Software Dependability Evaluation Based on Markov Usage Models

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

Abstract: A general technique for computing optimal state transition probabilities for software tests, based on a Markov usage model, is developed. The optimization criterion is maximum precision of unbiased dependability estimates derived from the test results. Three different dependability measures are considered: (i) risk, (ii) safety, and (iii) reliability. As input, pre–information on failure probabilities and losses in case of failure related with single operations is used. The optimization itself is done by means of a numerical procedure which is fast because of the convexity of the underlying stochastic optimization problem. The procedure can be improved by the construction of a distribution with a common lower bound on state transition probabilities; this distribution may also be used in the more general context of structural statistical testing of software.

Keywords: Dependability, importance sampling, Markov chains, optimization, reliability, risk, safety, software usage models, software testing.

1 Introduction

1.1 Markov Software Usage Models

In the last years, there is a growing number of publications on Markov–type software usage models (see, e.g., [25], [26], [27], [22], [1], [24], [18]). The reason seems to be that models of this type have turned out as flexible, easily understandable descriptions of the *operational profiles* of given programs or software systems, providing both a wide range of applications and a potential for deep formal analysis.

A software usage model characterizes the (estimated) distribution of possible uses of a given program or software system in its intended environment. Such a model is required whenever unbiased estimates of quantities like reliability or risk are to be derived from the results of *tests*. The testing method allowing such estimates is random testing with suitably chosen input distributions, and the most direct way to achieve the goal of unbiased estimation consists in choosing the operational profile, as described by the usage model, as the input distribution (cf. [14], [22] or [2]).

In its simplest form, a (partial) usage model is already specified by a table assigning to each program function the corresponding probability of a call. (To complete the model, it has to be supplied with a component that describes the selection of inputs for each program function, a problem which is usually solved in a pragmatic way.) However, the usage structure of many programs includes *time dependencies* of function calls. Then, from a probabilistic point of view, the usage model will rather be based on a stochastic process than on a single discrete random variable. Markov chains are, as stated above, very convenient model processes for a large spectrum of applications.

In this article, we consider three different quantities to be estimated from test results, namely (i) *risk*, (ii) *safety*, and (iii) *reliability*. These quantities are partial aspects of the more general notion of *dependability* (see [11]), which additionally includes other features like availability and security. We think that the presented approach can also be extended to the last–mentioned properties, but exclude these topics for the sake of brevity.

1.2 Non–Operational Profiles

We have mentioned that most usage–model–based attempts to predict dependability properties of a given program rely on testing with the operational profile, which is (hopefully) specified adequately by the usage model. However, as it will be argued, this approach has severe drawbacks. It will be shown that a software usage model can be applied for generating *non–operational* test cases as well, and, which is of vital interest, the result of executing these tests can nevertheless be used to derive unbiased estimates of the interesting quantities in a statistically sound way.

Let us start by recapitulating the limitations of testing with the operational profile. In the context of the Markov usage model literature, these limitations have been formulated very clearly by Walton, Poore and Trammell ([22], pp. 105 – 106): They state that it is often important to ensure the exercise of specific operations of the software irrespective of their usage probabilities. For example, there may be operations of high criticality (i.e., high loss in case of failure); nevertheless, if these operations are infrequently executed in real applications, they are possibly not tested at all during operational–profile testing. Another example are newly implemented operations which have a high prior failure probability and should therefore be given a good chance of being tested, no matter whether they are frequently or infrequently used. Walton et al. suggest modifications and extensions to operational–profile testing in order to overcome the mentioned drawbacks, but, as they admit, the suggested measures are not able to allow unbiased estimates of, say, the reliability of the software under test.

The aim of the present article is to develop a general method of sampling test cases from a Markov software usage model, such that

(1) test results can still be used to compute unbiased estimates of a dependability measure under consideration (risk, safety or reliability), although

(2) the distribution of test cases is shifted from the operational distribution to a test profile designed for exploiting prior information on possibly error–prone or critical operations.

Moreover, a technique shall be presented for computing *optimal* shifts of operational transition probabilities in the considered Markov usage model, where the criterion of optimality is minimal imprecision (statistically speaking: minimal variance) of the resulting estimate.

For some special cases, the suggested approach has already been indicated in other recent publications: the simple, non–Markovian case of *risk* estimation is treated in [6]; a generalization to Markov models under specific model assumptions (again for risk) is shortly outlined in [7], and some aspects of *reliability* estimation are discussed in [8]. The present paper develops the approach within a unified framework in full generality (section 2 – 3), specifies it, for two different dependability models, to the three particular dependability measures, including *safety* (section 4), and elaborates an efficient procedure for the iterative computation of optimal transition probabilities (section 5). In section 6, we develop an algorithm for computing a test distribution guaranteeing for each operation a certain minimal probability of being executed; this algorithm supports the procedure of section 5, but is also of interest for *structural statistical testing* ([19], [20]) outside the context of our approach. The final section 7 contains a short discussion of the results.

2 Formal Prerequisites

2.1 Formal Description of a Markov Model

First of all, a definition of a *Markov usage model* will be given (cf. also [22] and [26]): By a Markov usage model, we understand a Markov chain with a unique initial state, symbolizing program invocation, a unique final state, symbolizing program termination, and other states symbolizing intermediate usage or processing states of the program under consideration. As customary, the Markov chain will be represented by a directed graph $G = (V, A)$ and a function $p : V \times V \to [0, 1]$ with the following properties:

- $V = \{1, \ldots, n\}$ is a set of *nodes*, representing the program states (e.g., program invocation, program termination, input/output screens).
- A is a set of *arcs*, representing state transitions which always correspond to specific operations of the program. An arc from state i to state j is denoted by the ordered pair (i, j). Multiple arcs between i and j are not allowed.
- $p(i, j)$ is the *transition probability* from state i to state j, if (i, j) is an arc. Otherwise, we set $p(i, j) = 0$. The transition probabilities satisfy the conditions $0 \le p(i, j) \le 1$ and

$$\sum_{j=1}^{n} p(i, j) = 1 \quad (i = 1, \ldots, n).$$

The values $p(i, j)$ can be represented in a comprehensive form by a matrix $(p(i, j))_{i,j}$.

It is always supposed that state 1 is the initial state and state n is the final state. State n is assumed to be an *absorbing state*, i.e., it cannot be left anymore: $p(n, n) = 1$ and $p(n, j) = 0$ for $j \neq n$. Furthermore, we assume that each node $i \in V$ is *reachable* from node 1, i.e., there is a directed path in G from node 1 to node i.

Fig. 1 shows a small example of a Markov usage model in its graph representation. Here, the states (except the initial and the final state) correspond to input/output screens. The operations of the program can be assigned to the arcs: e.g., the execution of "insertion" corresponds to the arc (3,2).

A particular use of the program corresponds to a *path* $x = (x_0, x_1, \ldots)$ of the Markov chain, where $x_0 = 1$ (initial state), and $x_i \in V$ is the state after the ith transition. In Fig. 1, for example, $x = (1, 2, 3, 2, 5, 2, 6, 6, 6, 6, \ldots)$ represents a program use where first an insertion is performed, then the print function is called, and after that the program is terminated. Recalling that n is an absorbing state, we can simply write $(x_0, x_1, \ldots, n)$ instead of $(x_0, x_1, \ldots, n, n, n, \ldots)$, e.g.: $(1, 2, 3, 2, 5, 2, 6)$ for the previously considered path.

If the path results from random transitions in the Markov chain (as it is the underlying assumption of the Markov usage model), the states are random variables, forming a stochastic process. They shall then be written in the form $X = (X_0, X_1, \ldots)$, where capitalization denotes randomness.

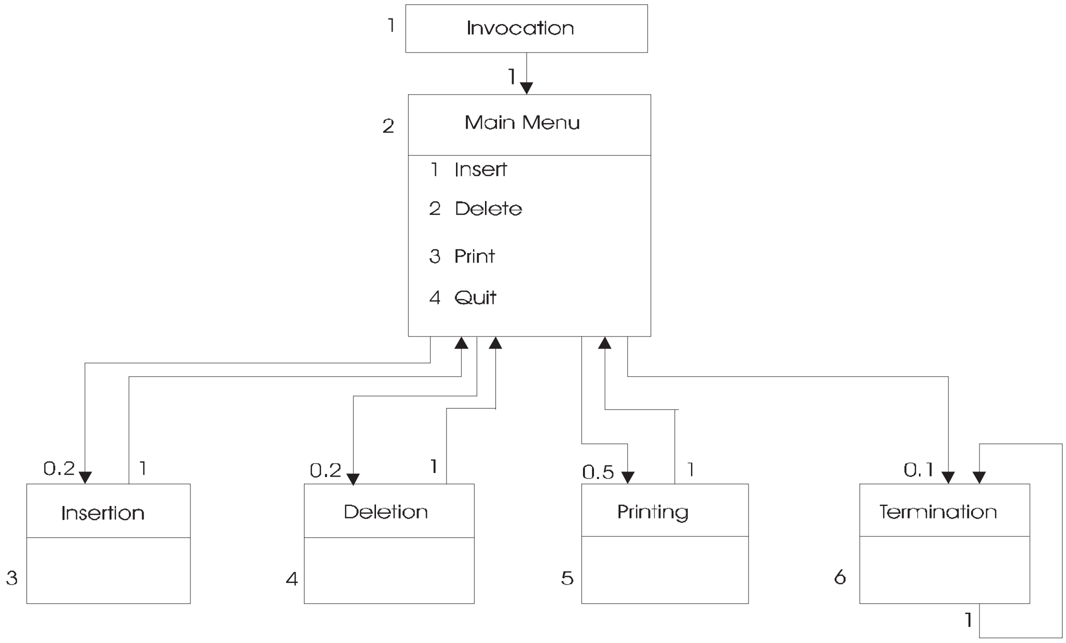

Fig. 1. A small Markov usage model ($n = 6$). A self–loop has been added to the "termination" state in order to obtain a complete Markov chain. The values assigned to the arcs are transition probabilities.

2.2 Usage Domain

By D, we denote the set of all possible *uses* of the given program π, and we call D the *usage domain* of π. At the finest granularity level, each input (resp. each sequence of inputs, if there are several inputs during a program execution) can be considered as a distinct use of π; then, D gets identical to the so–called *input domain* of π. In many applications, a coarser granularity level suffices for a suitable description of the possible program uses. Within our context, we suppose that the granularity level of the applied Markov usage model is already sufficiently fine for this purpose (as to the granularity of the usage model, cf. [22]): The uses of π will be identified with the possible paths of the Markov chain, which means that the usage domain D is simply the set of all possible paths x from state $x_0 = 1$ to the absorbing state n. The set D contains either a finite or, in the presence of loops, a countably infinite number of elements.

In the following sections, it will be assumed that the failure behavior of the program π only depends on the program use x and not (directly) on the program input. (In Hamlet's and Taylor's [9] terminology, this means that the subdomains of the input domain corresponding to the uses x are *homogeneous*. Weyuker and Ostrand [23] call such subdomains *revealing*.) Of course, this is a simplification: If two inputs i_1,i_2 correspond to the same program use (i.e., the same path in the Markov chain), one of them may be processed correctly, while the other may be failure–causing. In the Conclusion it shall be argued that the condition above is not an essential restriction for the application of our approach.

The *operational profile* is denoted by P. Since P describes the distribution of possible uses, it is a probability distribution on the usage domain D. In the present context, P is specified by the Markov usage model. To be more explicit: For each fixed path $x = (x_0, x_1, \ldots)$, the probability $P\{X = x\}$ that, under P, the current path X is identical to x, is given as the product of the transition probabilities between the states of x:

$$P(x) := P\{X = x\} = \prod_{k \geq 1} p(x_{k-1}, x_k). \tag{1}$$

In Fig. 1, for example, the path $x = (1, 2, 3, 2, 5, 2, 6)$ has a probability

$$P(x) = 1 \cdot 0.2 \cdot 1 \cdot 0.5 \cdot 1 \cdot 0.1 = 0.01.$$

2.3 Uncertainty on Program Behavior

For representing prior uncertainty in dependability estimation, we use a formal framework developed by Eckhardt and Lee [3], assuming that the program π under consideration has been selected from a conceptual population Π. This population consists of all program versions that would (possibly) be developed based on a fixed given specification. The actual program π is then one element of Π, and it can be imagined that it has been selected *randomly* from Π. The probability distribution of programs π in population Π will be denoted here by Q. [1] For describing relevant features of Q, specific models are required. Examples will be presented in section 4.

The motivation for considering π rather as a random variable than as a fixed, deterministic object is that the behavior of the actual program π is not known before the test. Assuming this behavior to be known would make any dependability test unnecessary. The prior information available before test is subject to uncertainties which can be quantified in a probabilistic manner by following the indicated approach. [2]

2.4 Cost Functions

Now we are in the position to define a *cost function* $c : D \times \Pi \to R$, assigning to each use $x \in D$ and each program $\pi \in \Pi$ a value $c(x, \pi)$ that represents the *costs by failure* of an execution of program π at use x. The interpretation of these costs depends on the special quantity to be estimated (see below). Let us recall the assumption that the failure behavior depends only on the use x, such that $c(x, \pi)$ is indeed a function of x and not of the special program input. By

[1] For the sake of consistency with [6], our notation is switched, compared to that in [3]: The authors denote the operational profile by Q and the distribution on Π by P.

[2] Some readers might find it astonishing that π, the given program to be tested, is considered as "unknown" in this model. Note, however, that neither the source code nor the machine code of π are the aspects of interest in our context, but its dependability behavior, and the last is unknown indeed. In [6], the difference between code and behavior has been made more explicit by distinguishing program π from F, its functionality. However, as soon as it is clear that we are only interested in the behavioral aspects of π, we may stick to the notation π as well.

E_P, we denote the expectation with respect to the probability distribution P (the operational profile). The symbols x and X stand for fixed and random uses, respectively. E_P always "acts" on the random variable X.

As our main applications, we consider the cost functions corresponding to three different dependability measures:

(i) *Risk:*

(Software) risk is expected loss by software failure, either during a specified time or during a single program execution (see [4], [17]). Here, we choose the definition of risk based on a single program execution. Let loss (x, π) denote the loss caused by failure if π is executed at use x. If π executes correctly at use x, we set loss $(x, \pi) = 0$. Now, for

$$c(x, \pi) := \text{loss}\,(x, \pi), \tag{2}$$

we have $E_P(c(X, \pi)) = \text{risk}\,(\pi)$.

(ii) *Safety:* Safety is defined as the probability that no *severe* failure occurs during a program execution (or during a specified time), see, e.g., [11]. Let $C \geq 0$ be the value of loss above which a failure is considered as severe. Now we define the cost function as follows:

$$c(x, \pi) = \begin{cases} 0, & \text{if loss}\,(x, \pi) \leq C, \\ 1, & \text{else.} \end{cases} \tag{3}$$

Then $E_P(c(X, \pi)) = P\{\text{loss}\,(X, \pi) > C\} = 1 - \text{safety}\,(\pi)$.

(iii) *Reliability:*

Reliability is the probability that no failure at all occurs during a program execution (or during a specified time). Here, we set

$$c(x, \pi) = \begin{cases} 0, & \text{if loss}\,(x, \pi) = 0, \\ 1, & \text{else.} \end{cases} \tag{4}$$

Obviously, this is the special case $C = 0$ of (ii). We obtain

$$E_P(c(X, \pi)) = P\{\text{loss}\,(X, \pi) > 0\} = 1 - \text{reliability}\,(\pi).$$

In view of (2) and (3), it could be argued that, formally, (un)safety is simply a special case of risk, obtained by choosing in (2) a particular 0–1–valued function loss (x, π). In our model, however, it is not possible to obtain (ii) by specialization from (i), since it shall be assumed in section 4 that the overall loss value, loss (x, π), is composed from loss values assigned to single operations in a specific, natural way, which does not allow arbitrary definitions of the function loss (x, π). So the treatment of safety becomes essentially different from that of risk. On the other hand, (iii) is indeed a special case of (ii).

3 Changing the Test Profile

The simplest way of obtaining unbiased estimates for risk(π), safety(π) and reliability(π) is *operational profile testing*: Let $c(x,\pi)$ be the cost function under consideration. Then, an unbiased estimate of $E_P(c(X,\pi))$ is obtained by selecting X according to the operational profile P, performing a test of π with path X, and determining $c(X,\pi)$ from the test result. In practical applications, this process is always iterated (cf., e.g., [22]): Draw N test paths $X^{(1)},\ldots,X^{(N)}$ according to P, perform tests with these paths and determine the values $c(X^{(1)},\pi),\ldots,c(X^{(N)},\pi)$. Then, the arithmetic mean

$$\bar{c} = \frac{1}{N}\left[c(X^{(1)},\pi)+\ldots+c(X^{(N)},\pi)\right] \tag{5}$$

is an unbiased estimate of $E_P(c(X,\pi))$, since $E_P(\bar{c}) = E_P(c(X,\pi))$. The higher the sample size N, the lower is the variance of S, i.e., the better is the precision of the estimate. For example, in the *safety* case, where $c(x,\pi)=0$ if loss $(x,\pi)\le C$ and 1 else, $\bar{c}$ is the relative frequency of single tests with a loss (i.e., failure severity) larger than C, and this relative frequency is obviously an unbiased estimate of the unsafety, $1-$ safety (π). However, the precision of this estimate is still low: Observe that severe failures can be considered as rare events, so in most cases safety(π) will be overestimated by a result of $\bar{c}=1$, while in some (rare) cases it will be drastically underestimated.

In order to overcome this drawback, we *change the test profile* from the operational profile to another, suitably chosen profile T. To preserve unbiasedness, the change has to be compensated by assigning the weights $P(X)/T(X)$, called *likelihood ratios*, to the test results. In Rare Event Simulation, this technique is called *Importance Sampling* (see, e.g., [15], [5] or [10]). Thus, instead of $c(X,\pi)$, the estimator

$$S := \frac{P(X)}{T(X)}\cdot c(X,\pi) \tag{6}$$

is used, where X is now selected according to T. Provided that $T(x)>0$ for each x with $P(x)>0$, the expression S is an unbiased estimator for $E_P(c(X,\pi))$, since

$$E_T(S) = \sum_x T(x)\cdot\frac{P(x)}{T(x)}\,c(x,\pi) = \sum_x P(x)\,c(x,\pi) = E_P(c(X,\pi)), \tag{7}$$

where the operator E_T denotes the expectation with respect to distribution T. As before, the variance of the estimate can be decreased by iterated testing with paths $X^{(1)},\ldots,X^{(N)}$ selected according to T and taking the arithmetic mean $\bar{S}$ of the resulting S–values.

For practical reasons, only special test profiles T will be considered here, namely profiles obtained from the original Markov usage model by a *change of the transition probabilities*: The values $p(i,j)$ are shifted to (other) transition probabilities $t(i,j)$, satisfying the condition that $t(i,j)>0$ exactly for those (i,j) for which $p(i,j)>0$. Then, the probability of a path $x=(x_0,x_1,\ldots)$ to be tested is

$$T(x) := T\{X=x\} = \prod_{k\ge 1} t(x_{k-1},x_k), \tag{8}$$

such that

$$S = \prod_{k\ge 1}\frac{p(X_{k-1},X_k)}{t(X_{k-1},X_k)}\cdot c(X,\pi).$$

In the same way as P is given by the matrix $(p(i,j))_{i,j}$ of the operational transition probabilities $p(i,j)$, the test profile T is given by the matrix $(t(i,j))_{i,j}$ of the test transition probabilities $t(i,j)$. In the sequel, P and T will often be identified with their matrix representations.

The purpose of using S with test profile T instead of $c(X,\pi)$ with test profile P is to achieve a variance reduction, i.e., an improvement of the precision of the estimate. This cannot be achieved by *any* test profile T; some particular profiles would even increase the variance. So T has to be carefully selected, exploiting prior information on the program π. Our aim is to find a (Markov–type) test profile T leading to *minimum* variance.

Let us make this more explicit. Ideally, the variance $var_T(S)$ of S with respect to distribution T would be an appropriate measure for the inprecision of the estimate S. However, S depends on π, and, as it has been argued in section 2, π is to be considered as a random variable (distributed according to Q), the actual instance of which is not perfectly known. So, instead of $var_T(S)$, its *expected value* $E_Q(var_T(S))$ with respect to Q is to be minimized: We look for that (Markov–type) test profile T that produces an estimate with least expected variance from the viewpoint of prior (uncertain) information on π. Note that E_Q acts on π, while E_P acts on X.

The exploitation of the prior information will be treated in the next section. The following general observation shall be used as the basic tool for the optimization of the test transition probabilities:

Lemma 3.1. *The expected variance $E_Q(var_T(S))$ is minimized with respect to T, if and only if*

$$G(T) := E_P(L(X)\,H(X)) \tag{9}$$

is minimized, where

$$L(x) := P(x)\,/\,T(x)$$

is the likelihood ratio, and H is given by

$$H(x) := E_Q([c(x,\pi)]^2). \tag{10}$$

Proof. Because of

$$var_T(S) = E_T(S^2) - [E_T(S)]^2 = E_T(S^2) - [E_P(c(X,\pi))]^2,$$

where the second term on the r.h.s. does not depend on T, minimization of $E_Q(var_T(S))$ reduces to minimization of

$$E_Q(E_T(S^2)) = E_Q\left(E_T\left(\frac{[P(X)]^2}{[T(X)]^2}\,[c(X,\pi)]^2\right)\right)$$

$$= E_Q\left(E_P\left(\frac{P(X)}{T(X)}\,[c(X,\pi)]^2\right)\right) = E_P\left(\frac{P(X)}{T(X)}\cdot E_Q\left([c(X,\pi)]^2\right)\right).$$

□

Note that the problem of finding a test profile T that minimizes (9) is a *stochastic optimization problem*, since the objective function (9) is represented as an expected value of a random variable.

In the following section, the function $H(x)$ will be represented explicitly for the diverse special cases of dependability measures. Based on these representations, it is easy to compute an approximation for the function $G(T)$: Sample N paths $X^{(1)}, \ldots, X^{(N)}$ according to profile P, compute for each $\nu = 1, \ldots, N$ the values $L(X^{(\nu)})$ (using (1) and (8)) and $H(X^{(\nu)})$ (using the formulas in section 4), and take the average of the products $L(X^{(\nu)})H(X^{(\nu)})$. This will be used in section 5 for numerical optimization.

Proceeding as described above, the paths $X^{(1)}, \ldots, X^{(N)}$ are always sampled according to the operational profile P. It is possible to use another sampling distribution as well, as the following generalization of Lemma 3.1 shows:

Lemma 3.2. *Let P_0 be defined by transition probabilities $p_0(i,j)$ with the property $p_0(i,j) > 0$ exactly if $p(i,j) > 0$. Then $G(T)$ can be represented as*

$$G(T) = E_{P_0} \left(\frac{[P(X)]^2}{P_0(X)\, T(X)} \, H(X) \right). \tag{11}$$

It is clear that in the special case $P_0(x) := P(x)$, (11) reduces to (9). The possible advantage of using the more complicated representation (11) instead of (9) will be discussed in section 5.

4 Optimal Probability Shifts for Specific Measures

In this section, the problem of computing optimal shifts of the transition probabilities of the Markov chain is reduced to the solution of specific stochastic optimization problems by deriving explicit representations of the function $H(x)$ defined by (10). We shall do this separately for the three considered measures of interest (risk, safety and reliability). The distribution Q will be taken into account by two alternative models, the parameters of which have to be estimated, e.g., by educated guesses or based on experiences with previous versions of the software under consideration. This approach allows an adaption of the test cases to be developed to any available pre–information (or even mere conjectures) concerning especially error–prone or critical parts of the software.

Let now x be fixed and π be randomly distributed according to Q. We are looking for appropriate models for describing the random variable loss (x, π). The following two models are very natural (the user has to decide which of them is more adequate for his or her special case of application):

Model I (static model):

(a) The operation corresponding to arc (i,j) has a prior probability $f(i,j) > 0$ of being incorrect.

(b) The events that different operations are incorrect are independent from each other.

(c) If, during the execution of the program, an arc (i,j) corresponding to an incorrect operation is traversed at least once, a loss of $l(i,j)$ occurs.

(d) Losses at different arcs are added.

Model II (dynamic model):

(a) Each execution of the operation corresponding to arc (i, j) has a prior probability $f(i, j) > 0$ of failing.

(b) Failures at different executions of an operation or of different operations occur independently from each other.

(c) Each time when an execution of the operation corresponding to arc (i, j) fails, a loss of $l(i, j)$ occurs.

(d) Losses at different executions of operations are added.

In both models, we always assume $f(n, n) = 0$: Once the final state is reached, no failure can occur anymore.

The convention that failures at an operation are counted only once in Model I, while they are counted as often as they occur in Model II, is in accordance with the different choice of the "processing unit" in the two models: in Model I, this unit is an *operation*, whereas in Model II, it is an *execution of an operation*.

In a certain sense, these two models represent opposite extreme cases: the essential difference between them is that in Model I, different executions of an operation are assumed to be identical in their failure behavior (full correlation), while in Model II they are assumed to be independent with respect to failure (no correlation at all). [3] Perhaps, a combined model would be the most realistic one; it is easy to extend our approach in this direction.

We admit that these two model types have been selected rather for the sake of mathematical tractability than motivated by practical considerations. A model that precisely reflects the situation in a real–world application would be too complex to be amenable to a computational analysis. Nevertheless, we think that most applications get close to one of the suggested model types, or to a suitable mixture of both, which is obtained by weighting two descriptions (one according to Model I, the other according to Model II) by certain factors.

Remark.

In order to avoid misunderstandings, we need to be precise what we mean by a "failure at (an execution of) an operation": we do *not* mean a program failure that gets *evident* during the execution of this operation, but instead a program failure caused by a fault that can be assigned specifically to this operation (resp. to the considered execution of this operation). If the first-mentioned interpretation would be chosen, assumption (b) above would not be satisfied, since once a failure has been occurred, the internal state of the program is likely to be corrupted, which increases the probability that subsequent executions of the same operation or of different operations get unsuccessful from the user's point of view.

Let us give an example for our interpretation: let a program consist of two subsequent operations $(1, 2)$ and $(2, 3)$. Operation $(1, 2)$ computes salaries of employees, while operation $(2, 3)$ shows these salaries on the monitor. Then an application of an incorrect formula for the salary computation will result, in our interpretation, only in a failure at operation $(1, 2)$, but

[3] The other difference between the two models, namely that failures at an operation (i, j) are counted only once in Model I, while they are counted as often as they occur in Model II, is not essential: this difference can be compensated, if necessary, by multiplying $l(i, j)$ by the average transition rate of operation (i, j).

not in a failure in operation $(2,3)$, although of course in the case of incorrect internal states of the variables containing the salaries, also operation $(2,3)$ is not successful from the viewpoint of the user. Following our interpretation, the estimate of the failure probability $f(2,3)$ should only cover failures that are caused by a faulty output of the internal values to the monitor, whatever these values are.

4.1 Risk

We start with the case of risk estimation.

Model I:

On the assumptions of Model I, one obtains the following representation:

$$\text{loss}\,(x,\pi) = \sum_{(i,j)\in A(x)} \text{fail}\,(i,j)\, l(i,j), \tag{12}$$

where $A(x)$ denotes the set of transitions (i,j) contained in path x, and

$$\text{fail}\,(i,j) = \begin{cases} 1, \text{if transition } (i,j) \text{ fails}, \\ 0, \text{else}, \end{cases}$$

such that $E_Q(\text{fail}(i,j)) = f(i,j)$. According to assumption (b) of Model I, the variables $\text{fail}(i,j)$ are considered as independent Bernoulli variables.

For the sake of a simplification of the formulas, we assume in this subsection that the failure probabilities $f(i,j)$ are small values, which is realistic for most applications: It does not make sense to expose π to a test for risk estimation, if π has not yet proved to be sufficiently reliable in a series of pre–tests with the aim of fault detection. Using this assumption, the following approximation can be derived:

Proposition 4.1. *Let $f(i,j)$ be of order $O(\epsilon)$ for $1 \le i,j \le n$ $(\epsilon \to 0)$. Then, in Model I and with $c(x,\pi) = loss\,(x,\pi)$, the function $H(x)$ can be approximated as follows:*

$$H(x) = \sum_{(i,j)\in A(x)} f(i,j)\,[l(i,j)]^2 \;+\; O(\epsilon^2) \quad (\epsilon \to 0). \tag{13}$$

Proof. Clear from equations (3) and (4) in [7]. □

Note that in general, the sum on the r.h.s. of (13) is of order $O(\epsilon)$.

Model II:

In Model II, (12) has to be replaced by

$$\text{loss}\,(x,\pi) = \sum_{k\ge 1} \text{fail}_k\,(x)\, l(x_{k-1},x_k), \tag{14}$$

where

$$\text{fail}_k\,(x) = \begin{cases} 1, \text{ if transition } (x_{k-1},x_k) \text{ fails}, \\ 0, \text{ else}, \end{cases}$$

such that $E_Q(\text{fail}_k(x)) = f(x_{k-1}, x_k)$.

According to assumption (b) of Model II, the variables $\text{fail}_k(x)$ $(k = 1, 2, \ldots)$ are considered as independent Bernoulli variables. Again, it is assumed that the probabilities $f(i, j)$ are small. Now the following approximation is obtained:

Proposition 4.2. *Let $f(i,j)$ be of order $O(\epsilon)$ for $1 \le i, j \le n$ $(\epsilon \to 0)$. Then, in Model II and with $c(x, \pi) = \text{loss}\,(x, \pi)$, the function $H(x)$ can be approximated as follows:*

$$H(x) = \sum_{k \ge 1} f(x_{k-1}, x_k)\,[l(x_{k-1}, x_k)]^2 \;+\; O(\epsilon^2) \quad (\epsilon \to 0). \tag{15}$$

Proof. An analogous calculation as in the proof of Proposition 4.1 yields

$$H(x) = \sum_k f(x_{k-1}, x_k)\,[l(x_{k-1}, x_k)]^2$$

$$+ \sum_k \sum_{m \ne k} f(x_{k-1}, x_k)\, f(x_{m-1}, x_m)\, l(x_{k-1}, x_k)\, l(x_{m-1}, x_m).$$

With $L := \max_{i,j} l(i,j)$ and $\text{length}\,(x) := \min\{k \mid x_k = n\}$, one obtains, using $f(n, n) = 0$, that the second sum on the r.h.s. is smaller or equal

$$L^2 \cdot \left(\sum_k f(x_{k-1}, x_k) \right)^2 \le L^2 \cdot (\text{length}\,(x))^2 \cdot O(\epsilon^2).$$

(Note that the chain X reaches the absorbing state n with probability one, i.e., $\text{length}(X) < \infty$ with certainty, such that $\text{length}\,(x)$ can be assumed to be finite.) □

4.2 Safety

In this subsection, the values $f(i,j)$ and $l(i,j)$ are used to model the safety expectation instead of the risk expectation, i.e., we are interested in the costs $c(x, \pi)$ defined by (3).

Model I:

The function $\text{loss}\,(x, \pi)$ is represented by (12) again. Now we obtain:

Proposition 4.3. *In Model I and for $c(x, \pi)$ defined by (3), the function $H(x)$ is given by*

$$H(x) = \sum_{u \in \mathcal{L}_{r(x)}(C)} \prod_{s=1}^{r(x)} [u_s f(a_s(x)) + (1 - u_s)\,(1 - f(a_s(x)))], \tag{16}$$

where $a_s(x)$ $(s = 1, \ldots, r(x))$ are the elements of $A(x)$, and

$$\mathcal{L}_r(C) = \left\{ u = (u_1, \ldots, u_r) \,\middle|\, u_s \in \{0, 1\}\; (s = 1, \ldots, r) \;\wedge\; \sum_{s=1}^{r} l(a_s(x)) u_s > C \right\}.$$

Proof. Denote by $I(B)$ the indicator function of event B, such that

$$c(x, \pi) = I(\{\text{loss}\,(x, \pi) > C\}).$$

Then, using (12), we find

$$H(x) = E_Q(I(\{\text{loss}\,(x,\pi) > C\})) = Q\{\text{loss}\,(x,\pi) > C\} = Q\left\{\sum_{s=1}^{r(x)} \text{fail}\,(a_s(x))\, l(a_s(x)) > C\right\}.$$

Let $r = r(x)$ and $a_s = a_s(x)$ $(s = 1, \ldots, r)$. Then for fixed $(u_1, \ldots, u_r) \in \{0,1\}^r$,

$$Q\left\{(\text{fail}\,(a_1), \ldots, \text{fail}\,(a_s)) = (u_1, \ldots, u_r)\right\} = \left[\prod_{s:\, u_s=0} (1 - f(a_s))\right] \cdot \left[\prod_{s:\, u_s=1} f(a_s)\right]$$

$$= \prod_{s=1}^{r} \left[u_s f(a_s) + (1-u_s)\,(1 - f(a_s))\right].$$

Thus

$$H(x) = \sum_{u\in\{0,1\}^r} Q\left\{\sum_{s=1}^{r} \text{fail}\,(a_s)\, l(a_s) > C \ \wedge\ (\text{fail}\,(a_1), \ldots, \text{fail}\,(a_r)) = (u_1, \ldots, u_r)\right\}$$

$$= \sum_{u\in\{0,1\}^r} Q\left\{\sum_{s=1}^{r} l(a_s) u_s > C \ \wedge\ (\text{fail}\,(a_1), \ldots, \text{fail}\,(a_r)) = (u_1, \ldots, u_r)\right\}$$

$$= \sum_{u\in\mathcal{L}_r(C)} Q\left\{(\text{fail}\,(a_1), \ldots, \text{fail}\,(a_r)) = (u_1, \ldots, u_r)\right\}$$

$$= \sum_{u\in\mathcal{L}_r(C)} \prod_{s=1}^{r} [u_s f(a_s) + (1-u_s)\,(1 - f(a_s))].$$

□

The set $\mathcal{L}_r(C)$ just contains the binary string representations of all subsets $\{i_1, \ldots, i_k\}$ of $\{1, \ldots, r\}$ with the property that their "total weight" $\sum_{\kappa=1}^{k} l(a_{i_\kappa})$ is larger than C. Let us mention that it is usually not necessary to generate all elements $u \in \mathcal{L}_r(C)$ for computing $H(x)$ to a sufficient degree of accuracy: In realistic applications, the failure probabilities $f(a_s)$ are close to zero, such that products of the form $f(a_{s_1}) \ldots f(a_{s_k})$ with more than 3 or 4 factors are negligible. Therefore, if the elements $u \in \mathcal{L}_r(C)$ are enumerated in such a way that we start with the vector $u = (0, \ldots, 0)$, continue with the vectors

$$u = (1, 0, \ldots, 0),\ (0, 1, 0, \ldots, 0),\ \ldots,\ (0, \ldots, 0, 1)$$

containing *one* 1–bit, then with the vectors containing *two* 1–bits, etc., the contributions to the sum in (16) get soon very small, such that the summation can be aborted.

Model II:

Now, we replace again the representation (12) for loss (x,π) by (14). The computation of $H(x)$ runs along the same line as in the proof of Proposition 4.3, with the following result:

Proposition 4.4. *In Model II and for $c(x,\pi)$ defined by (3), the function $H(x)$ is given by (16), where now $a_s(x) = (x_{s-1}, x_s)$, and $r(x) :=$ length (x) is the length of the Markov chain x before reaching the absorbing state.*

4.3 Reliability

As it has been mentioned in subsection 2.3, reliability can be considered formally as the special case $C = 0$ of safety. It is not convenient, however, to treat this case by the general method of subsection 4.2, since, as stated above, $\mathcal{L}_r(0)$ contains $2^r - 1$ elements, which makes the computation of $G(T)$ along the lines of the previous subsection inefficient. Nevertheless, there are simple representations of $H(x)$ in this case. We cite them without proof, since they have already been presented implicitly in [8]. Moreover, they can be derived in a rather straightforward way from the more general results of Proposition 4.3 and Proposition 4.4.

Proposition 4.5. *In Model I and for $c(x, \pi)$ defined by (4), the function $H(x)$ is given by*

$$H(x) = 1 - \prod_{s=1}^{r} (1 - f(a_s(x))),$$

where $a_s(x)$ $(s = 1, \ldots, r(x))$ are the elements of $A(x)$.

Proposition 4.6. *In Model II and for $c(x, \pi)$ defined by (4), the function $H(x)$ is given by*

$$H(x) = 1 - \prod_{k \geq 1} (1 - f(x_{k-1}, x_k)).$$

5 Computation of Optimal Transition Probabilities

5.1 A Numerical Solution Algorithm

In this section, we turn to the question how to solve the stochastic optimization problem from Lemma 3.1,

$$\text{Minimize} \quad G(T) = E_P(L(X)\, H(X)) \quad \text{for } T \in \mathcal{T}_P \tag{17}$$

numerically, where $\mathcal{T}_P$ denotes the set of all transition matrices $(t(i, j))$ $(1 \leq i, j \leq n)$ with the properties

$$t(i, j) \geq 0, \quad \sum_{j=1}^{n} t(i, j) = 1 \ (i = 1, \ldots, n), \quad t(n, n) = 1,$$

$$t(i, j) = 0 \quad \text{if and only if} \quad p(i, j) = 0.$$

In order to avoid technical complications, we further restrict ourselves to the set $\mathcal{T}_P^0 := \mathcal{T}_P \cap \{T \mid G(T) < \infty\}$ instead of $\mathcal{T}_P$, which obviously does not change the optimizers of (17).

The crucial observation is now that the objective function $G(T)$ of (17) is always *convex.* This has the consequence that the global optimizer of (17) can be computed in a very efficient way.

Theorem 5.1. *For arbitrary $H(x)$, the function $G(T)$ in (17) is a convex function of the n^2 variables $t(i, j)$ $(1 \leq i, j \leq n)$.*

Proof. The proof is a generalization of that of Proposition 4.1 in [8] to the case of arbitrary $H(x)$, so we keep the presentation rather concise. First of all, observe that for nonnegative integers $m_1, \ldots, m_n$, the function

$$\phi(x_1, \ldots, x_n) = -m_1 \ln x_1 - \ldots - m_n \ln x_n$$

is convex in $x_1, \ldots, x_n$, and hence also

$$\exp(\phi(x_1, \ldots, x_n)) = x_1^{-m_1} \ldots x_n^{-m_n} \tag{18}$$

is convex in $x_1, \ldots, x_n$. Now, let $H_P(X) := P(X)\, H(X)$, such that

$$G(T) = E_P(H_P(X) \cdot [T(X)]^{-1}).$$

For fixed x, let $M(i,j,x)$ denote the number of transitions k with $(x_{k-1}, x_k) = (i,j)$. For X distributed according to P, $M(i,j,X)$ becomes a random variable which is finite with probability one, except in the case $(i,j) = (n,n)$, which we exclude here. By means of the variables $M(i,j,X)$, the value $[T(X)]^{-1}$ can be represented as

$$[T(X)]^{-1} = \prod [t(i,j)]^{-M(i,j,X)}, \tag{19}$$

the product being over all (i,j) with the property that $t(i,j) > 0$, except (n,n). (Note that if $t(i,j) = 0$, the transition (i,j) cannot occur in path X.) The r.h.s. of (19) is, as a function in the variables $t_{ij} := t(i,j)$, of type (18) and hence convex. Since $H_P(X)$ does not depend on T, also $H_P(X) \cdot [T(X)]^{-1}$ is convex in these variables. The result follows now by the fact that E_P is a monotone and linear operator, i.e., E_P preserves convexity. □

It is not difficult to see that $\mathcal{T}_P$ is a convex set, since it can be represented as a cartesian product of one–point–sets and of simplices, i.e., sets of the form

$$\{\, x = (x_1, \ldots, x_k) \mid x_i \geq 0,\ \sum x_i = 1 \,\}.$$

Because the set $\{\, T \mid G(T) < \infty \,\}$ is also convex (this follows from the convexity of $G(T)$), we conclude that $\mathcal{T}_P^0$ is a convex set. So (17) with constraint $T \in \mathcal{T}_P^0$ is a convex optimization problem, which implies the following favorable property:

Corollary. *Each local minimizer of (17) with constraint $T \in \mathcal{T}_P^0$ is a global minimizer.*

Therefore, *local* (stochastic) optimization procedures can be applied to our problem. In the sequel, we propose a stochastic optimization algorithm of Frank–Wolfe–type (see [13]) for the solution of (17). Instead of using the representation (9) of $G(T)$, we shall formulate the algorithm for the more general representation (11) with an arbitrary sampling distribution P_0. Roughly speaking, the idea of the algorithm is the following: First, we set T equal to P. A sample of N independent paths according to P_0 is drawn. Now, instead of the expected value in (11), an approximation to this value obtained from the sample is minimized. This is done iteratively: For $s = 1, 2, \ldots$ and for each line i of the matrix T, its current elements $t(i,j)$ are modified, augmenting that $t(i,j^*)$ that leads to the maximal improvement of the estimated $G(T)$, and reducing the other values $t(i,j)$. In this way, a sequence of new matrices T is obtained, which are successively improving approximations to the optimal matrix.

More formally, our algorithm can be described as follows:

```
procedure OptimizeTest {
  set T := P;
  draw a sample of paths according to P_0;
  for s := 1 to maxs
    for i := 1 to n {
      for each j with p(i,j) > 0
        compute from the sample an estimate D(i,j) of the
        partial derivative of G(T) to the variable t(i,j);
      determine that j = j* for which D(i,j) is minimal;
      for each j with p(i,j) > 0 {
        t(i,j) := (1 - const/s) t(i,j);
        if (j = j*) t(i,j) := t(i,j) + const/s;
      }
    }
}
```

Some special points need further explanation:

(a) Estimating partial derivatives from the sample

An estimate $D(i,j)$ of the partial derivative $\partial G(T) \,/\, \partial t(i,j)$ is obtained from the sample $X^{(1)}, \ldots, X^{(N)}$ as follows: By (11) and the Law of Large Numbers,

$$G(T) \approx \frac{1}{N} \sum_{\nu=1}^{N} \frac{[P(X^{(\nu)})]^2}{P_0(X^{(\nu)})\, T(X^{(\nu)})}\, H(X^{(\nu)}). \tag{20}$$

Short calculation, using (19), shows that the partial derivative of the r.h.s. of (20) to the variable $t(i,j)$ is

$$- \sum_{\nu=1}^{N} \frac{[P(X^{(\nu)})]^2\, H(X^{(\nu)})}{P_0(X^{(\nu)})} \cdot \frac{M(i,j,X^{(\nu)})}{t(i,j)} \prod_{l,m} [t(l,m)]^{-M(l,m,X^{(\nu)})}, \tag{21}$$

so the last expression can be used for approximating $D(i,j)$. Therein, the product $\prod_{l,m}$ is over all pairs (l,m) of indices with $p(l,m) > 0$, i.e., over all arcs of the Markov chain.

(b) Stepsize

As the stepsize for reducing/augmenting the values $t(i,j)$, we have chosen a value decreasing proportionally to $1/s$ in successive iteration steps $s = 1, 2, \ldots$. (The value *const* is a proportionality factor; as to the choice of *const*, see the remarks under (d) below.) This is in accordance with a general principle in stochastic optimization: On rather weak conditions, a stepsize decrement of order $1/s$ is suitable to guarantee convergence of an adaptive stochastic optimization procedure (cf. [12]).

(c) Sampling distribution

The standard choice for the sampling distribution is $P_0 = P$. However, there are cases where it is advisable to choose P_0 *not* identical to the operational profile P: Because the procedure `OptimizeTest` shifts the transition probabilities based on a sample of paths distributed according to P_0, arcs (i,j) that have very low probabilities $p(i,j)$ in the operational profile would, for

$P_0 = P$, possibly not be represented at all in the sample, even if they are highly critical, i.e., if $l(i,j)$ is large. Then the procedure would not increase the test transition probabilities $t(i,j)$ of these arcs. For example, if in the usage model of Fig. 1, the operational probability $p(2,4)$ of the function "deletion" is only 0.001, but the loss $l(2,4)$ in case of failure at this function is very high, then the optimal value of $t(2,4)$ is considerably larger than 0.001. However, this cannot be recognized by the procedure `OptimizeTest` unless arc $(2,4)$ occurs at least in one sample path, which is unlikely for $P_0 = P$ and medium–size N.

Therefore, the sampling distribution P_0 should give all arcs a fair chance of occuring in the sample. In section 6, an algorithm will be described that constructs P_0 in such a way that it guarantees each arc a certain minimum probability of being traversed.

(d) Bootstrap

Of course, the replacement of the real distribution P (or P_0) by the distribution of the paths in the sample slightly changes the optimization problem (17), such that the obtained solution is only approximate, even after a large number of iterations. This drawback may be overcome by *repeating*, in an outer loop, the process of drawing a sample and modifying the current T. In order to preserve convergence, the proportionality factor *const* of the stepsize has to be decreased in each iteration of this outer loop. This leads to a gradual correction of the bias introduced by the first special sample of paths.

A further modification of this technique is to set P_0 equal to the current T at each iteration start of the outer loop: Then, in a kind of bootstrap, already obtained "good" test profiles T are used as a basis for sampling, which may lead to further improvements in the next iterations.

(e) Runtime

Although the algorithm `OptimizeTest`, together with formula (21), looks rather complex, it produces solutions for small to medium–size Markov chains in feasible time (cf. the results on a similar algorithm reported in [7]). This is due to the convexity of the optimization problem, which makes the application of time–consuming global optimization techniques unnecessary.

6 Constructing a distribution with arc traversal probability guarantees

In this section, we show how a distribution P_0, defined by transition probabilities $p_0(i,j)$, can be found that guarantees each fixed arc a probability of at least $1 \;/\; (n + \#(\text{arcs}))$ of being traversed. Let us emphasize that an algorithm for computing such a distribution is not only applicable within the context of our approach, but also useful in a broader range of application: Thévenod–Fosse has introduced the concept of *structural statistical testing* (see, e.g., [19] and [20]), whereby the distribution of random test inputs is chosen in such a way that each program path gets a certain minimum probability of being executed. This approach combines the advantages of structural testing (meeting structural coverage criteria) and random testing (dependability estimates can be derived). Our algorithm allows a transfer of this idea to test applications described by Markov usage models. Note that after testing with $T := P_0$, unbiased dependability estimates may be derived via the estimator (6).

For defining the distribution P_0, we start with the construction of a spanning directed rooted tree in the underlying graph G of the Markov chain. To fix terminology, let us denote a directed rooted tree where all arcs are oriented away from the root simply by a *tree*. By a *tree in* G, we understand a subgraph of G that is a tree in the sense above, with node 1 as the root. The smallest tree in G contains only node 1; we denote it by $\{1\}$. A tree in G is *spanning* if it contains each node of G.

Since each node i of G is reachable from node 1, there is, for every i ($i = 1, \ldots, n$) a path $x[i]$ leading from 1 to i. It is easy to see that each path $x[i]$ can be assumed to contain every arc at most once. Therefore, $x[i]$ can be considered as a subgraph of G. For two subgraphs G_1 and G_2 of G, we denote their union by $G_1 \cup G_2$. By a *prefix* of a path $(x_1, x_2, \ldots)$, we understand a subpath $(x_1, x_2, \ldots, x_k)$ starting at the same node x_1 as the given path and ending at an arbitrary node x_k of it.

Now, we construct a spanning tree in G by the following algorithm:

procedure `ConstructTree`(G) {
  set $tree := \{1\}$;
  for $i := 2$ **to** n {
    construct a path $x[i]$ from node 1 to node i;
    while ($tree \cup x[i]$ is not a tree) {
      find the shortest prefix $(1, \ldots, w)$ of $x[i]$ such that $tree \cup (1, \ldots, w)$ is not a tree;
      /* $w \in tree$, otherwise there would be a shorter prefix of this type */
      replace in $x[i]$ the prefix $(1, \ldots, w)$ by the (uniquely defined) path from node 1 to node w in $tree$;
    }
    set $tree := tree \cup x[i]$;
  }
}

Let us explain the procedure `ConstructTree`(G) using the example of the graph of Fig. 4 (see section 7 below). Initialization is done by letting the current tree consist only of node 1. In the first three iteration steps ($i = 2, 3$, and 4) of the **for**–loop, this tree is successively extended to the chains $1 \to 2$, $1 \to 2 \to 3$, and $1 \to 2 \to 3 \to 4$, respectively. In the next three iteration steps ($i = 5, 6, 7$), the last chain is further expanded successively to the tree of Fig. 2.

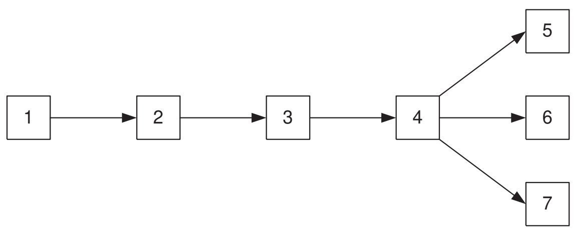

Fig. 2. Current tree after step $i = 7$.

Next, we could choose $x[8]$ as $(1, 2, 3, 8)$. However, in order to explain the **while**–loop, let us decide for another possible choice of path $x[8]$, namely $(1, 2, 3, 4, 5, 4, 3, 8)$. Joining $x[8]$ with the current tree (Fig. 2) would introduce cycles (i.e., we would not get a tree anymore). The shortest prefix $(1, \ldots, w)$ of $x[8]$ such that *tree* $\cup(1, \ldots, w)$ is not a tree is $(1, 2, 3, 4, 5, 4)$. So we replace in the current path $x[8]$ this prefix by $(1, 2, 3, 4)$, the path from node 1 to node 4 in *tree*. The new $x[8]$ is then $(1, 2, 3, 4, 3, 8)$. Still, joining $x[8]$ with *tree* yields a subgraph with a cycle. Now, the shortest prefix $(1, \ldots, w)$ of $x[8]$ such that *tree* $\cup(1, \ldots, w)$ is not a tree is given by $(1, 2, 3, 4, 3)$. We replace this prefix by $(1, 2, 3)$ and obtain, as $x[8]$, the path $(1, 2, 3, 8)$, which can be joined with *tree* without introducing a cycle.

In the next iteration steps of the **for**–loop, the current tree is further expanded in an analogous way, which finally yields, e.g., the tree in Fig. 3.

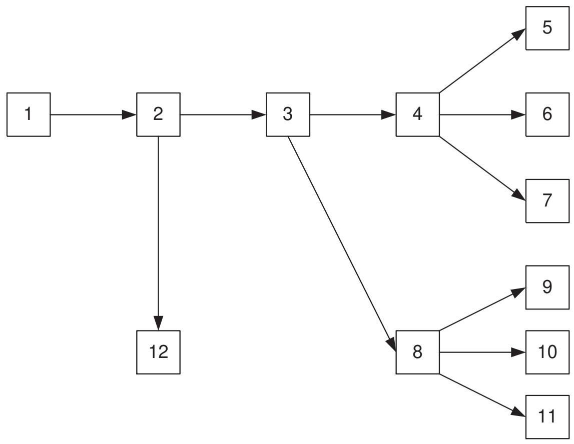

Fig. 3. Spanning tree of G.

It is not difficult to show that `ConstructTree` always "works":

Lemma 6.1. `ConstructTree` *produces a spanning tree in* G.

Proof. Follows immediately from the fact that after replacing the prefix $(1, \ldots, w)$ in the indicated way, the length of the shortest prefix of $x[i]$ that, joint with the current tree, does *not* yield a tree, is increased by at least one. □

We are now in the position to define P_0: Construct a spanning tree τ in G. For each node v of τ, let $leaves(v)$ be the number of leaves of τ in the subtree of τ rooted at v. Moreover, let $w_1, \ldots, w_k$ be the immediate successors of v in G. Now we set for each internal node v of G:

$$p_0(v, w_i) := \frac{leaves(w_i)}{leaves(v)} \quad (i = 1, \ldots, k),$$

$$p_0(v, w) := 0 \text{ for all } w \notin \{w_1, \ldots, w_k\}. \tag{22}$$

Note that $\sum_{i=1}^{k} p_0(v, w_i) = 1$, since $leaves(v) = \sum_{i=1}^{k} leaves(w_i)$.

Proposition 6.1. *For P_0 defined by the transition probabilities (22), each node v of G has a probability of at least $1/n$ of lying on a random path X distributed according to P_0.*

Proof. Since τ is a spanning tree, v lies on some path from the root of τ to some leaf of τ. Let m denote the number of such paths (i.e., the number of leaves). Obviously, $m \leq n$. We show by induction w.r.t. the size of τ that each of these paths has, under P_0, a probability $1/m$ of being traversed. For a tree τ consisting of a single node, the assertion is evidently true. If τ consists of more than one node, it has main branches $\tau_1, \ldots, \tau_k$, rooted at, say, $w_1, \ldots, w_k$, respectively.

According to (22), the probability of a transition $(root, w_i)$ is $leaves(w_i)/m$ $(i = 1, \ldots, k)$. Now observe that the rule for the random traversal of subtree τ_i is the same as that for the traversal of the whole tree τ. So, by induction assumption, each of the $m_i := leaves(w_i)$ paths from the root w_i of τ_i to one of the leaves of τ_i has the same probability $1/m_i$ of being traversed. In total, the probability that a given path starting at the root of τ, passing through w_i and ending at some fixed leaf is traversed, is

$$\frac{m_i}{m} \cdot \frac{1}{m_i} = \frac{1}{m},$$

which proves the assertion. □

Proposition 6.1 guarantees that each *node* is traversed with a certain minimum probability. It is now easy to modify the presented algorithm in such a way that each *arc* has a certain minimum probability of being traversed: For each arc (i, j), introduce an additional (artificial) node k, and replace (i, j) in G by the chain (i, k, j). Apply `ConstructTree` to the resulting extended graph G', which yields a tree τ'. Define P_0' by (22) based on τ'. Then, under P_0', each artificially introduced node has a probability of at least $1/(n + a)$ of being traversed, where a is the number of arcs in G. Omitting the artificial nodes again, this produces a distribution on G where each arc has a probability of at least $1/(n + a)$ of being traversed. By elementary probabilistic calculations we obtain from that the following assertion: For guaranteeing that a fixed arc is traversed with a probability of at least $1 - \epsilon$,

$$N = \frac{\ln \epsilon}{\ln(1 - 1/(n + a))}$$

sample paths are sufficient.

7 Example

As an illustration for the application of our technique, we re–investigate a (relatively small) Markov usage model described in [7] (see Fig. 4).

The model reflects part of a train schedule program, and the nodes 1 – 12 correspond to the following program states resp. screens: (1) start, (2) initial screen, (3) main menu, (4) information request, (5) departure information, (6) arrival information, (7) board service information, (8) booking places, (9) preferences, (10) place reservation, (11) acknowledgment, (12) stop.

The values $f(i, j)$ and $l(i, j)$ have been estimated in [7] by the numbers contained in the matrices F and L of Table 1.

It has been assumed that arc $(8, 10)$ corresponds to a newly implemented (i.e., probably error–prone) function with high criticality.

In [7], optimal test transition probabilities with respect to *risk* have already been computed, so we may simply cite the risks results here, and concentrate on the computation of the dependability measures *safety* and *reliability*. In all cases, Model I of section 4 has been chosen.

As to *safety*, we investigate four different choices of the critical–costs parameter C, namely $C = 5, 10, 50$, and 100. The case of *reliability* is treated as the special case $C = 0$ of *safety*.

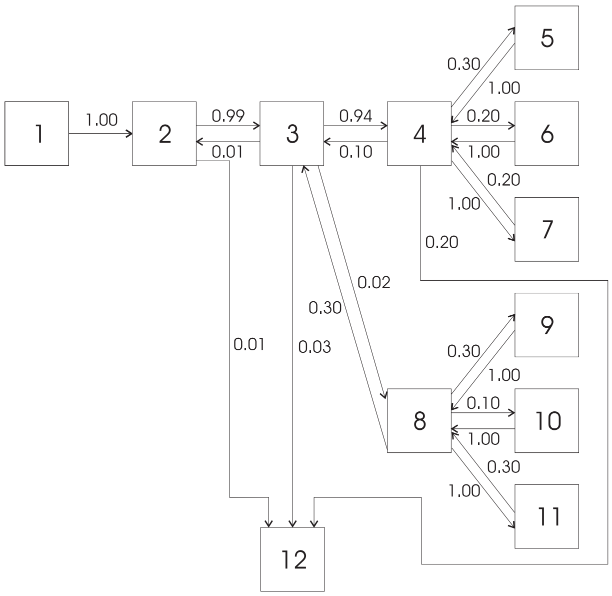

Fig. 4. Markov usage model from [7].

$F =$

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	l	0	0	0	0	0	0	0	0	0	0
2	0	0	m	0	0	0	0	0	0	0	0	l
3	0	l	0	m	0	0	0	l	0	0	0	l
4	0	0	l	0	m	m	l	0	0	0	0	l
5	0	0	0	l	0	0	0	0	0	0	0	0
6	0	0	0	l	0	0	0	0	0	0	0	0
7	0	0	0	l	0	0	0	0	0	0	0	0
8	0	0	l	0	0	0	0	0	l	h	l	0
9	0	0	0	0	0	0	0	l	0	0	0	0
10	0	0	0	0	0	0	0	l	0	0	0	0
11	0	0	0	0	0	0	0	l	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0

$L =$

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	m	0	0	0	0	0	0	0	0	0	0
2	0	0	m	0	0	0	0	0	0	0	0	l
3	0	m	0	m	0	0	0	m	0	0	0	l
4	0	0	m	0	m	m	m	0	0	0	0	l
5	0	0	0	m	0	0	0	0	0	0	0	0
6	0	0	0	m	0	0	0	0	0	0	0	0
7	0	0	0	m	0	0	0	0	0	0	0	0
8	0	0	m	0	0	0	0	0	l	h	m	0
9	0	0	0	0	0	0	0	m	0	0	0	0
10	0	0	0	0	0	0	0	m	0	0	0	0
11	0	0	0	0	0	0	0	m	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0

Table 1. Matrices F and L. In F, $l = 0.001$, $m = 0.01$, and $h = 0.1$. In L, $l = 1$, $m = 10$, and $h = 100$.

The resulting optimal test transition probabilities are presented in Table 2. We have applied the procedure OptimizeTest of section 5 with $N = 5000, maxs = 10$, and bootstrap as described in Remark (d) to the procedure OptimizeTest, with $K = 100$ iterations in the outer loop. In order to slightly accelerate the speed of convergence, we have worked with a stepsize of order $s^{-1.25}$ instead of s^{-1} (cf. Remark (b) to the procedure OptimizeTest).

With the parameters above, the runtime for one specific value of C was about 6 min 45 sec, using a C++ implementation on a PC Pentium–S, 200 MHz, 64 MB RAM. Within the chosen accuracy of two decimal digits, the results remained stable if N, $maxs$ and K were increased.

arc	$C = 0$ (*Reliab.*)	$C = 5$	$C = 10$	$C = 50$	$C = 100$	*Risk*
$(2, 3)$	99	99	99	99	99	99
$(2, 12)$	1	1	1	1	1	1
$(3, 2)$	1	1	1	1	1	1
$(3, 4)$	91	91	87	63	63	83
$(3, 8)$	5	5	10	33	33	11
$(3, 12)$	3	3	2	3	3	5
$(4, 3)$	12	12	12	15	15	11
$(4, 5)$	30	30	31	30	30	31
$(4, 6)$	20	20	21	20	20	20
$(4, 7)$	20	20	20	20	20	20
$(4, 12)$	18	18	16	15	15	18
$(8, 3)$	26	26	19	19	19	29
$(8, 9)$	30	30	30	30	30	15
$(8, 10)$	14	14	21	21	21	28
$(8, 11)$	30	30	30	30	30	28

Table 2. Optimal test transition probabilities (in percent) in the Example.

Comments:

1. It can be observed that the optimal test distribution for *reliability* ($C = 0$) lies closer to the operational distribution than that for *safety* (with high C) or that for *risk*. This can be explained by the fact that the failure severity, an additional factor re–shaping the optimal test distribution in the cases of safety and risk, has no influence in the reliability case.

2. Comparing $C = 5$ and $C = 10$, it can be noted that the result for $C = 10$ shifts the test probability from the branch starting with node 4 to the branch starting with node 8, and keeps the program states for a longer time within the last branch by reducing the probability of arc $(8, 3)$. In this way, the "high–loss–arc" $(8, 10)$ gets an increased probability of being tested.

3. Increasing C from $C = 50$ to $C = 100$ does not change the test profile anymore, at least not within the given degree of accuracy. This effect can be explained as follows: a loss larger than 50 is very unlikely to occur unless arc $(8, 10)$ is traversed, and if this is the case, the overall loss is likely to be even larger than 100.

4. Although, for $C = 100$, arc $(8, 10)$ is the only "critical" arc, the optimal test distribution only increases the transition probabilities from 0.10 (operational distribution) to 0.21. Both arc $(8, 9)$ and arc $(8, 11)$ keep their operational probabilities of 0.30, a value higher than 0.21 although the last two arcs are irrelevant for safety in this case! This may seem counterintuitive, but exact computations of the optimal test distributions for small Markov usage models, where an analytical treatment of the problem is still possible, show the same effect: the probability increase of critical arcs in loops by the shift to the optimal test distribution is rather cautious. As an explanation, one should keep in mind that stressing critical arcs in loops more intensely would produce a large number of test cases where such critical arcs are traversed *several* times, leading to negligible likelihood ratios, such that the information contained in the test results is not sufficiently exploited.

   These considerations may possibly indicate how difficult it is to choose good test transition probabilities *intuitively*, and emphasize the need for a sound formal method for their computation.

8 Conclusion

We have developed a general technique for computing optimal test transition probabilities in a Markov software usage model, based on pre–information on failure probabilities and losses in case of failure of single operations. The optimization criterion is maximum precision of dependability estimates obtained from the test results. Two different basic model (a static and a dynamic model) for the dependability aspects of the given program have been used. The optimization itself is done by means of a numerical procedure which is fast because of the convexity of the underlying (stochastic) optimization problem. The procedure uses a sampling distribution with a common lower bound on state transition probabilities; this distribution can also be applied in the more general context of structural statistical testing as introduced by Thévenod–Fosse.

Let us discuss some limitations of the approach presented here, as well as topics of further research. First of all, we have assumed that the failure behavior of the given program π depends on the input only via the program use x (i.e., the special path of the Markov chain). This "homogenity" assumption may be seen as a first approximation to reality; it can be relaxed to a certain extent, as it has been shown in [8] for the case of reliability estimation. For a practical treatment of inhomogeneous subdomains, the concept of *hazard sets*, as suggested by Sherer in [18], would possibly be particularly useful. Here, we do not deal with this question; let us only confine ourselves to two remarks: (i) The homogenity assumption can always be made satisfied by a suitable refinement of the Markov usage model. (ii) Also in cases where the homogenity assumption is not satisfied, our solutions may yield good approximations to the optimum, since it can be observed that inputs belonging to the same program use are much more correlated with respect to dependability and failure consequences than inputs belonging to different program uses.

Secondly, we have confined ourselves to two specific sets of assumptions for modeling the overall system dependability (a static and a dynamic model). It could be argued that other models would possibly lead to other results. Of course, more sophisticated models may be built in into our general framework; both the results of section 2 – 3 and those of section 5 remain valid in this case. However, since our two basic models represent extreme points of a scale, already a simple weighted average of the two resulting valuation functions $H(x)$ will possibly come very close to reality in many cases of application. Moreover, let us emphasize that for modeling the distribution Q reflecting the uncertainty on the behavior of the program under test, even coarse approximations to the true situation are sufficient to obtain significant gains, compared to testing with the operational profile: The dependability estimator S remains unbiased, whether Q is modeled correctly or not (cf. (7)), and its variance already decreases when only roughly modeled pre–information on failure probabilities and losses is allowed to enter into calculation.

Third, our procedure is not able to handle very large Markov usage models, for example in cases where the state space of the underlying Markov chain is a cartesian product of several large component sets (cf. the examples in [1] and [24]). An extension of our approach to this situation is possible, but it would exceed the scope of this paper. Also, models based on continuous state spaces cannot be considered here. Another interesting topic for future research seems to be an extension to continuous–*time* Markov chains, as they were investigated, e.g., in Shahabuddin's article [16] on the evaluation of highly dependable systems, or in Heidelberger's survey [10] on rare event simulation techniques in performance and dependability prediction.

The method described in this paper is well–founded from a statistical point of view, but, admittedly, conceptually rather complex. One may feel that simpler concepts of a heuristic type could suffice for practical purposes. (As an example of such a concept, we refer to [18].) Nevertheless, any heuristic for deriving test transitions in Markov usage models needs evaluation with respect to a well–defined performance criterion, such as our criterion of precision of the obtained dependability estimate. As a topic of future investigations, the theory presented in this article could be used for experimental evaluations of this kind.

A very challenging further topic of research would be an extension of the presented approach from dependability to *performance* measures (e.g., runtime or storage). Often, the runtime behavior of software is judged by means of experiments with random input, i.e., by simulation of the use in the real application. For such experiments, the choice of an appropriate input distribution is always crucial. Pre–information on especially time–consuming parts of the program can perhaps be used in an analogous way as we have done it here for dependability criteria. This could possibly lead to a composite Markov–model based performance/dependability evaluation technique, resembling in its flavor the composite performance and dependability analysis of fault–tolerant systems by Trivedi et al. [21]. Especially for software systems where workload considerations play an important role (queuing systems etc.), which results in close links between performance and dependability aspects, such a unified treatment would be highly desirable.

Acknowledgment. The author wants to thank the anonymous referees for their helpful comments to a previous version of the article.

References

[1] Avritzer, A., Weyuker, E. J., "The automatic generation of load test suites and the assessment of the resulting software", *IEEE Trans. Software Eng.*, vol. SE-21, 705 – 716 (1995).

[2] Bertolino, A., Strigini, L., "Acceptance criteria for critical software based on testability estimates and test results", *SAFECOMP 96, Proc. of the 15th International Conference on Computer Safety, Reliability and Security*, E. Schoitsch (ed.), 83 – 94, Springer (1997).

[3] Eckhardt, D. E., Lee, L. D., "A theoretical basis for the analysis of multiversion software subject to coincident errors", *IEEE Trans. Software Eng.*, vol. SE-11, 1511 – 1517 (1985).

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

[5] Glynn, P. W., Iglehart, D. L., "Importance sampling for stochastic simulations", *Management Science*, vol. 35 (11), 1367 – 1392 (1989).

[6] Gutjahr, W. J., "Optimal test distributions for software failure cost estimation", *IEEE Trans. Software Eng.*, vol. SE-21, 219 – 228 (1995).

[7] Gutjahr, W. J., "Failure risk estimation via Markov software usage models", *SAFECOMP 96, Proc. of the 15th International Conference on Computer Safety, Reliability and Security*, E. Schoitsch (ed.), 183 – 192, Springer (1997).

[8] Gutjahr, W. J., "Importance sampling of test cases in Markovian software usage models", *Probability in the Engineering and Informational Sciences*, vol. 11, 19 – 26 (1997).

[9] Hamlet, R., Taylor, R., "Partition testing does not inspire confidence", *Proc. 2nd Workshop on Software Testing, Verification, and Analysis*, July 1988, 206 – 215 (1988).

[10] Heidelberger, P., "Fast simulation of rare events in queuing and reliability models", *ACM Trans. on Modeling and Computer Simulation*, vol. 5 (1), 43 – 85 (1995).

[11] Laprie, J.-C., "Dependability — its attributes, impairments and means", in: *Predictably Dependable Computing Systems*, B. Randell, J.-C. Laprie, H. Kopetz and B. Littlewood (ed.), 3 – 24, Springer (1995).

[12] Ljung, P., Pflug, G., Walk, H., *Stochastic Approximation and Optimization of Stochastic Systems*, Birkhaus, Basel (1992).

[13] Murty, K. G., *Linear Complementary, Linear and Nonlinear Programming*, Heldermann (1988).

[14] Musa, J. D., "Operational profiles in software reliability engineering", *IEEE Software*, vol. 10, 14 – 32 (1993).

[15] Rubinstein, R. Y., *Simulation and the Monte Carlo Method*, Wiley (1981).

[16] Shahabuddin, P., "Fast transient simulation of Markovian models of highly dependable systems", *Performance Evaluation*, vol. 20, 266 – 286 (1994).

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

[18] Sherer, S. A., "Statistical software testing using economic exposure assessments", *Software Engineering Journal*, vol. 11, 293 – 298 (1996).

[19] Thévenod–Fosse, P., "From random testing of hardware to statistical testing of software", *Proc. IEEE COMPEURO '91*, 200 – 207 (1991).

[20] Thévenod–Fosse, P., Waeselynck, H., Crouzet, Y., "Software statistical testing", in: *Predictably Dependable Computing Systems*, B. Randell, J.-C. Laprie, H. Kopetz and B. Littlewood (ed.), 253 – 272, Springer (1995).

[21] Trivedi, K. S., Muppala, J. K., Woolet, St. P., Haverkort, B. R., "Composite performance and dependability analysis", *Performance Evaluation* 14, 197 – 215 (1992).

[22] Walton, G. H., Poore, J. H., Trammell, J., "Statistical testing of software based on a usage model", *Software – Practice and Experience*, vol. 25 (1), 97 – 108 (1995).

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

[24] Weyuker, E. J., "Using failure cost information for testing and reliability assessment", *ACM Trans. on Software Eng. and Methodology*, vol. 5, 87 – 98 (1996).

[25] Whittaker, J. A., "Markov chain techniques for software testing and reliability analysis", Ph. D. dissertation, University of Tennessee (1992).

[26] Whittaker, J. A., Poore, J. H., "Markov analysis of software specifications", *ACM Trans. on Software Eng. and Method.*, vol. 2 (1), 93 – 106 (1993).

[27] Woit, D. M., "Operational profile specification, test case generation, and reliability estimation for modules", Thesis, Queen's University, Kingston, Ontario, Canada (1994).