twofold rapid raising and lowering of the signal can be observed. As can be seen, both signal generators were unable to cover this first challenging part of the signal.

Both signal (d) and signal (f) possess a step-wise signal nature, their step size equaling 0.1 or 0.5 respectively. Although signal (d) looks very simple, the approximation was not as expected. The steep ascendancy within the first three seconds for example has not been approximated suitably by both approaches. The same problems were observed for the short steps in the center of the signal. Subject to the simplicity of signal (f), a fairly good approximation has been achieved by both approaches.

Finally, the difficulty for signal (e) is that of approximating both valleys at the beginning and both dynamic impulse-like signal changes at the end. Whereas L-GP was able to cover the valleys quite well, both techniques were unable to cover the highly dynamic parts of the signal as well as the constant signal flow in the center.

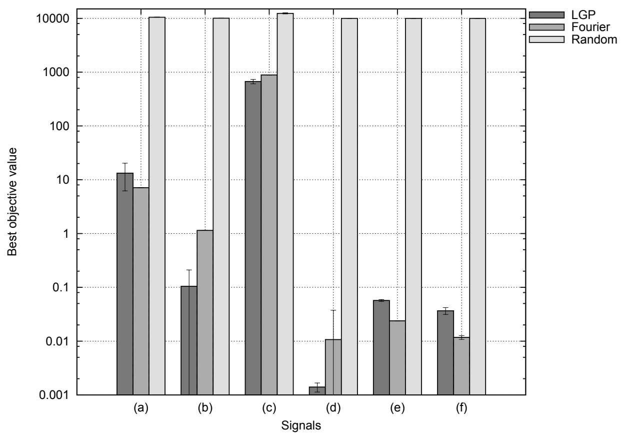


Figure 10. Experimental results for the effectiveness of the signal generation techniques regarding the industrial test objects

In addition to that, figure 10 shows the difference in effectiveness of both signal generation techniques in comparison to random data generation in terms of the best solution found during ten optimization runs. As can be seen both approaches outperform random testing by far. Compared to each other they give approximately equivalent results with a slight advantage for the linear genetic programming approach. In three cases Fourier found a better, i.e. a smaller, objective value, whereas in the remaining three cases L-GP was superior. In order to examine the statistical significance of the results concerning the comparison of Fourier and L-GP signal generation, a t-test was performed for the results of each experiment. The outcome of this test indicates that the majority of the differences in efficiency was statistically significant. Four of the differences (signals (b), (c), (e) and (f)) turned out to be very statistically significant (confidence 99%). The difference for signal (a) turned out to be significant (confidence 95%) and only the difference for signal (d) turned out to be not statistically sifnificant.

The experiments clearly point out that the proposed search-supported signal generation approaches were successful in approximating signals but still have difficulties with creating certain signal characteristics. Even though the fourier series approach was very successful in approximating signals, it can only be improved by raising the degree k, which unfortunately leads to a bigger search space hampering the chance of finding good solutions. The linear genetic programming approach was able to successfully approximate the majority of signals. Still, it could significantly be improved if knowledge about the system under test was available. If for example the signal amplitude boundaries are known, the algorithm would not have to explore search space areas that are invalid anyway. This ability of constraining the signal generation is the major advantage of the segment approach and thus of the linear genetic programming approach as well. Such constraint specifications are not possible for the fourier series approach.

4 Conclusion and Future Work

This paper reported on three novel approaches to generating and optimizing input signals for search-based testing. First we described the concepts of automating the signal generation process using these approaches. The developed system was then used to carry out approximation experiments with six industrial signals that exhibit different characteristics.

The results of the experiment show that the approaches are able to approximate given signals and thus are suitable to solving problems of evolutionary testing. However, it was also shown that the approximation of certain signals and particular signal characteristics may cause some minor problems.

However, more work still needs to be done in order for the approaches to be capable of reliably generating valuable signals. The effectiveness of this approach is highly dependent on the quality of the underlying optimization technique, thoroughly tuning the metaheuristic search engine used for optimization or improving and extending the linear genetic programming operators and tuning their associated parameters respectively is needed. Domain-specific signal segment transition types need to be developed in order to expand the segment and L-GP approach to being applicable in more areas. These areas could, for example, be in medical science where a signal transition featuring the characteristics of human nerve impulses would be imaginable to generate spike trains. Additionally, the incorporation of signal noise is another important extension.

5 Acknowledgment

This work was supported by EU grant 33472 (EvoTest).

References

[1] W. Banzhaf, F. D. Francone, R. E. Keller, and P. Nordin. *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[2] A. Baresel, H. Pohlheim, and S. Sadeghipour. Structural and functional sequence test of dynamic and state-based software with evolutionary algorithms. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 2428–2441, 2003.

[3] M. Brameier. *On Linear Genetic Programming*. PhD thesis, Fachbereich Informatik, Universität Dortmund, Germany, February 2004.

[4] M. I. Heywood and A. N. Zincir-Heywood. Dynamic page based crossover in linear genetic programming. *IEEE Transactions on Systems, Man, and Cybernetics: Part B - Cybernetics*, 32(3):380–388, June 2002.

[5] B. F. Jones, H. Sthamer, and D. E. Eyres. Automatic test data generation using genetic algorithms. *Software Engineering Journal*, 11(5):299–306, Sept. 1996.

[6] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press, December 1992.

[7] W. B. Langdon and W. Banzhaf. Repeated sequences in linear GP genomes. In *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*, Seattle, Washington, USA, 26 July 2004. AAAI.

[8] W. B. Langdon and W. Banzhaf. Repeated sequences in linear genetic programming genomes. In *Complex Systems*, pages 285–306, 2005.

[9] P. McMinn. Search-based software test data generation: A survey. *Software Testing, Verification and Reliability*, 14(2):105–156, 2004.

[10] P. Nordin, W. Banzhaf, and F. D. Francone. Efficient evolution of machine code for cisc architectures using blocks and homologous crossover. In *Advances in Genetic Programming III*, pages 275–299, Cambridge, MA, 1999. MIT Press.

[11] T. Perkis. Stack-based genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 148–153, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[12] H. Pohlheim. *Evolutionäre Algorithmen : Verfahren, Operatoren und Hinweise für die Praxis*. Springer, Berlin ; Heidelberg [u.a.], 2000.

[13] S. Wappler. *Automatic Generation Of Object-Oriented Unit Tests Using Genetic Programming*. PhD thesis, Technische Universität Berlin, 2007.

[14] J. Wegener, A. Baresel, and H. Sthamer. Evolutionary test environment for automatic structural testing. *Information and Software Technology*, 43(1):841–854, 2001.

[15] A. Windisch, S. Wappler, and J. Wegener. Applying particle swarm optimization to software testing. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1121–1128, 2007.

[16] Y. Zhan and J. A. Clark. Search based automatic test-data generation at an architectural level. In *GECCO (2)*, volume 3103 of *Lecture Notes in Computer Science*, pages 1413–1424. Springer, 2004.