Test-Goal-Specific Termination Criteria for Evolutionary White-Box Testing by Means of Software Measures

Frank Lammermann* Joachim Wegener*

* Daimler Chrysler AG, Research and Technology
Alt-Moabit 96a, 10559 Berlin, Germany
{frank.lammermann,joachim.wegener}@daimlerchrysler.com

1 Introduction

White-box testing is an important test procedure for the early detection of errors during software development. It defines, depending on the coverage criteria selected, test goals, such as, for example, requirements, branches or conditions, which need to be achieved for a complete white-box test. Here, test data generation plays a crucial role, because it defines error-prone test data. A successful approach to automated test data generation is *evolutionary white-box testing*. It is, however, difficult to define a suitable termination criterion for evolutionary white-box testing. This is because it is not known whether the program structure itself is not executable or whether the search for test data was not thorough enough if one of the program structures required by the sub-criterion is not achieved.

This paper researches the evolutionary white-box test [1, 2, 3] which has proved itself during numerous experiments. With its application it is possible to completely automate white-box test case generation. In the past, different papers have shown that evolutionary algorithms, compared to other optimisation procedures such as hill-climbing or random search, have proven to be more robust and are able to provide good results for all sorts of optimization tasks [4]. More simple heuristic methods, such as Simulated Annealing [5] are less suitable than evolutionary algorithms because of their local orientation and because they are not as powerful for the respective search space [1, 6, 7].

In this paper, a software measure will be introduced which estimates the test effort for every test goal of evolutionary white-box testing. With the aid of this software measure, it will be possible to individually adjust the termination criterion for every sub-goal. Experiments will show whether or not this increases the effectiveness of evolutionary white-box testing.

2 Definition of an Evolutionary Software Measure

The average number of test data generations is taken as a measure in order to quantify the test effort (E). This can easily be determined for every test goal and is of a sufficiently exact, but not

too detailed value range.

2.1 Relevant Attributes

Every software program can be reduced down to statements, conditions and loops [8]. The conditions alone, depending on the input parameters, affect the control flow, which is why we are able to limit the development of an evolutionary software measure to the attributes of conditions. Every condition has its own objective function, which shapes the search space. There are different approaches for the development of objective functions. Here, the approach developed in [9] was used.

Relevant condition attributes for the generation of the objective function in this approach are:

- **Solution density (SD):** the solution density is calculated from the quotient of the number of solutions divided by the size of the search space. Its value range is in interval [0, 1], whereby it assumes a value of 0 if the condition is unrealizable.
- **Operator type (OT):** conditions can have the same solution density using differently employed operators for their depiction. From the following depictions, different search spaces arise (figure 1):
  `(a < x + 1) && (a > x - 1)` ⇔ `(a == x)`

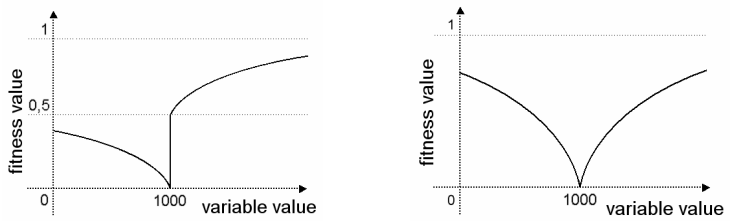

Figure 1: Search space structures with less than/greater than operators (left) and an equality operator (right) with an optimum at 1000.

- **Link type (LT):** when multiple basis predicates are engaged in fulfilling a general condition, their linkage can come about in different ways. Here, possibilities are *nesting* or *AND links* and *OR links*. Depending on the type of linkage of the basis predicates, different search spaces are formed (figure 2).

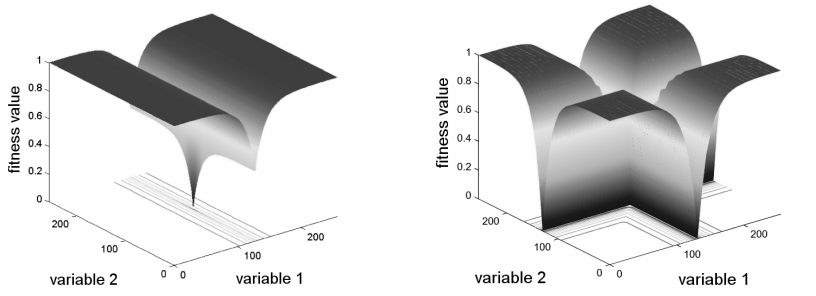

Figure 2: Search spaces from two AND- and two OR-linked equal conditions.

- **Link relationship (LR):** The respective attributes or, as the case may be, the respective test effort for a basis predicate engaged in linking, significantly influence the structure of the search space.

2.2 Measure Definition

Among the four attributes identified, two attributes solely influence the test effort for the basis predicates (SD, OT) and two others exclusively the test effort for the linkage of basis predicates (LT, LR). Since the values of the SD and the OT attribute do not interact, each of these attributes separately contributes in a certain proportion to the overall test effort for a basis predicate (E_{BP}). Thus, it is possible to calculate the test effort for a basis predicate by multiplicatively linking the test efforts for all the attribute values that influence the basis predicates. In the following, the term E_A refers to the average test effort as dependent upon the value of the attribute A.

Hypothesis 1: $E_{BP} = E_{SD} \cdot E_{OT}$

As evolutionary white-box testing is executed serially because of side effects [9], an increase in linkages can be gradually calculated, one after the other, according to the serial optimisation order. Because the number of possible attribute combinations of linked basis predicates is unlimited, we will restrict ourselves, in the case of linked conditions, to the test effort expressed by E_{BP}. Here, the value of the sub-conditions engaged in a link is negligible. By using the attributes LT and LR it is thus possible to estimate the test effort for binary links E_{BL} as follows:

Hypothesis 2: $E_{BL} = E_{LR}\,(LT, E_{BP\,2}, E_{BP\,1})$

If we expand the binary link to n randomly linked basis predicates, we are provided with the following recursive composition of the total test effort:

Hypothesis 3: $E_{total} = E_{LR}\,(LT, E_{BP\,n}, E_{LR}\,(LT, E_{BP\,n-1}, E_{LR}\,(\ldots, E_{BP\,2}, E_{BP\,1})\ldots))$

2.3 Measure Validation

All experiments performed in this section are based on the use of the framework for evolutionary white-box testing which was developed by DaimlerChrysler and is described in [2]. All the parameterisations of the evolutionary algorithm applied are described [10], whereby the number of test runs per test object has been set at 25. Overall, more than 1700 artificial test objects with different attribute values were examined.

All the results confirm hypothesis 3, as do parameterisations other than those described in [10]. Apart from the four attributes described, analysis of further attributes (e.g. the selection of the data type) showed no or negligibly small influence on the test effort.

3. Application of Evolutionary Software Measures

If one wants to calculate the test effort necessary for reaching a sub-goal, it is necessary to know all paths in the control flow graphs that lead from the starting node to the test goal. Along with

this, all the conditional statements on each of these paths must be combined by way of an AND-Link. If there are multiple possible paths by which a test goal can be reached, they can be combined with one another using an OR-Link, since each of these paths presents an independent possibility.

3.1 Initial Results

In order to evaluate the software measures on the basis of real test objects, 13 test objects were used, which were described in [10]. Table 1 shows the results for 15 test goals from nine test objects, each of which had a test effort larger than ten, and which seem especially suitable to be investigated because of their complexity. The test effort for the real measurement E_{meas} is between 40% and 163% of the predicted test effort $E_{predict}$. The standard deviation of E_{meas} from $E_{predict}$ is provided in the bottom row. The quotient of measured and predicted test effort averages out at 0.94.

Table 1: Comparison of measured and predicted test effort for nodes of real test objects.

Test object node	1-1	3-3	3-6	3-9	4-3	6-1	6-2	8-3	8-10	8-11	9-24	9-28	10-3	12-7	13-1
E_{meas}	10.8	21.0	23.3	87.8	26.8	42.6	38.0	50.7	33.0	32.0	16.4	63.2	45.6	35.0	82.8
$E_{predict}$	12.3	25.2	25.2	221	26.9	42.4	42.4	42.4	44.5	42.4	11.3	38.8	42.4	42.4	145
σ	9.3	9.4	9.2	135	10.0	4.5	4.7	10.9	13.0	12.3	9.5	33.4	9.5	6.7	73.9

3.2 Selecting a Test-Goal-Specific Termination Criterion

In the previous section, the standard deviation between the test effort predicted and the test effort measured was determined. The resulting confidence intervals can be calculated using the formula $\mu \leq \overline{x} + z_{1-\alpha} \cdot \frac{\sigma}{\sqrt{n}}$ [11], whereby n describes the number of random samples. The termination criterion (TC) is thus calculated from $TC = E_{predict} + \mu_{max}$ (figure 3). The following approximations for the termination criteria result via linear regression, according to the size of the confidence interval:

95%-TC = $-6.1+1.44 \cdot E_{predict}$, 98%-TC = $-7.7+1.56 \cdot E_{predict}$, 99.5%-TC = $-9.6+1.70 \cdot E_{predict}$.

The Bravais-Pearson correlation coefficient [11] reaches a value of r = 0.99.

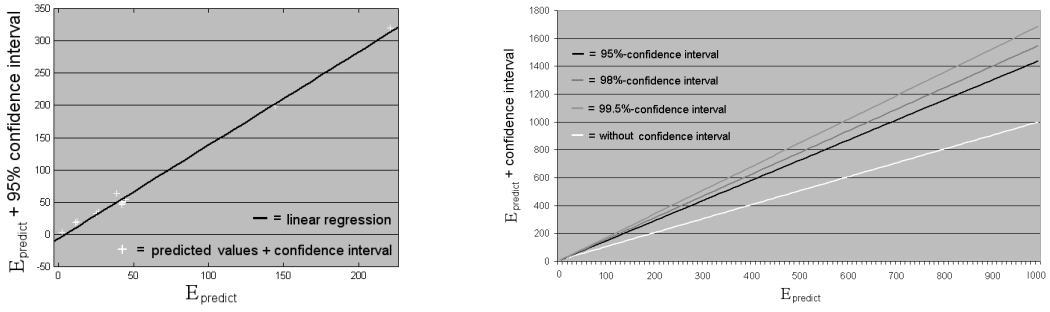

Figure 3: Confidence intervals for the termination criteria.

3.3 Results from complete applications

Without an evolutionary software measure, a uniform termination criterion was chosen for the evolutionary white-box test, which delivers a good trade-off between test effort and coverage [10] – for instance, for 200 generations of test data. When the test-goal-specific termination criterion is applied, in contrast, the tests are terminated if test effort exceeds the 95%-TC of a test goal. A maximum of more that 500 test data generations are not, however, carried out.

If we compare the results from [10] with those which are provided when using the evolutionary software measure, the values presented in table 2 result.

Table 2: Comparison of evolutionary white-box tests with and without an evolutionary software measure

	without evol. SW measure		with evol. SW measure	
Test object	Generations	Coverage	Generations	Coverage
1	2234	80.0%	1326	86.2%
2	448	94.4%	364	96.0%
3	308	90.0%	319	94.0%
4	228	80.0%	207	80.0%
5	6762	63.6%	2262	63.6%
6	79	100.0%	86	100.0%
7	818	80.0%	916	80.0%
8	70	100.0%	63	100.0%
9	3775	84.0%	4789	93.8%
10	47	100.0%	64	100.0%
11	863	95.5%	1726	95.5%
12	1635	73.3%	435	73.3%
13	8612	80.0%	5049	94.8%

Without the evolutionary software measure, an average coverage of 86.2 % for the 13 test objects investigated can be achieved; this figure is 89.0 % with an evolutionary software measure. With the help of an evolutionary software measure, the evolutionary white-box test requires only 94% of the number of test data generations.

4. Conclusion

When applying evolutionary white-box testing, it is difficult to decide when to terminate the test if a test goal has not been achieved. A uniform termination criterion for all the test goals has the disadvantage that test goals which are easier to achieve may be optimized for too long and that those which are more difficult to achieve may not be optimized for long enough. For this reason, this paper presents the development of an evolutionary software measure which is able to predict the test effort for individual test goals. It is based on four attributes of conditional

statements.

Initial results show that the predicted test effort for individual test goals corresponds well to real measurements. On the basis of the standard deviation which was measured, a confidence interval can be created which represents a suitable termination criterion for each test goal together with the predicted test effort. When the software measure is employed, the results improve from an original coverage of 86.2% to 89.0% coverage with only 94% test effort. This shows that, with the help of a termination criterion based on evolutionary software measures, the reliability, efficiency and effectiveness of evolutionary white-box testing can be increased.

References

[1] Pargas, R., Harrold, M., and Peck, R. (1999): "Test-Data Generation Using Genetic Algorithms". *Software Testing, Verification & Reliability*, vol. 9, no. 4, 263-282.

[2] Wegener, J., Baresel, A., and Sthamer, H. (2001): "Evolutionary Test Environment for Automatic Structural Testing". *Information and Software Technology*, vol. 43, 841-854.

[3] Baresel, A., Sthamer, H., and Schmidt, M. (2002): "Fitness Function Design to improve Evolutionary Structural Testing". Proceedings of *Generic and Evolutionary Computation Conference (GECCO 2002)*, New York, 1329-1336.

[4] Schwefel, H.-P. (1977): *Numerische Optimierung von Computer-Modellen mittels Evolutionsstrategie*. Birkhäuser Verlag, Germany.

[5] Kirkpatrick, S., Gelatt, C., and Veechi, M. (1983): "Optimization by Simulated Annealing". In: *Science*, 671-680.

[6] Wegener, J. und Grochtmann, M. (1998): "Verifying Timing Constraints of Real-Time Systems by means of Evolutionary Testing". *Real-Time Systems*, vol. 15, no. 3, S. 275 – 298

[7] Jones, B., Eyres, D., and Sthamer, H. (1998): "A Strategy for using Genetic Algorithms to Automate Branch and Fault-based Testing". *The Computer Journal*, vol. 41, no. 2, 98-107.

[8] Goldschlager, L. und Lister, A. (1988): *Computer Science: A Modern Introduction*. Prentice Hall, London, 2. Auflage.

[9] Jones, B., Sthamer, H., Yang, X. und Eyres, D. (1995): "The Automatic Generation of Software Test Data Sets using Adaptive Search Techniques". Proceedings of *3rd International Conference on Software Quality Management (SQM '95)*, Sevilla, Spain, 435 - 444.

[10] Lammermann, F., Baresel, A., and Wegener, J. (2004): "Evaluating Evolutionary Testability with Software Measurements". Proceedings of *Generic and Evolutionary Computation Conference (GECCO 2004)*, Part 2, Seattle, Washington, 1350-1362.

[11] Sachs, L. and Reynarowych, Z. (1984): *Applied Statistics: A Handbook of Techniques*. Springer-Verlag, Berlin, Germany.