

Test Oracles Using Statistical Methods

Johannes Mayer, Ralph Guderlei

Abteilung Angewandte Informationsverarbeitung, Abteilung Stochastik
Universität Ulm
Helmholtzstrasse 18
D-89069 Ulm, Germany
jmayer@mathematik.uni-ulm.de, rjg@mathematik.uni-ulm.de

Abstract: The oracle problem is addressed for random testing and testing of randomized software. The presented Statistical Oracle is a Heuristic Oracle using statistical methods, especially statistical tests. The Statistical Oracle is applicable in case there are explicit formulae for the mean, the distribution, and so on, of characteristics computable from the test result. However, the present paper only deals with the mean. As with the Heuristic Oracle, the decision of the Statistical Oracle is not always correct. An example from image analysis is shown, where the Statistical Oracle has successfully been applied.

1 Introduction

Especially if test input generation is very complicated, for example the generation of large complex images, random testing, i. e. random generation of test inputs, can easily be used to generate a large number test cases—in addition to the manually generated ones. According to [DN84, Fr98, Fr99, HT90], random testing is effective. Random testing increases the chance of finding bugs [Ho01]. Whereas the random generation of test inputs is simple, the corresponding expected results are usually not obvious. This is the well-known test oracle problem. A test oracle is responsible for the decision, whether a test case passes or not. If no expected results are provided, which can be compared to the actual results, more complex oracles are needed.

Furthermore consider testing randomized software, i. e. programs whose output is random. There can be no expected output for any test case input. For example the random simulation of a geometric model produces another result each time it is called. Thus more sophisticated test oracles are necessary also in this case.

The present paper contributes to the solution of the test oracle problem in the above cases if explicit formulae for the mean, the distribution, and so on, of characteristics computable from the test results are available. A Statistical Oracle using statistical methods is presented, being a special case of the Heuristic Oracle [Ho99] resp. Parametric Oracle [Bi00]. The Statistical Oracle is based on statistical methods, especially statistical tests. Therefore,

unlike usual oracles, the decision of a Statistical Oracle is unfortunately not always correct. The following section contains a review of some related work. Thereafter, the Statistical Oracle is described using a pattern. Then, the necessary statistical methods are presented to implement the components of the described pattern and implementation details are also discussed for a Java implementation. Finally, a successful application of the Statistical Oracle in the field of image analysis is described, followed by a conclusion and perspectives.

2 Related Work

Random testing, i. e. random generation of test inputs, is a well-known and efficient test case generation strategy [Ag78, DN84, Fr98, Fr99, HT90, Ha94, Sc79]. It requires test oracles to ensure the adequate evaluation of test results.

Most oracles, such as Solved Example Oracle, Simulation Oracle, Gold Standard Oracles, Reversing Oracle, Generated Implementation Oracle, and Different But Equivalent Oracle (all described in [Bi00]), check the exact actual output for correctness. In the above oracles, the expected results are specified by hand, generated through a simplified, prototype, or gold standard implementation, verified using simple computations, produced by an implementation generated from a formal specification [Ja89, Ja92], or generated from equivalent executions of the IUT, respectively.

The oracles so far exactly compared the expected outputs with the actual. However, this is not always feasible. Therefore, oracles exist that only verify some properties of the actual output or during the test. The Built-in Test Oracle [Bi00] requires the IUT to have some self-checking mechanisms integrated, such as assertions verifying constraints. Similarly, for complex software, checking of the actual outputs can be done exactly for simple inputs or approximately—concerning constraints, accuracy, or inconsistencies—as described in [We82]. Finally, the Heuristic Oracle [Ho99] resp. the Parametric Oracle [Bi00] extracts some parameters from the actual outputs and compares them with the expected values. Hoffman [Ho99] as well as Binder [Bi00] mention the use of statistical parameters such as the mean and the variance. However, they do not detail how the comparison is to be performed—which is essential in this case. The present papers tries to fill this gap by the explicit description of the Statistical Oracle—a special case of the Parametric Oracle resp. the Heuristic Oracle—giving also necessary implementation details especially for the comparison. The following section contains this description in the form of a pattern.

3 Statistical Oracle

Intent

Verify some statistical characteristics of the actual test results.

Motivation

In case of randomized software and random testing, the actual test result is random. Therefore, it is not possible to give an exact expected value. This pattern presents a test oracle for these cases employing statistical methods, esp. statistical tests, to compare expected statistical characteristics with the actual test results.

Applicability

This pattern can be used if

- the IUT is randomized software or
- test cases are generated randomly

and there are explicit formulae for the mean, variance, distribution, and so on, of characteristics computable from the test results. The main areas in which the above conditions usually hold are random simulations of various models such as graphs, models from stochastic geometry, and so on, and computation of characteristics from those models (in which case these models can be used for test input generation).

Examples

Testing the implementation of an algorithm that computes the area and boundary length of the black phase of a binary image requires many binary images as test inputs. It is quite complex to generate those images and determine the respective expected area and boundary length. However, generating images randomly as realizations of a spatial stochastic model (being discretized) is very simple and if there are explicit formulae for the mean area and mean boundary length, random testing can be used. This example is described in detail in Section 5.

Furthermore, given a trusted implementation for the computation of the area and the boundary length, the above scenario can be used to test the implementation of the simulation of the spatial stochastic model.

Participants

Figure 1 shows the principal structure of the Statistical Oracle in case of random test input generation. It consists of a statistical analyzer and a comparator. The statistical analyzer computes various characteristics that may be modeled as random variables. The comparator computes the empirical sample mean and the empirical sample variance of its

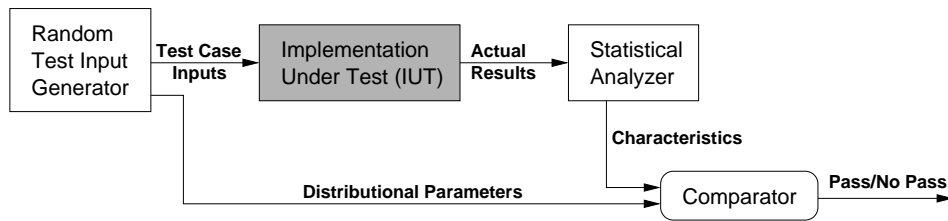


Figure 1: Statistical Oracle for random test input generation

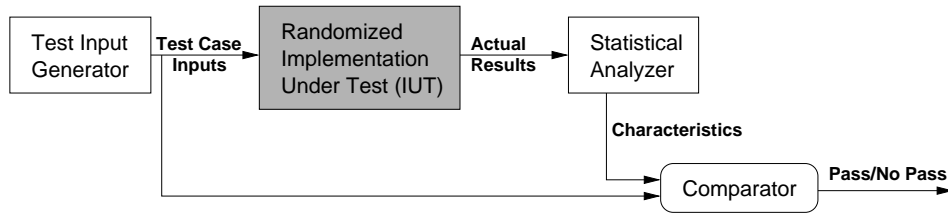


Figure 2: Statistical Oracle for randomized software

inputs. Furthermore, expected values and properties of the characteristics are computed by the comparator (based on the distributional parameters of the random test input).

For randomized software the Statistical Oracle is basically the same. Its principal structure is depicted in Figure 2. In this case, the test case inputs contain the parameters of the model simulated. Therefore, the comparator receives the test case inputs directly. Besides this, the comparator and the statistical analyzer are as described above.

Collaborations

- The statistical analyzer delivers the characteristics computed to the comparator.
- In case of random input generation, the comparator receives the distributional parameters from the random test input generator. Therefore, the random test input generator must be prepared to deliver the distributional parameters to the comparator.

Implementation

Section 4 details the implementation of the statistical analyzer and the comparator. Therefore, necessary statistical methods are presented and a Java implementation is discussed.

Consequences

The decision of a Statistical Oracle is not always correct—in contrast to usual oracles. At best, the probability for a correct decision can be given.

An important restriction of the Statistical Oracle is that it cannot decide whether a single test case passes or not. It can only make this decision for a couple of test cases. If a failure occurs, no single test case can be identified that detected the bug—it is the couple of test cases as a whole.

The Statistical Oracle does not check the actual output but only some characteristics of it. Therefore, as explained in [Bi00, Ho99], it does not suffice to perform all tests with a Statistical resp. Parametric resp. Heuristic Oracle, since these tests only focus on the observed characteristics. Therefore, other test cases and oracles are also necessary. However, in case of randomized software this seems to be impossible, since the output is always random.

Known Uses

Section 5 describes an example where the Statistical Oracle has successfully been applied.

Related Patterns

This pattern is a special case of the Heuristic Oracle [Ho99] resp. the Parametric Oracle [Bi00]. Here statistical characteristics are employed and compared using statistical methods, esp. statistical tests.

4 On the Implementation of the Statistical Analyzer and the Comparator

For an application of the Statistical Oracle, the implementations of the comparator and the statistical analyzer have to be detailed. The statistical analyzer outputs characteristics—some for each test case. The computation of these characteristics is usually based on estimators. The implementation of the statistical analyzer, therefore, depends on the IUT and cannot be generalized. The comparator collects the outputs of the statistical analyzer for n test cases and computes the sample mean and the sample variance. Thereafter, it decides based upon heuristics or statistical tests. The comparator allows for generalization. In the following, some statistical basics are explained that are necessary for the implementation of the comparator. More details on the necessary statistics are provided e. g. by [CB02]. Thereafter, the Java implementation is discussed.

4.1 Statistical Methods

Let X_1, \dots, X_n denote the random variables that model the inputs of the comparator for a single characteristic, where X_i belongs to the i th test case. Since the individual test runs are completely independent of each other, the random variables X_i are independent and identically distributed, say with mean μ and variance σ^2 . Both, μ and σ^2 , are unknown, since they depend on the IUT which is to be tested.

The sample mean of these random variables X_1, \dots, X_n is

$$\overline{X}_n := \frac{1}{n} \sum_{i=1}^n X_i.$$

According to the central limit theorem, it holds that

$$\frac{\overline{X}_n - \mu}{\sigma/\sqrt{n}} \xrightarrow{d} \mathcal{N}(0, 1)$$

for $n \rightarrow \infty$. Thus, for practical purposes, \overline{X}_n can be regarded as approximately normally distributed with mean μ and variance σ^2/n if $n \geq 30$ (a common rule of thumb). The greater n gets, the less likely deviations from μ become (which is also known as the weak law of large numbers).

Additionally, the sample variance

$$S_n^2 := \frac{1}{n-1} \sum_{i=1}^n (X_i - \overline{X}_n)^2$$

of the random variables X_1, \dots, X_n , that approaches σ^2 as n goes to infinity, will also be necessary for the statistical tests.

So far, only random variables have been considered. In case of a concrete test run, x_i , \overline{x}_n , and s_n^2 denote the respective realizations of these random variables.

In the following μ_0 , denotes the mean that the random variables X_i are expected to have. (The input generator is required to deliver the distributional properties to the comparator. Based on them, μ_0 can be computed by the comparator.) The following approaches are used to decide whether the actual mean μ equals μ_0 or not.

Simple Approach Given the expected mean μ_0 , a first approach to check whether the mean μ equals μ_0 is to compare the empirical sample mean \overline{x}_n with μ_0 . If the sample mean deviates more than ten percent, say, of μ_0 from μ_0 , the IUT does not pass, i. e. if

$$|\overline{x}_n - \mu_0| > 0.1 \cdot \mu_0.$$

The heuristic presented so far is only a simple strategy to check that the mean of the characteristic is approximately equal to the expected value.

Advanced Approach Now, a statistical hypothesis test should be employed to test, whether the mean is equal to the expected value. However, it is not that simple. It seems to be obvious to use the t-test. However, the null hypothesis of this test states that the mean is equal to the expected value. This hypothesis thus states that the IUT is correct in that respect. So, a Type I error is in this case that the test does not pass whereas the IUT is correct (at least regarding this aspect). This is not the error whose error probability should be controlled. It would be preferable if the probability that the IUT passes whereas it is buggy, could be chosen arbitrarily. It is however not possible to simply exchange the null hypothesis and the alternative hypothesis.

Using an intersection-union test [CB02] that combines two one sided t-tests, this problem can be overcome. Let $\varepsilon > 0$ be chosen arbitrarily to define an environment around the mean, the null hypothesis can be stated as

$$\mu \notin [\mu_0 - \varepsilon, \mu_0 + \varepsilon].$$

The probability $\alpha \in (0, \frac{1}{2})$ for a Type I error, i. e. that the IUT passes though the IUT is not correct, can be chosen arbitrarily.

Then, if

$$\frac{\bar{x}_n - (\mu_0 - \varepsilon)}{s_n/\sqrt{n}} \geq t_{n-1, \alpha/2} \quad \text{and} \quad \frac{\bar{x}_n - (\mu_0 + \varepsilon)}{s_n/\sqrt{n}} \leq -t_{n-1, \alpha/2}$$

hold, the null hypothesis is rejected and thus the implementation passes. $t_{n-1, \alpha/2}$ denotes the $(1 - \alpha/2)$ -quantile of the t-distribution with $n - 1$ degrees of freedom.

The probability of a Type II error, i. e. that the IUT does not pass whereas there is no error regarding the tested aspect, cannot be chosen arbitrarily. However, it is important that the probability of this error is not too high, since in this case time is wasted for searching a non-existent bug. The probability of this error can however, given a fixed value for α , be decreased by increasing the sample size n .

4.2 Java Implementation

To support automatical testing, the presented approaches can easily be implemented using Java and the JMSLTM Numerical Library 2.5 from Visual Numerics, Inc.¹ The implementation of the simple approach is obvious. Therefore, only the implementation of the advanced approach is shown:

```
public boolean passes(double[] x, double mu_0,
                    double eps, double alpha) {
    int n = x.length;
    double x_n = com.imsl.stat.Summary.mean(x);
    double s_n = com.imsl.stat.Summary.variance(x);

    double t_1 = (x_n - (mu_0 - eps)) / (s_n / Math.sqrt(n));
```

¹<http://www.vni.com/products/imsl/jmsl.html>

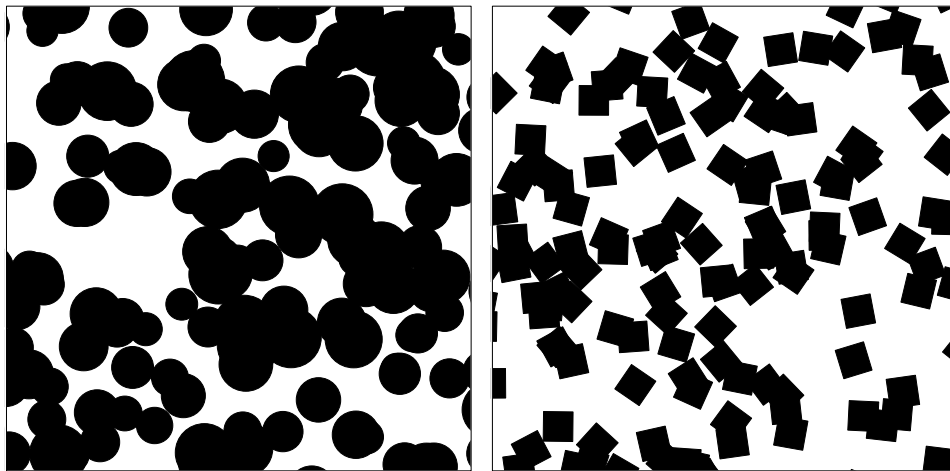


Figure 3: Possible realizations of the Boolean model

```

double t_2 = (x_n - (mu_0 + eps)) / (s_n / Math.sqrt(n));

double quantile = -com.imsl.stat.Cdf.inverseStudentsT(alpha/2.0, n-1);

return t_1 >= quantile && t_2 <= -quantile;
}

```

The above method determines whether the null hypothesis is rejected and consequently the IUT passes.

5 An Example from Image Analysis

In [SS04] estimators for so-called morphological characteristics, e. g. area and boundary length, are presented, while [KSS04] deals with algorithms computing these characteristics for the black phase of digital binary (i. e. black and white) images.

In the following, it is described, how implementations of these algorithms have been tested. It would be very time consuming to generate a lot of complex large input images together with the respective values of area and boundary length by hand. So, this has only been done for a few special cases.

Otherwise, the presented approach has been used with random test input generation. Therefore, the Boolean model (cf. e. g. [Mo97]), a spatial stochastic model, has been employed, which has been studied for a long time and for which many theoretical results exist. Figure 3 shows two possible realizations of the Boolean model. The Boolean model is simply the union of random grains (such as discs with random radius or squares with random rotation) each translated in another point of an underlying point process. A detailed in-

roduction to the Boolean model is given in e. g. [Mo97]; see also [MSS04] for related simulation issues. Many different types of Boolean models (e. g. with discs, rectangles, and squares) were used for random test input generation, and so many relevant situations for the practical use of the algorithm were tested. So far the random input generation.

Tests using the Statistical Oracle are thus quite flexible, since the type of test inputs can easily be changed. Only some parameters of the random input generator and the reference values within the comparator have to be changed.

The algorithm implemented in the IUT is based on the estimators for area and boundary length of the black phase. A key assumption is that the properties of these estimators hold for the algorithm—despite of discretization issues—and thus the computed means of area and boundary length equal the theoretical means.

In this case, the statistical analyzer was only a dummy that directly passed the actual results of the IUT to the comparator.

The actual test results were collected in the comparator for the analysis. The simple approach was straightforward to implement. As described in Section 4.2, the advanced approach was implemented in Java. Using a high-quality JMSL library reduces the risk of bugs in the test code, as the statistical tests are more complicated to implement and require non-standard functionality as quantile functions.

During the development, both, the simple and the advanced approach, were applied. As the simple approach is very easy to implement, it was used from the beginning. So it was possible to detect major bugs early in the development process. The detected bugs affected errors in memory access, errors in branching decisions, wrong operators, or several bugs depending on object states. These bugs were reliably indicated by large deviations of the empirical mean from the reference value. The problem we encountered with the simple approach were reference values close to zero (i. e. $\leq 10^{-5}$). In this case, even small deviations in absolute values resulted in large relative deviations. So the test cases failed using the simple approach, despite the result was correct in five or more decimals.

The advanced approach was performed for every Boolean model used for the input generation. That took place late in the development process, and so no more bugs could be found, but the results from the simple approach were verified. As a result of the advanced approach, one could say (with a given probability of at least 99%) that the IUT is correct with respect to the mean value of area and boundary length.

For some of the Boolean models, the advanced approach test cases failed, but the failures did not indicate a bug in the implementation as the failures could be explained by discretization effects. The realizations of the Boolean model are discretized when the digital binary images are generated. Therefore, the resulting structures have slightly different characteristics (area and boundary length) than the smooth ones. As the calculation of the reference values is based on the characteristics of the smooth structures, the actual output differs from the reference values. The difference is so small, that the simple approach did not detect it, but the statistical test failed due to this difference for some of the scenarios. These failures could be avoided by considering the polygonal structure for the computation of the reference values.

6 Conclusion and Perspectives

The present paper dealt with test oracles in combination with random test input generation or randomized software. Its contribution is the formalization of the Heuristic Oracle in combination with statistical methods, and mainly the description of its implementation. A pattern for the Statistical Oracle has been presented and discussed for both cases. For the implementation of this patterns, the necessary statistical methods and Java details have been described. An example from image analysis has been shown, where such an oracle could successfully be applied.

The presented approach is applicable in situations where characteristics for which explicit formulae are known concerning their mean, distribution, and so on, can be computed from the test results. However, the present paper only dealt with tests for the mean.

It is important to notice that the output of a Statistical Oracle is not always correct. For the advanced approach error probabilities can be given. Under the null hypothesis, i. e. given a faulty IUT (with respect to the considered aspect), its output is correct with chosen probability $1 - \alpha$. Otherwise, the error probability, i. e. the probability of a Type II error, can be computed numerically.

It has been shown that, using a suitable null hypothesis, the probability that a faulty IUT passes can be chosen to fit given needs. Additionally, it has been explained how the sample size, i. e. the number of test results grouped for the computation of the characteristics, can be used to reduce the probability that a correct IUT (at least with respect to the considered characteristics) does not pass. The greater the sample size, the smaller this error probability gets.

In the present paper, only statistical methods for the mean value have been presented. However, there are many further possibilities. If the distribution is known, e. g. a normal distribution, this can also be tested using a chi-square test. Furthermore, analogous to the mean value similar tests for the variance also exist. These can also be applied if the theoretical variance is known.

Acknowledgment

The authors are grateful to Frank Fleischer, Hendrik Schmidt, and Evgueni Spodarev for valuable comments.

References

- [Ag78] Agrawal, V. D.: When to Use Random Testing. *IEEE Transactions on Computers* **27** (1978) 1054–1055
- [Bi00] Binder, R. V.: *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley, 2000.

- [CB02] Casella, G., Berger, R. L.: *Statistical Inference*. Wadsworth Group, Duxbury, CA, USA, 2002.
- [DN84] Duran, J. W., Ntafos, S. C.: An Evaluation of Random Testing. *IEEE Transactions on Software Engineering* **10** (1984) 438–444
- [Fr98] Frankl, P. G., Hamlet, R. G., Littlewood, B., Strigini, L.: Evaluating Testing Methods by Delivered Reliability. *IEEE Transactions on Software Engineering* **24** (1998) 586–601
- [Fr99] Frankl, P. G., Hamlet, R. G., Littlewood, B., Strigini, L.: Correction to: Evaluating Testing Methods by Delivered Reliability. *IEEE Transactions on Software Engineering* **25** (1999) 286
- [Ha94] Hamlet, R. G.: Random testing. In: Marciniac, J. J. (ed.), *Encyclopaedia of Software Engineering*, John Wiley & Sons, 1994.
- [HT90] Hamlet, R. G., Taylor, R.: Partition Testing Does Not Inspire Confidence. *IEEE Transactions on Software Engineering* **16** (1990) 1402–1411
- [Ho01] Hoffman, D.: Using Oracles in Test Automation. In: *Proceedings of the Nineteenth Annual Pacific Northwest Software Quality Conference (PNSQC)*, Portland, Oregon, USA (2001) 90–117
- [Ho99] Hoffman, D.: Heuristic Test Oracles. *Software Testing & Quality Engineering* **1** (1999) 29–32
- [Ja89] Jalote, P.: Testing the Completeness of Specifications. *IEEE Transactions on Software Engineering* **15** (1989) 526–531
- [Ja92] Jalote, P.: Specification and Testing of Abstract Data Types. *Computer Language* **17** (1992) 75–82
- [KSS04] Klenk, S., Schmidt, V., Spodarev, E.: A New Algorithmic Approach to the Computation of Minkowski Functionals for Polyconvex sets. Preprint, University of Ulm, 2004. (submitted)
- [MSS04] Mayer, J., Schmidt, V., Schweiggert, F.: A Unified Simulation Framework for Spatial Stochastic Models. *Simulation Modelling Practice and Theory* **12** (2004) 307–326
- [Mo97] Molchanov, I.: *Statistics of the Boolean Model for Practitioners and Mathematicians*. John Wiley & Sons, Chichester, 1997.
- [SS04] Schmidt, V., Spodarev, E.: Joint Estimators for the Specific Intrinsic Volumes of Random Sets. *Stochastic Processes and their Applications* (to appear)
- [Sc79] Schneck, P. B.: Comment on “When to Use Random Testing”. *IEEE Transactions on Computers* **28** (1979) 580–581
- [We82] Weyuker, E. J.: On Testing Non-Testable Programs. *Computer Journal* **25** (1982) 465–470