Optimal Test Distributions for Software Failure Cost Estimation

Walter J. Gutjahr
Department of Statistics, Operations Research and Computer Science
University of Vienna

Abstract — In this article, we generalize the input–domain based software reliability measures by Nelson and by Weiss and Weyuker, introducing expected failure costs under the operational distribution as a measure for software unreliability. This approach incorporates in the reliability concept a distinction between different degrees of failure severity. It is shown how to estimate the proposed quantity by means of random testing, using the Importance Sampling technique from Rare Event Simulation. A test input distribution that yields an unbiased estimator with minimum variance is determined. The practical application of the presented method is outlined, and a detailed numerical example is given.

Index Terms — Input distributions, input–domain–based models, reliability estimation, risk assessment, software failure costs, software reliability, software testing.

I. Introduction

In their book [20], Thayer, Lipow and Nelson proposed a method originally developed by Nelson [14] for estimating the reliability of a computer program. The method is based on statistical sampling theory, and its application requires repeated execution of the given program, guided not by traditional testing aims such as fault detection, but by the aim of obtaining optimal reliability estimates. As a criterion for the optimality of an unbiased reliability estimator, minimum variance was suggested: estimators with small variance have a higher "precision" and deserve more confidence than estimators with large variance. Reliability testing, as advocated by Thayer, Lipow and Nelson, is usually recommended as a seperate step in a software quality assurance process, to be performed after a testing phase oriented on fault detection.

Let us give a short recapitulation of Nelson's approach (cf. [20], Sections 5.2.4 and 6.2): He defines the reliability $R = R(\pi)$ of a program π as the probability that a run of π with random input, selected according to the so–called *operational distribution* (the distribution of inputs during the real use of the program) results in correct execution. In order to estimate R, Nelson divides the input domain D into disjoint subdomains $D_1, \ldots, D_s$. Each D_j is further partitioned into the set D'_j of correctly executed inputs and the set D''_j of failure–causing inputs. Let P_j,P_j' and P_j'' denote the probabilities of an input to be element of D_j, D_j' and D_j'', respectively. Then

$$R = R(\pi) = \sum_{j=1}^{s} P_j' = 1 - \sum_{j=1}^{s} P_j''. \tag{1}$$

For reliability testing, samples of sizes $n_1, \ldots, n_s$ are selected from subdomains $D_1, \ldots, D_s$, where $n_1 + \ldots + n_s = n$ is the total sample size. As an estimator for R,

$$\hat{R} := 1 - \sum_{j=1}^{s} \frac{f_j}{n_j} P_j$$

is used, with f_j denoting the number of tested inputs from D_j that evoked failures, i. e., that belonged to D_j''. The optimal sample sizes n_j for fixed n with respect to variance minimization are then given by

$$n_j \;\approx\; \left(n\sqrt{P_j'P_j''}\right) / \left(\sum_{k=1}^{s} \sqrt{P_k'P_k''}\right) \qquad (j = 1, \ldots, s), \tag{2}$$

where the symbol $\approx$ indicates that n_j has to be rounded to an integer.

The reader should notice that in practical applications, the values P_j' and P_j'' are never known exactly (cf. [20], p. 243). Indeed, if they were, reliability testing would be unnecessary: R could be computed directly from (1). Although this fact does not impair the usefulness of Nelson's model, one could, for theoretical reasons, be interested in a re–formulation of the model where optimizing the determination of R does *not* require that R is already known.

In the last years, some authors argued that Nelson's notion of reliability is too restricted in so far as he considers only the binary characterization of outputs as *correct* or *incorrect* with respect to the given specification. He does not, for example, take different degrees of accuracy of a numerical output into account. In [21], Weiss and Weyuker developed a more general notion of reliability which allows a refined judgment of the discrepancy between the actual behavior of the program and its specification. Their reliability notion is also based on the operational distribution and contains Nelson's definition of reliability as a special case. Weiss and Weyuker gave hints for practical reliability estimation according to their definition, but they did not yet set up a fully elaborated statistical theory of *optimal* reliability estimators, as Nelson did for his concept.

At the end of their article [21], Weiss and Weyuker suggested that one should look for a further generalization of the proposed reliability notion: it could be important to incorporate *error costs* into the definition of reliability. This point was mentioned again in Weiss, Weyuker and Hamlet [23]: The authors argued that, for example, misspelled messages are far less serious failures than cases where a financial system credits deposits to the wrong accounts. So it would be desirable to have a notion of reliability that allows a weighting of different possible failures according to their respective costs.

The same position was advocated in Sherer's book [18]. The author conceives the neglect of failure cost considerations as a main limitation of traditional testing theory; to her opinion, the allocation of test effort should be planned in such a way that it can most effectively reduce *risk*, i. e., expected loss by failures.

In the present article, we modify Nelson's approach to optimal reliability testing in the following three directions:

1. We use *expected failure costs under operational distribution* as a measure for software (un)reliability. It will be shown that the proposed measure is a generalization of Weiss' and Weyuker's, which, in turn, generalizes Nelson's measure. To consider expected failure costs as a fundamental parameter is also in accordance with a recent attempt in software engineering to use risk (defined as above) as a structuring element in the software development process: see Boehm's concept of "risk driven management" of software products [2].

2. The model presented in [20] requires a partition of the input domain into subdomains as well as *deterministic* information on the number of failure–causing inputs in each domain. [1] In that model, it is not possible to encode *probabilistic* prior information, for example of one of the following types:

   (a) "The probability that the numerical input x is failure–causing is estimated to be proportional to the number of digits in the representation of x."

   (b) "There is strong evidence that subdomain D_j is *revealing* (in the sense of Weyuker and Ostrand [22]), i. e., either all inputs in D_i are failure–causing or none, and the probabilities for these two possible cases are estimated to be 0.1 and 0.9, respectively."

   Here, we shall develop a model that basically works without a partition of the input domain. It allows the representation of probabilistic prior information by means of a clear distinction between the given program π under test and its unknown *functionality*, modeling the latter by a random variable. [2] In this way we are also able to avoid the above–mentioned difficulty of the concept presented in [20] that it requires knowledge that, if completely available, would make reliability testing unnecessary. Partitioning the input domain may be used in our approach, if this is helpful for a description of the prior informations accessible to he tester, but the model is also open for other possible descriptions, e. g., by mathematical functions (cf. Section IV).

3. We follow a pure random selection strategy, described by a probability distribution on the entire input domain D. For cases where a partition into subdomains is used, this has the consequence that the sample size n_j for subdomain D_j becomes random. The advantage lies in the fact that we always obtain *unbiased* estimators, whereas

[1] In [15], Nelson investigated a probabilistic extension of his original model, which is again based on a partition of the input domain. This extension was further improved by Ramamoorthy's and Bastani's model (see [16]).

[2] A similar approach was chosen by Gourlay in his analysis of Fault–Based Testing in [6].

Nelson's formula (2) may lead to biased estimators. For example, if $s > n$, at least one n_j has to be zero; in this case, Nelson's estimator $\hat{R}$ is biased on the high side (cf. [20], p. 240).

The basic statistical technique used in our approach is the *Importance Sampling* method of Rare Event Simulation (see, e. g., Kalos and Whitlock [10], Section 4.1): From parts of the input domain that promise to give more information on the expected failure costs, samples are drawn with a higher probability. This "probability change" is compensated afterwards by special weights adjusted to the outcomes of the test.

II. Cost Functions

As in Section I, let D be the *input domain* of a given program π. We always assume that D is finite, i. e., $D = \{x_1, \ldots, x_N\}$. However, N is usually a very large number (for small N, one could resort to exhaustive testing). By $\bar{D}$, we denote the *output domain* of D. For the sake of simplicity, we always assume that π *terminates* at each given input $x \in D$ with some output $y \in \bar{D}$. (For practical applications, this assumption is reasonable, since program execution may be aborted after a pre–specified time. In order to include also non–terminating programs into the model, we would have to extend our definitions along the same lines as in [21], which is possible.) Then the input/output behavior of π is described by a function $F : D \to \bar{D}$, which we call the *functionality* of π.

In practice, the functionality F of a program π under test is neither known with certainty before testing nor after testing, unless exhaustive testing is possible. Nevertheless, *some* (maybe uncertain) information on the functionality is usually already available after the coding phase or after the first test runs. The tester might guess, for example, that specific inputs evoking newly implemented routines have a higher probability of leading to failures than other inputs evoking only routines that have proved to work correctly in a previous version of the program.

We take account of this situation by assuming that the functionality F of a given program is a *random variable*. Randomness is used here to model our uncertainty about the exact input/output behavior of the program.

Let $\mathcal{F} = \{F_1, \ldots, F_M\}$ denote the set of functionalities the tester considers as possible for the given program π. The set $\mathcal{F}$ contains one or several functionalities that are correct with respect to the given specification, and many functionalities that are not correct. The assumption that both D and $\mathcal{F}$ are finite is not essential for our model and can be dropped by the use of more sophisticated technicalities (infinite probability spaces). For the sake of clearer presentation, we restrict ourselves here to the finite case.

Let q_k denote the probability the tester would assign to the event that π has functionality F_k ($k = 1, \ldots, M$). We assume that $q_1 + \ldots + q_M = 1$, i. e., $q = (q_1, \ldots, q_M)$ is a probability distribution on $\mathcal{F}$. Of course, in practical applications it is impossible to estimate all the values q_k; even for relatively small programs, M has to be thought of as a number of astronomical size. We shall see, however, that much less information than the values q_k suffices for the application of the method proposed in the sequel.

By p_i $(i = 1, \ldots, N)$, we denote the probability that input x_i occurs during the real use of π, such that the vector $p = (p_1, \ldots, p_n)$ describes the operational distribution on D.

Now let us assume that a *cost function* C is given. To each input $x \in D$ and output $y \in \bar{D}$, the nonnegative real number $C(x, y)$ specifies the costs arising in the case where π produces output y to input x.

For input/output pairs (x, y) satisfying the specification, no costs arise. So if the specification is described by the function $F : D \to \bar{D}$, it is natural to set $C(x, F(x)) = 0$ for all $x \in D$. In the case of "loose" specifications allowing several output values y to an input value x, we may set $C(x, y) = 0$ for each of these allowed outputs y. If, on the other hand, (x, y) is an input/output pair *not* satisfying the specification, positive *or* zero costs may be assigned to that fault. (Notice that there are situations where a failure is costless.)

For input $x_i \in D$ and functionality $F_k \in \mathcal{F}$, let us set

$$c_{ik} := C(x_i, F_k(x_i))$$

for abbreviation. The number c_{ik} denotes the costs caused by input x_i, provided that π has functionality F_k.

Moreover, using the random variable

$$Y_i := \text{costs caused by input } x_i,$$

we may write

$$c_i := \sum_{k=1}^{M} q_k c_{ik} = E(Y_i), \tag{3}$$

where E denotes the expected value of a random variable.

If π has functionality F_k, the *expected value of the failure costs* under operational distribution is

$$Z(\pi) = Z_k := \sum_{i=1}^{N} p_i c_{ik}. \tag{4}$$

For shortness, we shall sometimes call $Z(\pi)$ the *risk*, a terminology that is consistent with the definition of software failure risk in Sherer [18], p. 26.

Our aim is to estimate the risk $Z(\pi)$. Of course, if the entire functionality of the program under test was known, its risk could (at least theoretically) be determined from the operational distribution. The problem is to derive a risk estimate also in cases where only part of the input/output behavior is observed during the test. For an appropriate selection of test inputs, we shall use prior information on the random variables Y_i.

By the following small example we do not intend to reflect a realistic test situation, but only to explain our formalism:

Example 1. Let the input domain D of a program be given as $D = \{1, 2, 3, 4\}$, such that $x_i = i$, and let the output domain $\bar{D}$ be the set of integers. The program is expected

i	$F_1(i)$	$F_2(i)$	$F_3(i)$	$F_4(i)$
1	1	1	2	0
2	4	2	4	0
3	9	3	6	0
4	16	4	8	0

$i \backslash k$	1	2	3	4
1	0	0	1	1
2	0	2	0	4
3	0	6	3	9
4	0	12	8	16

Table 1. Functionalities F_k, and costs c_{ik} caused by input i under functionality F_k in Example 1.

to compute the square of the input integer. In other words, the specification (the correct functionality) can be described by the function $F(x) = F_1(x) = x^2$. Now let us consider three incorrect functionalities:

- $F_2(x) = x$ ("square" omitted),
- $F_3(x) = x \cdot 2$ (computation of "times" instead of "to the power"),
- $F_4(x) = 0$ (output 0 instead of the computed result)

(see Table 1, left part). We decide to choose the cost function $C(x, y) = |x^2 - y|$, i.e., the costs are the absolute differences between expected and real output. Then the numbers c_{ik}, denoting the costs caused by input x_i in case of functionality F_k, are given by

$$c_{ik} = |x_i^2 - F_k(x_i)| = |i^2 - F_k(i)|$$

(see Table 1, right part). Assume that the probability distribution on the set $\mathcal{F}$ of functionalities is

$$q_1 = 0.7, \quad q_2 = 0.1, \quad q_3 = 0.1, \quad q_4 = 0.1,$$

i.e., there is a 70 percent probability that the program is correct, and a 10 percent probability for each of the considered incorrect functionalities. For the expected costs c_i caused by input i, we obtain then from (3):

$$c_1 = 0.2, \quad c_2 = 0.6, \quad c_3 = 1.8, \quad c_4 = 3.6.$$

Now let us assume that the operational distribution is the uniform distribution on the input domain D, i.e., $p_1 = \ldots = p_4 = 0.25$. Then, according to (4), the risks $Z(\pi_1), \ldots, Z(\pi_4)$ of programs $\pi_1, \ldots, \pi_4$ with functionalities $F_1, \ldots, F_4$, respectively, are the following values:

$$Z(\pi_1) = 0, \quad Z(\pi_2) = 5, \quad Z(\pi_3) = 3, \quad Z(\pi_4) = 7.5.$$

Note that the correct program π_1 has risk zero.

$\bowtie$

Let us consider two special types of cost functions:

1. Nelson Measure

Assume that the specification is given by the function F, and set

$$C(x,y) := \begin{cases} 0, & \text{if } y = F(x), \\ 1, & \text{if } y \neq F(x). \end{cases} \tag{5}$$

Here all failures get the same weight 1. If π has functionality F_k, we obtain

$$c_{ik} = \begin{cases} 0, & \text{if } x_i \text{ is processed correctly,} \\ 1, & \text{otherwise,} \end{cases}$$

and hence $Z(\pi) = Z_k = \sum p_i$, the sum running through all indices i of failure–causing inputs x_i. Therefore

$$1 - Z(\pi) = R(\pi),$$

where $R(\pi)$ is Nelson's reliability measure defined in Section I.

2. Weiss/Weyuker Measure

In [21], program reliability is defined essentially as follows: Assume that an integer–valued *distance measure* $d : \bar{D} \times \bar{D} \to N$ (satisfying the properties of a metric) is given on the output domain $\bar{D}$. Furthermore, let a *tolerance function* $\alpha : D \to N^+$ be defined on the input domain. Again denoting the specification and the functionality of π by F and F_k, respectively, one may set

$$d_\alpha(F, F_k, x) := \min\{\alpha(x),\ d(F(x), F_k(x))\}.$$

Weiss and Weyuker define the (generalized) reliability R_g of π as

$$R_g(\pi) := 1 - \sum_{i=1}^{N} p_i \, \frac{d_\alpha(F, F_k, x_i)}{\alpha(x_i)}.$$

We show that $R_g(\pi) = 1 - Z(\pi)$ for an appropriate cost function C. Set

$$C(x,y) := \frac{\min\{\alpha(x),\ d(F(x), y)\}}{\alpha(x)}.$$

One obtains

$$c_{ik} = \frac{\min\{\alpha(x_i),\ d(F(x_i), F_k(x_i))\}}{\alpha(x_i)} = \frac{d_\alpha(F, F_k, x_i)}{\alpha(x_i)},$$

and hence

$$R_g(\pi) = 1 - \sum_{i=1}^{N} p_i c_{ik} = 1 - Z_k = 1 - Z(\pi).$$

So also the Weiss/Weyuker reliability measure can be expressed in terms of risk. (In Example 1, we used, up to a constant factor, the Weiss/Weyuker measure.)

III. Failure Cost Estimators

Now let us turn to the question how the risk $Z(\pi)$ may be estimated by reliability tests. We shall use *random testing* with repeated, independent selection of test cases from D. A random test is characterized by a probability distribution $t = (t_1, \ldots, t_N)$ on the input domain: t_i indicates the probability that, in a single test run, input x_i is selected as a test case.

For example, we may perform random testing under the operational distribution, i. e., we choose $t = p$. This seems to be the most commonly used technique of reliability testing (cf., e. g., [3] or [12]). The relative frequency of unsuccessful runs under the operational distribution is an unbiased estimate of the unreliability $1 - R(\pi)$ according to Nelson's definition.

If $t \neq p$, this is not true anymore. We may, however, weight runs that produced failures by appropriate scores, such that we obtain unbiased estimates again. In the sequel, we shall develop this approach at the most general level, namely for expected failure costs (risk).

For given $t = (t_1, \ldots, t_N)$, let us define a cost estimator as follows: Let $\bar{x}_1, \ldots, \bar{x}_n$ be n random inputs, selected from the set D independently from each other, each according to distribution t, i. e., $P\{\, x_i \text{ is drawn} \,\} = t_i$ for $i = 1, \ldots, N$. (Note that x_i stands for the ith element in D, while $\bar{x}_i$ stands for the ith random input, so $x_i \neq \bar{x}_i$ in general.) We do sampling *with replacement*: A particular input x_i can be selected more than once, although this is very unlikely to happen, if the sample size n is small compared to the input domain size N.

Assume that the functionality of π is F_k. For the νth obtained input, $\bar{x}_\nu$, we denote by $\bar{p}_\nu$, $\bar{t}_\nu$ and $\bar{c}_{\nu k}$ the operational probability of $\bar{x}_\nu$, the test probability of $\bar{x}_\nu$, and the costs caused by input $\bar{x}_\nu$, respectively. In other words, if $\bar{x}_\nu$ comes out as x_i, then $\bar{p}_\nu = p_i$, $\bar{t}_\nu = t_i$, and $\bar{c}_{\nu k} = c_{ik}$.

Let us now define *average weighted costs during test*,

$$\phi_n := \frac{1}{n} \sum_{\nu=1}^{n} S_\nu \tag{6}$$

with

$$S_\nu := (\bar{p}_\nu / \bar{t}_\nu) \cdot \bar{c}_{\nu k}. \tag{7}$$

Observe that, although F_k is unknown in total, the values $\bar{c}_{\nu k}$ can nevertheless be determined directly from the results of the test.

As weights for the observed costs, we have chosen in (7) the quotient of operational probability $\bar{p}_\nu$ and test probability $\bar{t}_\nu$. We shall exclusively work with test probabilities $\bar{t}_\nu > 0$ (cf. Remark 1 following Theorem 2), so the quotient is always defined.

Now the following holds:

Theorem 1. The average weighted costs during test, ϕ_n, yield an unbiased estimator for the risk $Z(\pi)$, i.e., the expected value of ϕ_n is $Z(\pi)$.

Proof. One immediately computes

$$E(\phi_n) = E\left(\frac{1}{n}\sum_{\nu=1}^{n} S_\nu\right) = \frac{1}{n}\sum_{\nu=1}^{n} E(S_\nu) = E(S_1) = \sum_{i=1}^{N} t_i \cdot \frac{p_i}{t_i} c_{ik} = \sum_{i=1}^{N} p_i c_{ik} = Z_k = Z(\pi).$$

□

We emphasize that Theorem 1 holds independently of the functionality of π: Whatever the true value of k is, ϕ_n is *always* an unbiased estimator.

In the special case (5) of the Nelson measure and with $t = p$, (7) reduces to

$$S_\nu = \begin{cases} 0, & \text{if } \bar{x}_\nu \text{ is processed correctly,} \\ 1, & \text{otherwise,} \end{cases}$$

which yields the usual procedure for estimating $R(\pi)$.

Keeping in mind that ϕ_n is an unbiased estimator for $Z(\pi)$ for *each* test distribution t, what criterion of optimality should we choose in order to select the *best* t? A frequently used criterion in statistical estimation theory is the variance of an estimator: If more than one estimator is available, the one with least variance should be selected. This criterion expresses the demand that estimates should be accurate not only in the average on the long run, but also in a specific single case of application.

In the problem considered here, the situation is more complicated in so far as the functionality of the program π is unknown; each of the possible functionalities $F \in \mathcal{F}$ leads to a specific value for the variance of the estimator ϕ_n when test distribution t is applied. As a measure for the accuracy of the estimate, we shall choose the *expectation* of all these variances, i. e., their average value over all functionalities, weighted by the probabilities q_k. This weighted average is to be minimized.

Theorem 2. For fixed number n of test inputs, the test distribution t that yields minimum average variance of the cost estimator ϕ_n (where the average is taken over all possible functionalities, weighted by their probabilities) is given by

$$t_i = \frac{p_i g_i}{p_1 g_1 + \ldots + p_N g_N} \qquad (i = 1, \ldots, N) \tag{8}$$

with

$$g_i = \sqrt{E(Y_i^2)}. \tag{9}$$

Therein, the random variable Y_i denotes the costs caused by input x_i.

For the proof of Theorem 2, see the Appendix.

We call g_i the *failure cost parameter* of input x_i.

Remark 1. In practical applications, it can always be assumed that both $p_i > 0$ and $g_i > 0$, such that also $t_i > 0$: If $p_i = 0$ should hold, then it would be impossible that input

$x_i \in D$ occurs during the operational use of the program, which contradicts the definition of the input domain as the set of possible inputs. On the other hand, $g_i = 0$ can only hold if $Y_i = 0$ with probability one (note that $Y_i \geq 0$), which means that input x_i cannot lead to failure costs. Such inputs need not to be tested: if we are absolutely sure that x_i is processed correctly, then we may just as well consider the program restricted to the smaller input domain $D \setminus \{x_i\}$, where x_i has been omitted from D.

Remark 2. In the "Nelson" special case (5), we have $Y_i = 1$ if and only if input x_i is failure–causing; otherwise $Y_i = 0$. Therefore, the failure cost parameter g_i is the square root of the *failure probability* of input x_i in this case.

Remark 3. Suppose that the *coefficient of variation* $(Var(Y_i))^{1/2} \,/\, E(Y_i)$ of Y_i can be expected to be nearly constant for all i. This means that the standard deviation $(Var(Y_i))^{1/2}$ of the tester's estimation of the costs caused by input x_i is approximately proportional to the estimated cost value, an assumption that is often justified in practice. (Consider, for example, the effect of currency unit changes to cost estimations: If the costs of a failure are estimated in 1000 \$ units instead of 1 \$ units, a cost estimate of, say, 2000 ± 100 units translates into an estimate of 2 ± 0.1 units.) Then

$$E(Y_i^2) = Var(Y_i) + (E(Y_i))^2 \approx \text{ const } \cdot (E(Y_i))^2 \tag{10}$$

and therefore

$$g_i \approx \text{ const } \cdot E(Y_i).$$

In other words: Inserting in (8) for g_i the expected costs $c_i = E(Y_i)$ caused by input x_i yields a good approximation for the optimal test probabilities t_i.

Remark 4. Let us emphasize that from an exact knowledge of the model parameters g_i, it is not yet possible to determine $Z(\pi)$ by direct computation, i. e., without testing. This is contrary to Nelson's approach (cf. the remarks in Section I).

Remark 5. If the operational distribution *changes* during the use of the program in the application field, say from the values p_i to values $\tilde{p}_i$, new tests for the failure cost estimation are *not* necessarily required. All one needs to do is to change the weights assigned to the failures that have occured during the original test from p_i/t_i to $\tilde{p}_i/t_i$. Summation yields again an unbiased failure cost estimate $\tilde{\phi}_n$ for the program under the new operational distribution (although with sub–optimal variance).

IV. Application

Since the number of inputs in D is usually very large, it is in general impossible to estimate the failure cost parameters g_i separately for each $x_i \in D$. There are essentially two ways to overcome this difficulty:

1. We may use *mathematical formulas* for describing the estimated values g_i. For example, for a program with input variables u, v, w of the type integer, a tester could

possibly estimate the failure cost parameter corresponding to $x = (u, v, w)$ as approximately proportional to the total number of digits required for the representation of the integers u, v, w, argueing that the probability of failures is the higher, the more digits are processed. Then it is reasonable to set

$$g_i = \text{ const } \cdot [\log(|u_i|) + \log(|v_i|) + \log(|w_i|)].$$

In such cases, Theorem 2 yields optimal test distributions that can be realized by well–known techniques of nonuniform random variate generation (see Devroye [4]).

2. A *partition* of the input domain D into disjoint subdomains $D_1, \ldots, D_s$ may be used. This should be done in such a way that all inputs x_i in a fixed subdomain D_j have the same (or nearly the same) estimated failure cost parameter. This is easy in cases where a program performs disjoint functions and each of them can be associated with certain risks (cf. Example 2 at the end of this section). In other cases, such a decomposition may be a more delicate task which requires a compromise between precision of estimation (increasing the number of subdomains to be considered) and economy (restricting this number). After this decomposition step, the operational probability $p^{(j)}$ of each subdomain D_j has to be estimated, such that finally optimal sampling probabilities for the subdomains can de derived.

In the following, we shall explain the second alternative in more detail, and give a practical example. Let D_j $(j = 1, \ldots, s)$ be the partition of D, and let us denote the elements of D_j by $x_{j1}, \ldots, x_{jr_j}$. Element x_{jm} $(m = 1, \ldots, r_j)$ has failure cost parameter $g^{(j)}$ not depending on m, and probability p_{jm} to occur during the operational use, such that

$$p^{(j)} := \sum_{m=1}^{r_j} p_{jm}$$

is the operational probability of the entire subdomain D_j.

Then by Theorem 2, the optimal distribution for failure cost estimation is given by the following probabilities for the elements x_{jm} $(m = 1, \ldots, r_j;\ j = 1, \ldots, s)$:

$$t_{jm} = \frac{p_{jm} g^{(j)}}{\sum_{l=1}^{s} \sum_{m=1}^{r_l} p_{lm} g^{(l)}} = \frac{p_{jm} g^{(j)}}{\sum_{l=1}^{s} p^{(l)} g^{(l)}}.$$

Thus, with

$$t^{(j)} := \sum_{m=1}^{r_j} t_{jm} \qquad (j = 1, \ldots, s)$$

denoting the probability that an input from D_j is chosen for the test, one obtains

$$t^{(j)} = \frac{p^{(j)} g^{(j)}}{\sum_{l=1}^{s} p^{(l)} g^{(l)}}. \tag{11}$$

We have

$$t_{jm} = t^{(j)} \cdot \frac{t_{jm}}{t^{(j)}} = t^{(j)} \cdot \frac{p_{jm}}{p^{(j)}},$$

and $p_{jm}/p^{(j)}$ is obviously just the conditional probability that during operational use, the input is x_{jm}, given that the input comes from subdomain D_j. Therefore one may generate the optimal test distribution in the following way:

- Estimate the values $g^{(j)}$ (failure cost parameter for any input in D_j) and $p^{(j)}$ (total operational probability of subdomain D_j), and compute the values $t^{(j)}$ by (11).
- For each test run, select j according to the distribution $(t^{(1)}, \ldots, t^{(s)})$, and select m according to the (conditional) operational distribution on D_j.

For estimating the risk via (6) and (7), one needs the coefficients

$$p_{jm}/t_{jm} = p^{(j)}/t^{(j)}. \tag{12}$$

These coefficients are constant within each subdomain D_j and can be immediately computed for each $j = 1, \ldots, s$.

More informally speaking, an application of the partitioning variant of the described method requires the following actions:

- The probability for selecting subdomain D_j during the test is changed from the operational probability $p^{(j)}$ to the probability $t^{(j)}$ given by (11). *Within* each subdomain, the (conditional) operational distribution is left unchanged.
- For obtaining a reliability estimate, each failure occuring during the test caused by an input from D_j is weighted by $p^{(j)}/t^{(j)}$.

The values $g^{(j)}$ may be estimated *subjectively*, i.e., by educated guesses, involving failure probability estimations and failure cost estimations (cf. Example 2 below). There is a vast literature on subjective risk assessment (see, e.g., Moore [13], ch. 3); a method that could be especially useful for our purpose is the well–known DELPHI–technique (see Linstone and Turoff [11]).

On the other hand, one could also try to obtain an *objective* estimate of the failure cost parameters $g^{(j)}$. For this purpose, a preliminary sample may be drawn (cf. [20], p. 243). Assume that k_j test cases with inputs from subdomain D_j have been executed in the preliminary sample. Then one can determine the sample mean $\tilde{C}_j$ and the sample variance $\tilde{V}_j$ of the costs assigned to the k_j input/output pairs. Now, in view of (10), $g^{(j)}$ may be estimated by

$$g^{(j)} \approx \sqrt{\tilde{V}_j + \tilde{C}_j^2}.$$

It can happen that both $\tilde{C}_j$ and $\tilde{V}_j$ are zero, such that the resulting estimate $g^{(j)} = 0$ would exclude subdomain D_j from the test. In view of (12), this would lead to technical complications (cf. also Remark 1 in Section III). In this case, one may either (a) pass over again to a subjective estimate of $g^{(j)}$ by a small, but positive value, or (b) use the Bayesian technique developed by Miller et al. [12] to get a positive estimate for the failure probability of an input from D_j: provided that the costs for all failure–causing inputs in

D_j can be expected to be (nearly) equal, an estimate of $g^{(j)}$ is then immediatly obtained as the product of failure probability and costs.

Another possibility to found the estimation of the parameters $g^{(j)}$ on objective information could be the application of *software metrics*. Consider, for example, a special program function, related to a subdomain D_j. There is a specific part of the code written exclusively for the realization of this function, and another part of the code "shared" by this and other functions. For both parts, diverse software metrics can be evaluated (see, e.g., Beizer [1], ch. 7). Now, correlations between such metrics and empirically observed failure rates and failure severities in similar software products or previous versions of the same product may be investigated. Using statistical regression techniques, one may then try to predict the current parameter $g^{(j)}$ from the measured metrics of the function under consideration.

Clearly, each of the above–mentioned approaches for the estimation of the failure cost parameters has its particular limitations. If necessary, the approaches may be combined; such a combination (for example a weighted average) can be expected to lead to fairly good estimates.

The application of the method is illustrated by the following example:

Example 2. We consider a program that has three different functions: function 1 does nonnumerical processing, function 2 does numerical processing, and function 3 is a special safety–critical function. The functions 1 – 3 are evoked when inputs come from subdomains D_1, D_2, and D_3 of the input domain, respectively. Inputs that are invalid with respect to the specification are collected in a subdomain D_4.

First of all, we assign *costs* to special types of failures:

Subdomain D_1 (nonnumerical processing):

$$\text{costs} = \begin{cases} 0 & \text{for correct execution,} \\ 1 & \text{for incorrect execution.} \end{cases}$$

Subdomain D_2 (numerical processing):

$$\text{costs} = \begin{cases} 0 & \text{for correct output,} \\ 0.5 & \text{for a small deviation from the correct output,} \\ 2 & \text{for a large deviation from the correct output.} \end{cases}$$

Subdomain D_3 (safety–critical function):

$$\text{costs} = \begin{cases} 0 & \text{for correct execution,} \\ 100 & \text{for incorrect execution.} \end{cases}$$

Subdomain D_4 (invalid inputs):

$$\text{costs} = \begin{cases} 0 & \text{for correct error messages,} \\ 0.5 & \text{for incorrect behavior.} \end{cases}$$

Next, we have to determine the failure cost parameters $g^{(j)}$. Remember that we assume that all inputs $x_i \in D_j$ have the same failure cost parameter, hence one can compute $g^{(j)}$ as $(E(Y_i^2))^{1/2}$ for an arbitrary input $x_i \in D_j$. The random variable Y_i stands for the costs caused by input x_i, so probabilities for certain cost values are required for the application of the model.

(a) Let us first assume a situation where failure probabilities for each subdomain as well as the operational distribution can be estimated — by some means or other. From the estimated failure probabilities, one immediately computes the failure cost parameters:

Subdomain D_1: For a single input from D_1, we estimate

$$P\{\text{ incorrect execution }\} = 10^{-4},$$

therefore

$$g^{(1)} = \sqrt{(1-10^{-4}) \cdot 0^2 + 10^{-4} \cdot 1^2} = 0.01.$$

Subdomain D_2: For a single input from D_2, we estimate

$$P\{\text{ small deviation }\} = 10^{-2}, \quad P\{\text{ large deviation }\} = 10^{-3},$$

therefore

$$g^{(2)} = \sqrt{(1-10^{-2}-10^{-3}) \cdot 0^2 + 10^{-2} \cdot (0.5)^2 + 10^{-3} \cdot 2^2} = 0.0806.$$

Subdomain D_3: For a single input from D_3, we estimate

$$P\{\text{ incorrect execution }\} = 10^{-5},$$

therefore

$$g^{(3)} = \sqrt{(1-10^{-5}) \cdot 0^2 + 10^{-5} \cdot 100^2} = 0.3162.$$

Subdomain D_4: For a single input from D_4, we estimate

$$P\{\text{ incorrect behavior }\} = 2 \cdot 10^{-2},$$

therefore

$$g^{(4)} = \sqrt{(1-2 \cdot 10^{-2}) \cdot 0^2 + 2 \cdot 10^{-2} \cdot (0.5)^2} = 0.0707.$$

The operational distribution is estimated as follows: Functions 1 – 3 are used in 90 %, 7 % and 2 % of all program runs, respectively. In 1 % of all program runs, the input is invalid.

Table 2 shows the computation of the optimal test distribution $(t^{(1)}, \ldots, t^{(4)})$. The last column contains the cumulated values $t_{cum}^{(j)} = \sum_{l=1}^{j} t^{(l)}$. A random test input is determined from Table 2 as follows:

j	$p^{(j)}$	$g^{(j)}$	$p^{(j)}g^{(j)}$	$t^{(j)}$	$t^{(j)}_{cum}$
1	0.90	0.0100	0.00900	0.415	0.415
2	0.07	0.0806	0.00564	0.260	0.675
3	0.02	0.3162	0.00632	0.292	0.967
4	0.01	0.0707	0.00071	0.033	1.000

Table 2. Computation of the optimal test distribution in Example 2, assumptions (a).

- Draw a uniformly distributed real random number X between 0 and 1;
- determine that subdomain in Table 2 the value $t^{(j)}_{cum}$ of which is the smallest occuring value $t^{(j)}_{cum}$ larger than X;
- select a test input from this subdomain according to the operational distribution restricted to the subdomain.

The process is repeated n times, where n is the pre–specified number of test cases. The failure cost estimate ϕ_n has initial value zero. Whenever a failure occurs at an input from a subdomain D_j, the value

$$\frac{1}{n}\,\frac{p^{(j)}}{t^{(j)}}\cdot C$$

is added to ϕ_n, where C are the costs assigned to the respective failure.

(b) Now let us assume, contrary to (a), that we do *not* know the operational distribution, and that we have no information whatever on the failure probabilities. Then all we can do is to apply Laplace's principle of indifference (see, e.g., Smithson [19], ch. 3.3; cf. also Miller et al. [12]), assigning equal probabilities to events that we cannot rank with respect to their likelihood:

$$p^{(1)} = \ldots = p^{(4)} = 0.25,$$

and

$$P\{\text{ incorrect behavior }\} = \text{ const } = x$$

for each subdomain D_j, where x is an unknown probability. For subdomain D_2, we set (again by the principle of indifference):

$$P\{\text{ small deviation }\} = P\{\text{ large deviation }\} = x/2.$$

As in case (a), the failure cost parameters can be computed, which yields now

$$g^{(1)} = \sqrt{x}, \quad g^{(2)} = 1.4577\sqrt{x}, \quad g^{(3)} = 100\sqrt{x}, \quad g^{(4)} = 0.5\sqrt{x}.$$

If this is inserted into (11), $\sqrt{x}$ cancels out, hence one can set $x = 1$ without loss of generality. Thus, we end up with Table 3 instead of Table 2. The costs of a failure at an input from D_j are now to be weighted by $p^{(j)}/t^{(j)} = 0.25/t^{(j)}$.

j	$p^{(j)}$	$g^{(j)}$	$p^{(j)}g^{(j)}$	$t^{(j)}$	$t^{(j)}_{cum}$
1	0.25	1.000	0.250	0.010	0.010
2	0.25	1.458	0.365	0.014	0.024
3	0.25	100.000	25.000	0.971	0.995
4	0.25	0.500	0.125	0.005	1.000

Table 3. Computation of the optimal test distribution in Example 2, assumptions (b).

The large differences between the values $t^{(j)}$ in (a) and (b) show the importance of an adequate estimation of the operational distribution. $\bowtie$

V. Conclusions

We have presented a method for computing efficient test distributions for the estimation of risk, i.e., expected failure costs. Our approach is, in a certain sense, "semi–Bayesian": in order to obtain precise estimates, we use (probabilistic) prior information on costs caused by different inputs, but this information is only applied for reducing the variance of the estimation and does not influence its expected value: the estimate remains unbiased even if the prior assumptions are unrealistic.

In this article, we have judged test distributions exclusively under the aspect of their ability for reliability estimation resp. failure cost estimation and not under the aspect of their efficiency with respect to certain *testing* aims such as fault detection. In [7], different measures of *testing efficiency* (based on concepts by Gourlay [6], Hamlet [8], [9], and Weyuker et al. [23]) will be compared, and it will be shown that the distribution (8) can also be expected to be superior to the operational distribution with respect to *testing* aims. For the sake of brevity, this point is not discussed here.

Let us now consider some possible objections to the presented method. First of all, the method requires information on the operational distribution, and one might argue that it is difficult to estimate that distribution. This is admittedly true; however, information on the operational distribution is indispensible *whenever* reliability or risk are to be estimated. A software developer with a restricted budget for testing (that is to say, almost *every* software developer) should try to get such information also for another reason: his testing activity may fail to be cost–effective, if he concentrates on input constellations that are rarely or never used, instead of thoroughly testing the relevant ones. One may hope that techniques for the estimation of operational profiles will be improved in the future (cf. the remarks by Weiss and Weyuker in [21], Section VII). Some practical examples for the estimation of operational profiles for commercial software can be found in Sherer [18].

A second difficulty might lie in the estimation of the parameters g_i. The simplest escape from this difficulty (and as well from the difficulty of missing operational distribution estimates) is an application of the principle of indifference, as demonstrated in Example 2, part (b). Nevertheless, we think that often much more can be done. An experienced tester should be able to make educated guesses at the parameters g_i. If not enough information is available, the use of preliminary samples or of software metrics (as described in Section IV) may be helpful. The reader should notice that during the real use of a software product, the failure cost parameter can be determined in a quite objective way from failure statistics recording additionally the costs each failure has caused. This feedback enables a *learning effect*: Developers and testers can improve their skill to make estimations by comparing previous estimates with the real outcomes. Besides of the benefit for the testing approach described in this article, such a learning process has still other positive effects in so far as it increases the competence to judge the failure risks of different functions of a software product in a quantitative way.

In any case, we have to stress again the point that a *poor* estimate of the values g_i does *not* lead to biased failure cost predictions; it only makes the variance of the prediction less favorable than it could be. Even rough estimates of the values g_j may yield improvements, compared to the *very* rough estimate that is implicit in testing with the operational distribution, namely the estimate that all inputs have the same failure probability and entail the same costs.

A third criticism might concern the problem of partitioning the input domain D into the disjoint subsets D_j. As mentioned in Section IV, such a partition should be based on the main functions of the software under test. Of course, a subdomain is not always homogenous with respect to the failure cost parameters of its elements, so an overall estimate can only be considered as an approximation. Moreover, sometimes functions overlap; in these cases, a pragmatic solution has to be found by assigning each intersection of two or more function domains to one of the involved functions. Perhaps a future extension of the presented method (for example the introduction of a second estimation step refining the first, coarse classification) can overcome these drawbacks.

In some cases, the test distribution obtained by the application of our method may be very unbalanced: Suppose, for example, a situation where one or several functions are that risky that their optimal test probabilities dominate the test probabilities of other, less risky functions by a factor of 10^4 or more. Then the test would with a high probability concentrate exclusively on the risky functions. Provided that the reliability test is to be started immediately, this is in fact advisible. Nevertheless, the advantages of a risk–oriented test can be better exploited: One may try to reduce, in a first step, the failure probabilities of the risky functions by the application of *formal verification techniques*. If formal verification turns out to be successful, the increased confidence in the correct behavior of the risky functions expresses as a decrease of the failure probability estimates. Then our method will produce a more balanced optimal test distribution. As an alternative to formal verification, also special pre–tests for risky functions can reduce the failure probability estimates.

A particular problem for reliability estimation is a situation where the test *reveals no failures at all*. On the other hand, just this is the event one hopes for in the *validation* phase of the testing process. If that event occurs, then our failure cost estimate (as well as the unreliability estimate $1 - \hat{R}$ obtained from Nelson's approach described in Section I) is *zero*. A more precise estimate in such a situation can be given if Bayesian prior assumptions on the failure probability intrude not only into the sampling distribution, but also into the expected value of the resulting estimate itself. Miller et al. [12] deal explicitly with reliability estimation in this case. [3] It should be possible to generalize their Bayesian approach to the estimation of *risk*, but this is a topic for further research. Let us confine ourselves here to the observation that even if a reliability test has revealed no failures, it makes a difference *under which test distribution* this result was achieved. The significance of "Test o. k." is in general higher, when the variance of the used reliability (or failure cost) estimator is lower.

An obvious field of application for our approach is *safety–critical software*, since here the difference between low–cost and high–cost faults is especially drastic. We do not assert, however, that our approach can overcome the principal limitations to reliability estimation for ultra–reliable software (failure rate $< 10^{-7}$ per hour), as they were exposed by Butler and Finelli in their lucid paper [3].

Finally, let us remark that reliability testing is an essential component of special modern development methods such as *Cleanroom* that put more emphasis on error–prevention in early phases than on fault detection in late phases, and consider the testing process rather as a *certification* of program reliability and quality (see, e. g., [17]). We think that in a Cleanroom development context, the presented approach would possibly be particularly useful in so far as it enables the selection of test cases with special regard to the *consequences* of possible failures. In this way, reliability certificates could gain a higher significance from an economic point of view.

References

[1] B. Beizer, *Software Testing Techniques*, 2nd edition, New York: Van Nostrand Reinhold, 1990.

[2] B. Boehm, "Software risk management", *Proc. ESEC*, Warwick, UK, Sept. 1989, pp. 1–19.

[3] R. W. Butler and G. B. Finelli, "The infeasibility of quantifying the reliability of life–critical software", *IEEE Trans. Software Eng.*, vol. SE-19, pp. 3–12, Jan. 1993.

[4] L. Devroye, *Non–Uniform Random Variate Generation.* Berlin, Heidelberg, New York: Springer, 1986.

[5] W. D. Ehrenberger, "Combining probabilistic and deterministic verification efforts", *Proc. SAFECOMP '92* (ed. H. Frey), pp. 299–304, Oxford: Pergamon Press, 1992.

[3] As another approach to risk judgment in cases without failure observations, let us mention Ehrenberger [5], where stratified sampling is used for deciding whether the risk of a program is, at a certain level of confidence α, below a given limit. This is the viewpoint of classical statistical test theory (in contrast to statistical estimation), as it was already advocated in Hamlet's probable correctness theory for computer software [8].

[6] J. S. Gourlay, "A mathematical framework for the investigation of testing", *IEEE Trans. Software Eng.*, vol. SE-9, pp. 686–709, Nov. 1983.

[7] W. J. Gutjahr, "Probabilistic software test efficiency measures and optimal test data selection", in preparation.

[8] R. G. Hamlet, "Probable correctness theory", *Information Processing Letters* 25, pp. 17–25, 1987.

[9] R. G. Hamlet, "Theoretical comparison of testing methods", *Proc. Third Symposion on Testing, Analysis, and Verification*, Key West, Florida, Dec. 1989, pp. 28–37.

[10] M. H. Kalos and P. A. Whitlock, *Monte Carlo Methods*, vol. I. New York: Wiley, 1986.

[11] H. A. Linstone and M. Turoff, *The Delphi Method*, London: Addison–Wesley, 1975.

[12] K. W. Miller, L. J. Morell, R. E. Noonan, S. K. Park, D. M. Nicol, B. W. Murrill, and J. M. Voas, "Estimating the probability of failure when testing reveals no failures", *IEEE Trans. Software Eng.*, vol. SE-18, pp. 33–43, Jan. 1992.

[13] P. G. Moore, *The Business of Risk*, Cambridge: Cambridge University Press, 1993.

[14] E. C. Nelson, "A statistical basis for software reliability assessment", TRW-SS-73-03, March 1973.

[15] E. C. Nelson, "Estimating software reliability from test data", *Microelectronics and Reliability*, vol. 17, pp. 67–74, 1978.

[16] C. V. Ramamoorthy and F. B. Bastani, "Software Reliability — Status and Perspectives", *IEEE Trans. Software Eng.*, vol SE-8, pp. 354–371, July 1982.

[17] R. W. Selby, V. R. Basili, and F. T. Baker, "Cleanroom software development: an empirical investigation", *IEEE Trans. Software Eng.*, vol. SE-13, pp. 1027–1037, Sept. 1987.

[18] S. A. Sherer, *Software Failure Risk*. New York, London: Plenum Press, 1992.

[19] M. Smithson, *Ignorance and Uncertainty*, New York: Springer, 1988.

[20] Th. A. Thayer, M. Lipow, and E. C. Nelson, *Software Reliability*. Amsterdam: North–Holland, 1978.

[21] S. N. Weiss and E. J. Weyuker, "An extended domain–based model of software reliability", *IEEE Trans. Software Eng.*, vol. SE-14, pp. 1512-1524, Oct. 1988.

[22] E. J. Weyuker and T. J. Ostrand, "Theories of program testing and the application of revealing subdomains", *IEEE Trans. Software Eng.*, vol. SE-6, pp. 236–245, May 1980.

[23] E. J. Weyuker, S. N. Weiss, and D. Hamlet, "Comparison of program testing strategies", *Proc. Symp. on Testing, Analysis, and Verification*, Victoria, British Columbia, Oct. 1991, pp. 1–10.

APPENDIX

Proof of Theorem 2: We start with considering the functionality F_k as *fixed*. This yields

$$V_k = Var(\phi_n) = Var\left(\frac{1}{n}\sum_{\nu=1}^{n} S_\nu\right)$$

as the variance of the cost estimator ϕ_n. One obtains immediately

$$V_k = \frac{1}{n^2}\sum_{\nu=1}^{n} Var(S_\nu) = \frac{1}{n^2}\cdot n\ Var(S_1) = \frac{1}{n}\ Var(S_1).$$

Notice that Var can be interchanged with $\sum$, since the random variables $S_1, \ldots, S_n$ are independent: For fixed functionality F_k, the variable S_ν is a function of the input x_{i_ν}, therefore independence of S_ν ($\nu = 1, \ldots, n$) follows from independence of x_{i_ν} ($\nu = 1, \ldots, n$).

Using Theorem 1, we find:

$$Var(S_1) \;=\; E(S_1^2) - (E(S_1))^2 \;=\; \sum_{i=1}^{N} t_i\left(\frac{p_i}{t_i}c_{ik}\right)^2 - Z_k^2 \;=\; \sum_{i=1}^{N}\frac{p_i^2 c_{ik}^2}{t_i} - Z_k^2.$$

Now we compute the weighted average over all possible functionalities:

$$\sum_{k=1}^{M} q_k V_k \;=\; \frac{1}{n}\sum_{k=1}^{M} q_k \sum_{i=1}^{N}\frac{p_i^2 c_{ik}^2}{t_i} - \frac{1}{n}\sum_{k=1}^{M} q_k Z_k^2. \tag{13}$$

The second expression on the right hand side of (13) does not depend on $t = (t_1, \ldots, t_N)$. Hence minimization of $\sum_{k=1}^{M} q_k V_k$ for fixed n is equivalent to minimization of

$$\sum_{k=1}^{M} q_k \sum_{i=1}^{N}\frac{p_i^2 c_{ik}^2}{t_i} \;=\; \sum_{i=1}^{N}\frac{p_i^2}{t_i}\sum_{k=1}^{M} q_k c_{ik}^2 \;=\; \sum_{i=1}^{N}\frac{p_i^2}{t_i}\,E(Y_i^2) \;=\; \sum_{i=1}^{N}\frac{p_i^2 g_i^2}{t_i}.$$

With

$$\alpha_i := (p_i g_i)^2,$$

this yields the optimization problem

$$\sum_{i=1}^{N}\frac{\alpha_i}{t_i} \to \min \tag{14}$$

subject to the restrictions

$$\sum_{i=1}^{N} t_i = 1, \qquad t_i \geq 0 \qquad (i = 1, \ldots, N). \tag{15}$$

Because of the convexity of the functions $1/t_i$, the objective function $\sum_{i=1}^{N}\alpha_i/t_i$ is also convex and possesses therefore a unique minimum on the simplex $\{t : \sum t_i = 1\}$.

By introducing the Lagrange function

$$L = \sum_{i=1}^{N}\frac{\alpha_i}{t_i} + \lambda\left(\sum_{i=1}^{N} t_i - 1\right),$$

we can solve (14) – (15), and obtain

$$t_i = \frac{\sqrt{\alpha_i}}{\sqrt{\alpha_1} + \ldots + \sqrt{\alpha_N}} \qquad (i = 1, \ldots, N)$$

as the optimal values for $t_1, \ldots, t_N$. □