Experiments on the Test Case Length in Specification Based Test Case Generation

Gordon Fraser*
Institute for Software Technology
Graz University of Technology
Inffeldgasse 16b/2
A-8010 Graz, Austria
fraser@ist.tugraz.at

Angelo Gargantini
Dip. di Ing. dell'Informazione e Metodi Mat.
University of Bergamo
Viale Marconi 5
24044 Dalmine, Italia
angelo.gargantini@unibg.it

Abstract

Many different techniques have been proposed to address the problem of automated test case generation, varying in a range of properties and resulting in very different test cases. In this paper we investigate the effects of the test case length on resulting test suites: Intuitively, longer test cases should serve to find more difficult faults but will reduce the number of test cases necessary to achieve the test objectives. On the other hand longer test cases have disadvantages such as higher computational costs and they are more difficult to interpret manually. Consequently, should one aim to generate many short test cases or fewer but longer test cases? We present the results of a set of experiments performed in a scenario of specification based testing for reactive systems. As expected, a long test case can achieve higher coverage and fault detecting capability than a short one, while giving preference to longer test cases in general can help reduce the size of test suites but can also have the opposite effect, for example, if minimization is applied.

1. Introduction

Software testing remains the most important technique in practice to find errors in programs and to gain confidence in software quality. Automation is desirable because testing is a very complex and error prone task. Many automated techniques to derive test cases have been presented in the past, differing on the underlying software artifacts, algorithms, and actual interpretations of what is a test case. In this paper we experimentally investigate the effects of the structure of test suites, and in particular try to determine whether it is preferable to use many short test cases or fewer long test cases.

The experiments are performed in a scenario of specification based testing for reactive systems. In such a setting, test cases are sequences of test data, and as test cases can be derived from a specification or model the expected output is known as well. In order to derive such test cases we use model checkers, and apply several different well known coverage criteria, mutation testing, and random testing. Using these ingredients we generate test suites differing on the number of test cases and their length, and analyze them with regard to several important aspects:

- How does the test case length influence the fault detecting capability? This question is addressed by coverage and mutation analysis.
- How does the test case length influence the computational costs of testing? To answer this question the effects of the test case length on the test suite size are investigated as well as the effects on redundant test cases, test suite minimization, and monitoring of coverage items during test case generation.

Of course, there is no definite preference between short and long test cases, as the choice will always depend on the requirements of the concrete testing scenario. However, as a consequence of the experiments we identify important insights that help deciding whether to give preference to short or long test cases.

This paper is organized as follows: Section 2 gives background information on the testing scenario we assume for our experiments and on the testing techniques we apply. Section 3 describes the experimental setup, i.e., the specifications, tools, techniques, and the actual experiments. Section 4 contains a representative selection of the data gath-

*The research herein is partially conducted within the competence network Softnet Austria (www.soft-net.at) and funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).

ered by these experiments, Section 5 discusses the results in detail, and the paper is concluded in Section 6.

2. Preliminaries

In this section we aim to clarify the testing scenario we assume for our experiments and present the necessary background information.

2.1. Testing reactive systems

Software testing differs very much depending on the type of system under test (SUT). In this paper, we assume that the SUT is a reactive system: A reactive system is a system that changes its actions and outputs in response to stimuli from within or outside. Such a system is usually tested by providing input values (test data) and comparing the resulting outputs with expected values (test oracle). Reactive systems can often be executed in an infinite loop, which means that we assume a test case can have any length and from any state there is a path to any other state (i.e., the underlying automaton is fully connected).

2.2. Testing with model checkers

Model checking [10] describes the process of determining whether an automaton model satisfies a specification given as temporal logic properties. In practice, one of the most useful features of model checkers is their ability to create counterexamples to illustrate how properties are violated. Such counterexample sequences can be interpreted as test cases under certain constraints; e.g., we assume that the system under test and its specification are deterministic.

To use model checkers for test case generation the test objective (e.g., satisfaction of a coverage criterion) is usually encoded as a set of temporal logic properties, such that for each distinct test requirement (e.g., coverage item) of the overall objective there is one property (*trap property* or *test predicate*) [16, 21]. Any counterexample to a trap property represents a test case that satisfies the test requirement posed by the property. Other test objectives include, for example, mutation testing [2, 15] or combinatorial testing [5]. A noteworthy advantage is that once a framework has been created it is very easy to apply any of these techniques or combine several at the same time. For a detailed overview of testing with model checkers we refer to [13].

Counterexamples are sequences of states; in this paper, we assume that a *test case* is also a sequence of states, and each state represents input and output values serving as test data and test oracle: For each state, test data is provided as input to the system under test and the returned outputs are compared to the expected output values to derive a verdict. The length of a test case is defined as the number of states it consists of. A *test suite* is a set of test cases, its size is the number of test cases it consists of, and its length is the sum of the lengths of its test cases.

2.3. Generating test cases of variable length

The traditional approach to test case generation using model checkers is to call the model checker once for each trap property. A call on a feasible trap property results in a counterexample starting in an initial state, while trap properties for infeasible test requirements simply do not result in counterexamples. The length of the counterexamples is determined by the underlying model checking technique. For example, explicit state model checking with breadth first search or bounded model checking will result in short test cases, while explicit state model checking with depth first search will result in longer test cases.

For the purposes of our experiments we need a way to influence the length of the test cases generated. Therefore, we use an approach similar to that proposed in [17]: After creating a test case for a trap property the final state of the test case serves as initial state for the next counterexample. That way, the next test case can be interpreted as an extension of the previous test case. Technically, this can be achieved by explicitly setting the model's initial state after each test case, or by rewriting the trap properties to an implication on the desired initial state. It is not possible to derive test cases of a precise length with this method, but to choose after each trap property whether to continue extending the current test case or to start a new test case.

3. Experimental Setup

While intuitively longer test cases are expected to have higher fault detecting capability and to be computationally more expensive, it is necessary to take into account that for a given test objective using long test cases will usually reduce the number of test cases in a test suite, as each individual test case is likely to cover more test requirements; under that light it so obvious what to prefer. Consequently, the experiments aim to answer the following research questions:

Research Question 1: Is it preferable with regard to fault detecting capability to have few test cases, each very long and covering many test requirements, or to have many short test cases, each covering fewer test requirements?

Research Question 2: Is it computationally more expensive to have few test cases, each very long and covering many test requirements, than to have many short test cases, each covering fewer test requirements?

Specifications

The experiments performed in this evaluation create large numbers of different test suites. In this paper, we present the results of experiments on two different specifications known in the testing literature: The Safety Injection System (SIS) specification models the control of coolant injection in a nuclear power plant. It was introduced in [4] and has since been used frequently for studying automated test case generation. The Cruise Control (Cruise) specification models a simple automotive cruise control. It is based on [19], and has also been used several times for automated test case generation, e.g., [2].

Testing techniques

There is a large number of different testing techniques to choose from, even when only considering those techniques suitable for test case generation with model checkers. The evaluation presented in this paper uses techniques where each test requirement (coverage item for coverage criteria or mutant in the case of mutation analysis) is encoded in temporal logic, such that a counterexample derived for the property is a test case for the underlying test requirement. To keep the number of results to a tractable number we have selected a representative subset of independent techniques, motivated by an evaluation [1] of specification based coverage criteria:

Mutation testing (e.g., [2, 15]) encodes small changes in the specification as trap properties, and resulting test cases can distinguish between original and mutant specification. *Transition pair* [1] coverage requires that all possible pairs of transitions of the formal specification are executed. *Modified Condition Decision Coverage* (MCDC) requires for each literal (condition) to be shown to independently affect the value of the expression (decision) it is part of. We use masking MCDC [6], and apply it to all expressions in the specifications. *Pairwise testing* [5] requires that all possible pairs of values for monitored variables are covered. Finally, *random testing* just uses random test cases generated with predefined length.

Tools

We use version 2.4.3 of the freely available model checker NuSMV [8], which supports symbolic and bounded model checking; for our experiments we used the symbolic model checker. NuSMV implements an algorithm for counterexample generation [9] which does not guarantee the shortest possible counterexamples, but still creates very short ones. In order to generate random test cases we use the interactive command line interface provided by NuSMV: At each state NuSMV picks one of the possible successor states with equal probability.

3.1. Experiments

The experiments that were performed for this evaluation are described below. Because the order in which trap properties are considered during test case generation can have an impact on the results [12], each of the experiments was repeated 10 times with different random ordering of the trap properties, and the results were averaged.

Experiment 1: For each of the specifications and test techniques test suites are generated according to the following procedure: For the first test suite a distinct test case is generated for each trap property. Then, for the second test suite one test case is generated for two trap properties, where the second trap property is used to extend the test case generated for the first trap property. Then this is done for every three trap properties, and so on, until at the very end of the spectrum a test suite consisting of only a single test case covers all trap properties. To reduce the number of test suites a little bit we only consider test suites that differ in the number of test cases, without any loss of generality. For each of the test suites the coverage and mutation score are measured.

Experiment 2: The second experiment uses the test suites generated in the first experiment, removes duplicate or subsumed test cases such that only unique test cases remain, and then minimizes the remaining test suite using a greedy minimization algorithm. A test case t is a duplicate, if there exists another test case t' consisting of exactly the same state sequence. A test case t is subsumed by another test case t', if t is a prefix of t'.

The aim of test suite minimization is to find a subset of the test cases that still fulfills the test objective. The motivation for minimization is that the costs of running a complete test suite against the software repeatedly can be quite high, but in general not all test cases of a test suite are necessary to fulfill some given test objective. A test suite is minimal [18] with regard to some objective if removing any test case from the test suite will lead to the objective no longer being satisfied. The problem of finding the optimal (minimal) subset is NP-hard, which can be shown by a reduction to the minimum set covering problem [14].

In this experiment, we use a simple greedy heuristic [7] to the minimum set covering problem for test suite minimization: The heuristic selects the test case that satisfies the most test requirements and remove all test requirements satisfied by that test case. This is repeated until all test requirements are satisfied.

Experiment 3: A single test case will usually cover more than one test requirement, and so it is not strictly necessary

Table 1. Numbers of trap properties.

Criterion	Cruise		SIS	
	Total	Feasible	Total	Feasible
Mutation	476	351	295	184
MCDC	88	75	60	54
Transition Pair	132	34	156	102
Pairwise	654	576	140	138

to generate test cases for all trap properties. If test case generation is computationally expensive it will be necessary to generate test cases only for those trap properties that are not already covered by other test cases; we say that the trap properties are *monitored* during test case generation (e.g., [11]). In many other settings this is also beneficial as it can greatly reduce the test suite size. As longer test cases are intuitively expected to satisfy more test requirements the question is what effects the test case length will have when monitoring is used. Note that monitoring and minimization can behave very differently: Minimization requires existing, full test suites while monitoring checks trap properties on the fly during test case generation. On the other hand, monitoring does not guarantee minimal test suites.

Similar to experiment 1, in this experiment we first create test suites where each test case is generated for one trap property; in contrast to experiment 1 only trap properties that are not already covered by a previous test case are considered. Then we create test suites where each test case is extended once, then twice, and so on until a single test case covers all trap properties.

Experiment 4: This experiment investigates how the fault detecting capability of individual test cases relates to their length. To do so the average mutation score per test case for the test suites created in experiment 1 is determined. In addition, we use randomly generated test cases of different length and measure their mutation score as well.

4. Results

Because we generated thousands of different test suites in the course of our experiments we can only present selected results that are useful for the discussion here.

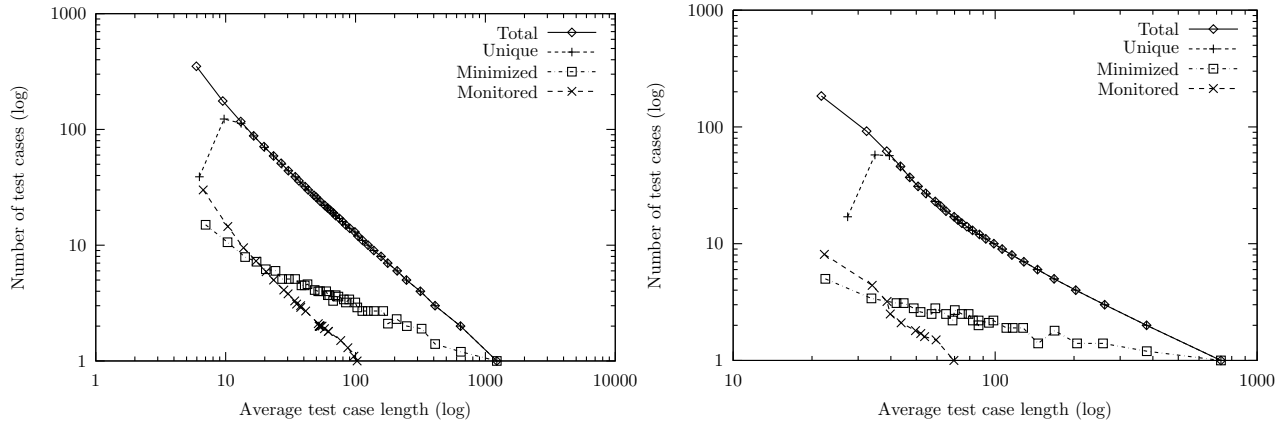
Table 1 lists some statistics about the specifications and test objectives used for the experiments. The trap properties were generated automatically from the NuSMV specification. Test requirements can be infeasible, which in the case of testing with model checkers simply results in a trap property being satisfied by the model. This means that the infeasible trap properties contribute to the costs of the test case generation, but not to the resulting test suites.

Figure 1 illustrates how the average test case length relates to the number of test cases in a test suite. 'Total' describes the total number of test cases generated (experiment 1), 'Unique' represents the number of unique test cases, i.e., the test cases that remain after removing duplicate and subsumed test cases (experiment 2). 'Minimized' represents the number of test cases after the greedy minimization algorithm is applied with regard to the coverage criterion that was used to generate the test cases (experiment 2). 'Monitored' represents the number of test cases using monitoring during test case generation (experiment 3). Figure 1 shows the results for mutation adequate test case generation in greater detail; the results with regard to all other criteria are similar and therefore not shown here.

Note that our test case generation method does not result in test suites with test cases of equal length. There is a relatively high standard deviation of the test case length for low numbers of trap properties per test case (not shown in the figures to avoid data overlapping; it is often greater than 10% for less than 3 trap properties per test case) but quickly decreases with increasing numbers of trap properties per test case. While the standard deviation is constant for the case of one trap property per test case (without monitoring) it can vary when using more trap properties per test case, therefore the results shown are averaged over 10 runs with different random orders of the trap properties.

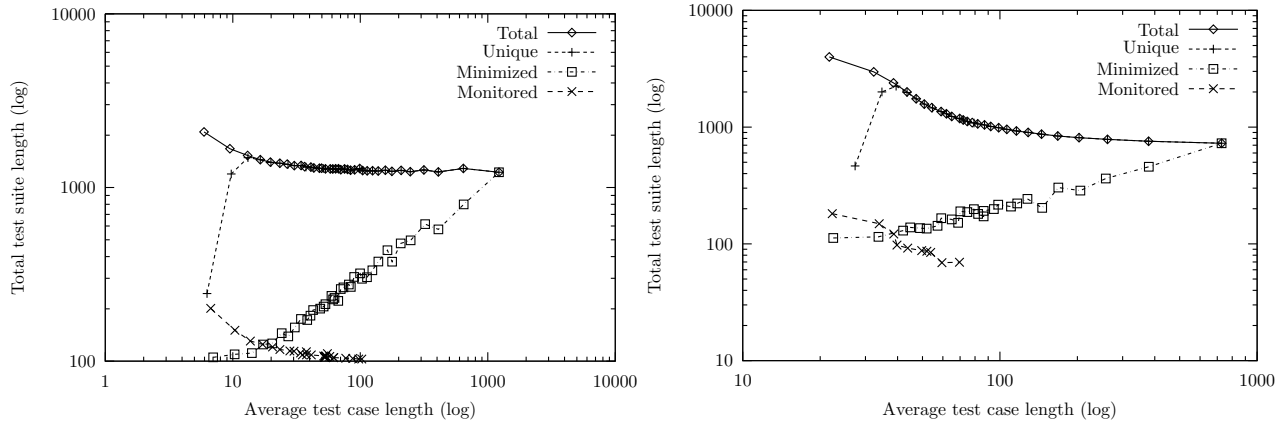
Duplicate and subsumed test cases are common for short lengths, but starting at a certain length there are hardly redundant test cases, and the curves for 'Total' and 'Unique' collapse. At a much greater length there is also a point where the test suites generated are minimal, which means that no more test cases can be removed, and the curves for 'Total' and 'Minimized' collapse. This is typically the case when there are only in the order of three test cases in a test suite. While monitoring does create more test cases initially, the number of test cases is smaller than when using minimization for longer test cases.

Figure 2 shows how the average test case length is related to the total length of a test suite, which is calculated as the sum of the lengths of all test cases. The figure shows mutation adequate test case generation in greater detail, while the other criteria are omitted for space reasons again but are similar in nature. Considering test suites of unique test cases there is a noticeably increase in the total test suite length with increasing test case length initially, but once the number of unique test cases and total number of test cases coincide this increase is reversed and the total length continues to decrease. A test suite consisting of only a single very long test case is usually similar in length to a test suite consisting of as many as possible short test cases. With regard to the minimized test suites Figure 2 reveals that the total length increases with the average test case length, while for monitoring the total length decreases; for longer test cases,



(a) Cruise (b) SIS

Figure 1. Mutation test suites, average length vs. number of test cases.



(a) Cruise (b) SIS

Figure 2. Mutation test suites, average length vs. total length.

monitoring results in the smallest test suites with regard to the total length.

Figure 3 relates transition pair coverage and average test case length for the Cruise example, and pairwise coverage for the SIS example. Once more, the other criteria are omitted for space reasons but are similar in nature. If the coverage is low for the short test cases then in all experiments longer test cases increase the values. However, in many of our experiments the coverage level was already relatively high and the increase was only minor. Note that we only consider unique test cases here because duplicate and subsumed test cases do not add to coverage values or mutation scores in our setting.

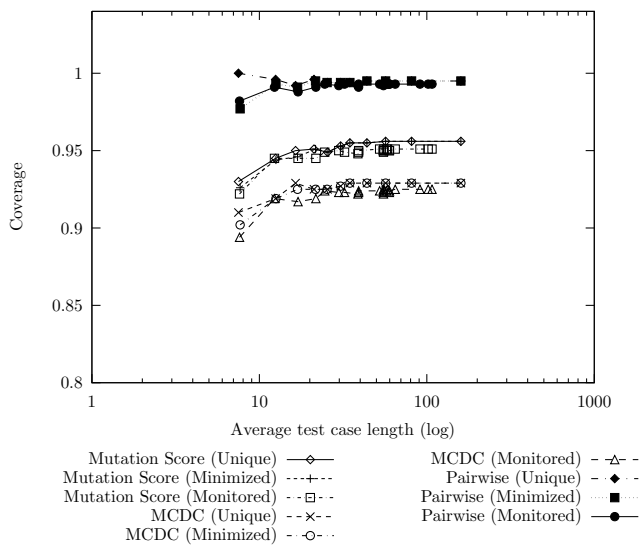
Figure 4 gives an example of the effects of the average test case length on the time necessary to create test suites. In all experiments, there was no significant change in the time necessary to generate a full test suite; the post-processing steps of removing redundant test cases and minimization are not considered here. In contrast, the time necessary to generate a test suite with monitoring is significantly lower for longer test cases.

While the previous figures considered the effects of the average test case length in a test suite, Figure 5 shows how for individual test cases the length relates to their mutation scores (experiment 4). This serves to demonstrate the basic assumption that longer test cases are likely to cover more faults. While the criteria we considered all achieved very high mutation scores, random test cases performed comparatively bad.

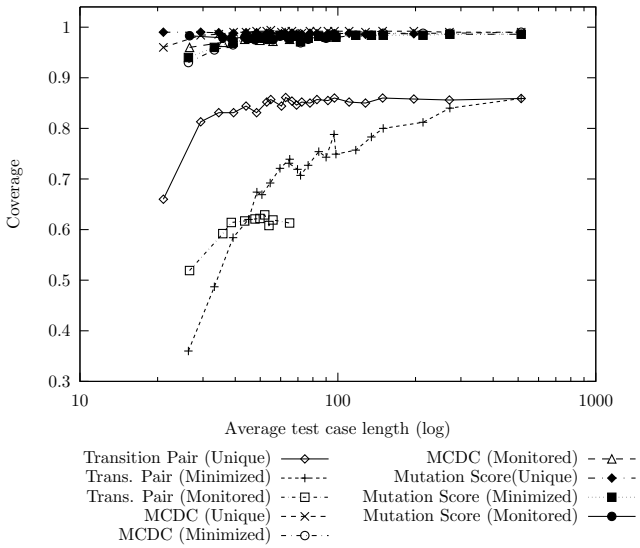
5. Discussion

5.1. Fault detecting capability

When considering the fault detecting capability of a single test case then as expected mutation score (Figure 5) and



(a) Cruise, Transition Pair



(b) SIS, Pairwise

Figure 3. Average length vs. coverage.

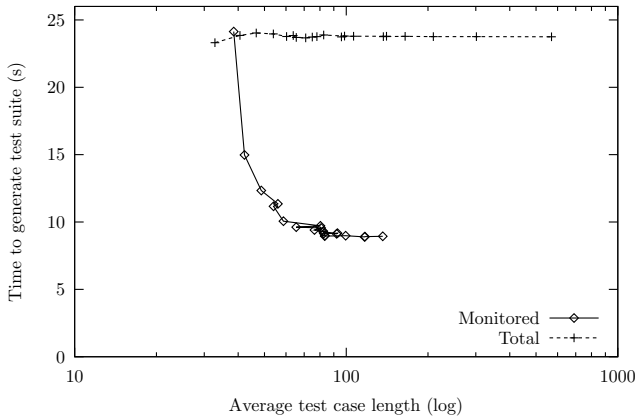


Figure 4. Time to generate test suites vs. average test case length, SIS example, transition pair coverage.

coverage increase with the test case length. However, the fault detecting capability cannot simply be seen as a function of the test case length: Extending the length of random test cases does in general increase the coverage and mutation score, but at a much slower rate than when using different coverage criteria to extend test cases (Figure 5). An obvious explanation for this is that random testing tends to repeatedly explore already visited parts of the state space while using coverage criteria will guarantee that every extension of a test case reaches some previously uncovered part of the state space. Because random testing is so cheap it is feasible to generate much longer test cases than considered in these experiments, which is likely to lead to higher fault detecting capability.

If we consider the corresponding test suites then the effect of the average test case length is not so predictable – after all, the longer the test cases the less test cases are needed for a full test suite (see Figure 1). As illustrated in Figure 3 the coverage usually increases together with the average test case length, even though the number of test cases is reduced. This was observed for the majority of coverage criteria, but there are exceptions (e.g., see pairwise coverage for the Cruise specification in Figure 3). It can also be observed that while the shortest test cases have the smallest coverage and an increase of length improves the coverage initially, there is less additional improvement when further increasing the length very much.

Consequently, in order to answer research question 1 we assume that in order to increase the fault detecting capability it seems feasible to increase the test case length and not use the shortest possible test cases. As to how long exactly the test cases should be for *maximal* fault detecting capability there are two possible suggestions: One is to create test cases that cover as many test requirements as possible, possibly even only a single test case covering all test requirements: Our experiments show that this gives good values for coverage and mutation score. The alternative suggestion is to test as much as possible in terms of the test suite length. As our experiments show this is usually achieved with test case lengths closer to the short end of the scale, where the total test suite length is maximized (around the point where total and unique number of test cases collapse, see Figure 2). This point also coincides with the point where

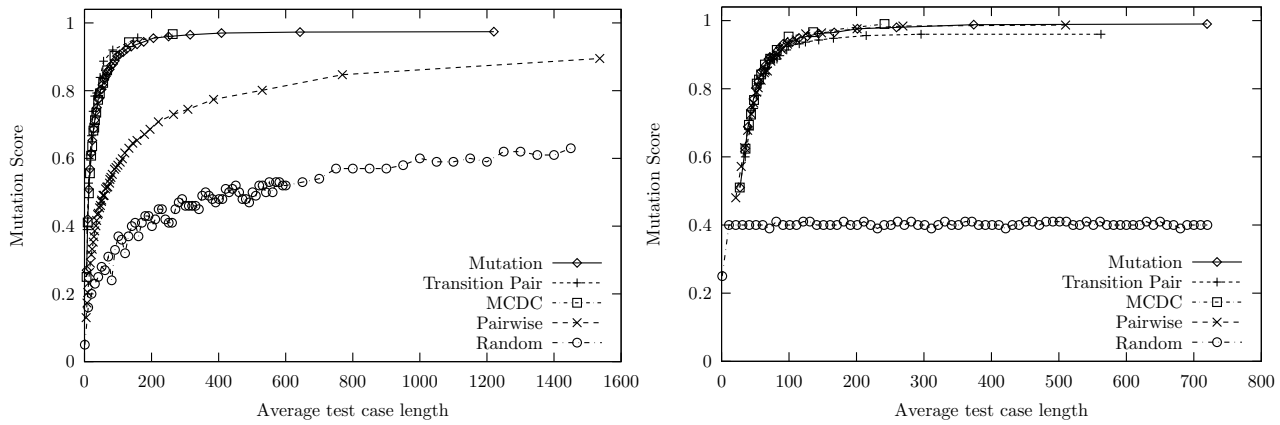


Figure 5. Mutation score of individual test cases.

the coverage with increasing test case length stops increasing significantly (see Figure 3).

5.2. Costs

It is difficult to draw generic conclusions regarding the influence of the test case length on the testing costs. The execution costs depend on several factors: Depending on the system under test the setup and pull-down costs of a single test case might be very high; in such a case it is advantageous to minimize the number of test cases. If setup and pull-down costs are negligible, then it will be advantageous to minimize the total length of a test suite. In addition, there might be other factors such as whether test cases have to be understandable for test engineers, in which case they might, for example, have to be as short as possible. Consequently, we distinguish three different aspects to minimize the costs: The size, the total length, and the test case generation costs.

The trivial answer to minimize the number of test cases is to simply generate a single very long test case that satisfies all test requirements. However, as can be seen in Figure 2 this long test case is in most cases longer than the total length of test suites with shorter test cases.

For test suites without any optimization ('Total' in the graphs) there is a steady decrease of the total length with increasing average test case length, therefore the minimal test suite length in this case is achieved with maximal test case length (see Figure 2) . Consequently, the longer the test cases the smaller the test suite length, and the answer to research question 2 in such a scenario is no in general. When removing duplicate and subsumed test cases ('Unique') the minimal length is achieved for shorter test cases, as the number of test cases that can be dropped reduces with increasing test case length, up to a point where there are no more redundant test cases (see Figure 1). Consequently, the answer to research question 2 in such a scenario is yes.

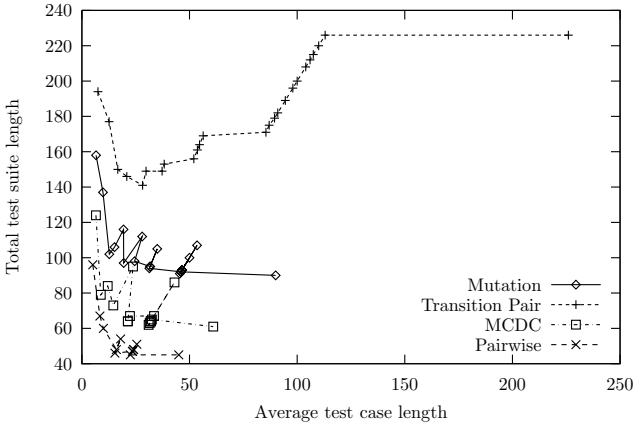


Figure 6. One run of experiment 3 (monitoring) on the Cruise example; total test suite length vs. average test case length.

Interestingly, when applying minimization an increase in test case length actually leads to a greater total test suite length (Figure 2). In such a scenario the answer to research question 2 is yes. This is because with increasing test case length less test cases can be dropped from a test suite with regard to achieving coverage. Consequently, when using minimization the smallest possible test suite will be achieved by using short test cases.

When using monitoring during test case generation the total test suite length also generally decreases with increasing test case length. However, we observed that the order in which trap properties are selected can sometimes have an effect on this: Figure 6 shows one particular run of experiment 3 on the Cruise specification for transition pair coverage where the test suite length initially decreases but then

starts increasing again at a certain average test case length (around 38 states). A closer look reveals that the number of test cases stays almost constant in the time where the total length increases again. However, the average case shown in Figure 2 shows that the number of test cases decreases with increasing test case length. Consequently, we conclude that in a scenario where monitoring is applied the shortest possible test suite can be created by using as long as possible test cases, as long as the increase in length leads to a reduction of the number of test cases. In such a scenario the answer to research question 2 is no.

The computational costs of the actual test case generation are also minimized by using monitoring and long test cases (see Figure 4). Although experiments showed no effect on the test case generation time when not using monitoring, it is still advisable to increase the test case length such that duplicate test cases are avoided, as this reduces the post-processing effort. There might also be other factors dependent on the underlying test case generation technique.

5.3. Threats to validity

There are several threats to the validity of our results: First of all, the question is how far the results in our testing scenario can be generalized to other testing scenarios. For example, it might not always be possible to execute test cases of deliberate length. Even though in such a case long test cases can be simulated with a dedicated reset transition further experiments in this direction are necessary. Furthermore, the influence of the test case generation technique was not analyzed in this paper: For example, a depth first search resulting in very long test cases might change some of the findings.

Our experiments only considered deterministic systems; for non-deterministic systems test cases are not necessarily linear sequences possibly leading to different conclusions. We did not discuss the costs of test objective decomposition, test case generation and assembling in detail in this paper: The test objective decomposition in our scenario is independent of how test suites are generated, and the costs of test case generation with our method basically correlate with the test suite length except when applying monitoring (e.g., see Figure 4).

In our experiments we chose trap properties in random order. Each experiment was repeated 10 times with different random orders and the values were averaged. It is conceivable that the order in which trap properties are selected can have an influence on the results; for example, the length of a test case might vary according to the order of trap properties, and consequently the size of a test suite can vary also. However, experiments with regard to trap property order [12] lead us to the assumption that repeating the experiments 10 times should be sufficient to remedy effects of the order.

Some kinds of faults, not considered in this paper, may provide an additional motive for preferring either long or short test cases. For instance, discovering a fault in an implementation that adds new extra states requires very long input sequences, also called *combination locks*. This fact has been proved for finite state machines [20], where the minimum length of tests capable of discovering k extra states increases exponentially with k.

5.4. Related work

Andrews et al. [3] investigated the question whether to execute a small number of long test cases or a large number of short test cases in the context of random testing. We only considered random testing in a limited scope (e.g., only similar test case lengths as for the other techniques), therefore this work can be seen as complementary to ours. Similarly to our findings, their results show that the length has a major influence on the effectiveness of random testing, but the optimal length with regard to quality and costs varies.

Rothermel et al [22] performed experiments on the *test suite granularity* – which essentially is similar to the number of test cases and their length – in the context of regression testing, which was not part of our investigations; they arrived at similar conclusions as this paper regarding the effects on test suite reduction and fault finding effectiveness. In contrast, Xie and Memon [23] showed that in a GUI testing context it is favorable to initially generate many short test cases to cover 'shallow' bugs, and then turn to longer test cases for 'deep' bugs.

6. Conclusions

In this paper we have reported on investigations regarding the structure of test suites. In particular it is of interest how the length of individual test cases influences the characteristics of a test suite. To analyze this we have performed a set of experiments, where the average length of test cases is continually increased and the effects on test suite size, length, coverage, redundancy, minimization, and monitoring were observed.

When deciding whether to prefer long or short test cases there are many different special cases depending on the testing environment that need to be considered. For example, longer test cases are counterproductive if minimization is applied as a post-processing step. However, in most scenarios it seems feasible to give preference to fewer longer test cases instead of many short test cases: In fact an increase in test case length can reduce the overall size and length of the resulting test suites while actually increasing the fault detecting capability at the same time.

References

[1] A. Abdurazik, P. Ammann, W. Ding, and J. Offutt. Evaluation of Three Specification-Based Coverage Testing Criteria. In *Proceedings of the 6th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2000)*, pages 179–187, Tokyo, Japan, September 2000. IEEE Computer Society.

[2] P. E. Ammann, P. E. Black, and W. Majurski. Using Model Checking to Generate Tests from Specifications. In *Proceedings of the Second IEEE International Conference on Formal Engineering Methods (ICFEM'98)*, pages 46–54. IEEE Computer Society, 1998.

[3] J. Andrews, A. Groce, M. Weston, and R.-G. Xu. Random Test Run Length and Effectiveness. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008)*, pages 19–28, Sept. 2008.

[4] R. Bharadwaj and C. L. Heitmeyer. Model Checking Complete Requirements Specifications Using Abstraction. *Automated Software Engineering*, 6(1):37–68, 1999.

[5] A. Calvagna and A. Gargantini. A Logic-Based Approach to Combinatorial Testing with Constraints. In *Tests and Proofs*, volume 4966 of *Lecture Notes in Computer Science*, pages 66–83. Springer-Verlag, 2008.

[6] J. Chilenski and L. A. Richey. Definition for a masking form of modified condition decision coverage (MCDC). Technical report, Boeing, 1997.

[7] V. Chvatal. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, 4(3), 1979.

[8] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: A New Symbolic Model Verifier. In *CAV '99: Proceedings of the 11th International Conference on Computer Aided Verification*, pages 495–499, London, UK, 1999. Springer-Verlag.

[9] E. M. Clarke, O. Grumberg, K. L. McMillan, and X. Zhao. Efficient Generation of Counterexamples and Witnesses in Symbolic Model Checking. In *Proceedings of the 32st Conference on Design Automation (DAC)*, pages 427–432. ACM Press, 1995.

[10] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA., 1 edition, 2001. 3rd printing.

[11] G. Fraser and F. Wotawa. Using LTL Rewriting to Improve the Performance of Model-Checker Based Test-Case Generation. In *A-MOST '07: Proceedings of the 3rd International Workshop on Advances in Model-based Testing*, pages 64–74, New York, NY, USA, 2007. ACM Press.

[12] G. Fraser and F. Wotawa. Ordering Coverage Goals in Model Checker Based Testing. In *A-MOST'08: Proceedings of the 4th International Workshop on Advances in Model-based Testing*, pages 31–40, April 2008.

[13] G. Fraser, F. Wotawa, and P. E. Ammann. Testing with model checkers: a survey. *Software Testing, Verification and Reliability*, 2009. To appear.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[15] A. Gargantini. Using Model Checking to Generate Fault Detecting Tests. In *Proceedings of the International Conference on Tests And Proofs (TAP)*, volume 4454 of *Lecture Notes in Computer Science*, pages 189–206, Zurich, Switzerland, 2007.

[16] A. Gargantini and C. Heitmeyer. Using Model Checking to Generate Tests From Requirements Specifications. In *ESEC/FSE'99: 7th European Software Engineering Conference, Held Jointly with the 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, volume 1687, pages 146–162. Springer, 1999.

[17] G. Hamon, L. de Moura, and J. Rushby. Generating Efficient Test Sets with a Model Checker. In *Proceedings of the Second International Conference on Software Engineering and Formal Methods (SEFM'04)*, pages 261–270, 2004.

[18] M. J. Harrold, R. Gupta, and M. L. Soffa. A Methodology for Controlling the Size of a Test Suite. *ACM Trans. Softw. Eng. Methodol.*, 2(3):270–285, 1993.

[19] J. Kirby. Example NRL/SCR Software Requirements for an Automobile Cruise Control and Monitoring System. Technical Report TR-87-07, Wang Institute of Graduate Studies, 1987.

[20] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - A survey. In *Proceedings of the IEEE*, volume 84, pages 1090–1126, 1996.

[21] S. Rayadurgam and M. P. E. Heimdahl. Coverage Based Test-Case Generation Using Model Checkers. In *Proceedings of the 8th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2001)*, pages 83–91, Washington, DC, April 2001. IEEE Computer Society.

[22] G. Rothermel, S. Elbaum, A. Malishevsky, P. Kallakuri, and B. Davia. The impact of test suite granularity on the cost-effectiveness of regression testing. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 130–140, New York, NY, USA, 2002. ACM.

[23] Q. Xie and A. M. Memon. Studying the Characteristics of a "Good" GUI Test Suite. In *ISSRE '06: Proceedings of the 17th International Symposium on Software Reliability Engineering*, pages 159–168, Washington, DC, USA, 2006. IEEE Computer Society.