Optimising the reliability that can be claimed for a software-based system based on failure-free tests of its components

Peter Bishop, Andrey Povyakalo
City, University of London

Abstract

This short paper describes a numerical method for optimising the conservative confidence bound on the reliability of a system based on tests of its individual components. This is an alternative to the algorithmic approaches identified in [1]. For a given maximum number of component tests, the numerical method can derive an optimal test plan for any arbitrary system structure.

The optimisation method is based on linear programming which is more efficient that the alternative integer programming. In addition, the optimisation process need only be performed once for any given system structure as the solution can be re-used to compute an optimal integer test plan for a different maximum number of component tests.

This approach might have broader application to other optimisation problems that are normally implemented using integer programming methods.

Keywords: Statistical testing, Confidence bounds, Software reliability, Fault tolerance, Linear programming

1 Introduction

Statistical testing [2, 3, 4] provides a direct estimate of the software probability of failure on demand (*pfd*) of a demand-based system to some confidence bound, and it is recommended in functional safety standards such as IEC 61508 [5]. The standard approach to deriving a confidence bound on the *pfd* of a software-based system is to perform statistical testing on the whole system as a "black-box". In practice, performing tests on the entire system

may be infeasible for logistical reasons, such as lack of availability of all component subsystems at the same time during implementation.

To address this issue, a general method was developed for deriving a conservative confidence bound based on independent statistical tests applied to individual software-based components within the system [1]. The approach is completely general – it can be used to derive a conservative *pfd* bound for any system architecture (represented by a structure function) for a given component test plan.

The choice of component test plan affects the *pfd* bound that can be achieved. The paper showed that for symmetrical architectures (like n out of m vote structures), an even split of tests between components always produces the optimal *pfd* bound (regardless of whether the software components are diverse or identical).

Deriving an optimal test plan for arbitrary, asymmetric structures proved to be more of a challenge. Two sub-optimal test plan strategies were identified that are optimal for some asymmetric structures – but not in general.

This paper presents an alternative to the test plan algorithms described in to [1] that derives an optimal test plan using linear programming. We first summarize the main elements of the theory presented in [1], and then present our alternative method for generating an optimal test plan using numerical methods.

2 Confidence Bounds from Component Tests

Failure-free testing over m individual components can be characterized by a test plan vector

$$\mathbf{n} = (n_1, n_2, \ldots, n_m)' \tag{1}$$

where m is a number of components, n_j is the number of (failure-free) tests for component j, and the total number of tests is

$$N = \sum_{j=1}^{m} n_j. \tag{2}$$

To characterize the fault tolerance capability of a system architecture, we define $\mathbf{x} = (x_1, x_2, \ldots, x_m)'$ as a random binary vector of indicators of component failure. If component j fails, $x_j = 1$ and $x_j = 0$ otherwise.

The failure-proneness of the overall system is represented by a structure function $\varphi(\mathbf{x})$, where $\varphi(\mathbf{x}) = 1$ if the system fails for a given combination

of component failures and successes $\mathbf{x}$. Such a system state is known as a *cutset*.

Table 1 shows the states for a 2 out of 3 (2oo3) vote structure where two or more component failures will result in system failure (i.e. where $\varphi(\mathbf{x}) = 1$), e.g. in state $\mathbf{x}_4$, failure of components c_1 and c_2 causes system failure.

Table 1: Example 2oo3 vote structure function

Component	c_1	c_2	c_3	
State $\mathbf{x}$	x_1	x_2	x_3	$\varphi(\mathbf{x})$
$\mathbf{x}_0$	0	0	0	0
$\mathbf{x}_1$	1	0	0	0
$\mathbf{x}_2$	0	1	0	0
$\mathbf{x}_3$	0	0	1	0
$\mathbf{x}_4$	1	1	0	1
$\mathbf{x}_5$	1	0	1	1
$\mathbf{x}_6$	0	1	1	1
$\mathbf{x}_7$	1	1	1	1

A general proof given in [1] shows that, for any structure $\varphi(\cdot)$, the upper confidence bound, q_s, for the system *pfd* can be conservatively approximated as

$$q_s \leq \min\left(\frac{\ln(1/\alpha)}{N_{min}}, 1\right) \tag{3}$$

where N_{min} is the smallest total number of component tests in a cutset, i.e.

$$N_{min} = \min_{\forall \mathbf{x}:\varphi(\mathbf{x})=1} (\mathbf{n} \cdot \mathbf{x}) \tag{4}$$

where $\mathbf{n} \cdot \mathbf{x}$ is the scalar product of the two vectors, i.e. $\sum_{j=1}^{m} n_j x_j$. For example, for the case where $\mathbf{x} = \mathbf{x}_4$ in Table 1, the scalar product will be

$$1.n_1 + 1.n_2 + 0.n_3 = n_1 + n_2$$

For symmetrical structures, the optimal test plan is simple – the N tests are apportioned equally between the m components, e.g., in the 2oo3 vote structure, each component is assigned $N/3$ tests so $N_{min} = 2N/3$.

It proved to be more difficult to identify the optimal test plan for arbitrary asymmetric structures. It was shown in [1], that for any structure, the optimum test plan would always be able to achieve:

$$N_{min} \geq \frac{N}{P} \tag{5}$$

where P is the length (number of operational components) of the shortest success path. For example, in a 2oo3 vote structure, $P = 2$ because we need at least two working components for correct system operation.

Two test plan strategies were identified in [1] that are optimal for some asymmetric structures – but not in general. For example, one strategy assigned the N tests equally to the P components on a single shortest path. In the 2oo3 example, where $P = 2$, this would mean assigning $N/2$ tests to, say, c_2 and c_3, and zero to c_1. This allocation results in $N_{min} = N/2$ which is clearly worse than the optimal value of $N_{min} = 2N/3$.

While further test plan allocation algorithms were examined, it was always possible to identify a counter-example structure where the allocation would be sub-optimal.

The alternative approach is to derive an exact optimal test plan using integer programming, but this solution approach is *NP* hard [6]. We have developed a less computationally expensive approach by treating the number of component tests as non-negative real numbers rather than discrete integers.

In our alternative solution method, we maximize N_{min} in the continuous domain using linear programming, then convert the continuous test plan values back to discrete integers. The approach is described in more detail in the section below, and an example R script implementation of the method is given in Appendix A.

3 Test Plan Optimization using Linear Programming

Let us denote

m is the number of components;

$\mathbf{f} = (f_1, f_2, \ldots, f_m)' \in \mathbb{R}^m$ is the fraction of tests allocated to each component, i.e. $f_j = n_j/N,\ j = 1..m$;

s is the number of minimal cutsets

$\mathbf{1}_s = (1, 1, \ldots, 1)'$ is a unit vector of size s

Y is a $s \times m$ incidence matrix for minimal cutsets where $y_{ij} = 1$ if component c_j belongs to minimal cutset i, $y_{ij} = 0$ otherwise.

In order to maximize the minimum number of tests across all minimal cutsets, we are looking for the best among (sub-optimal) test plans that allocate the same fraction of tests g to all minimal cutsets in Y, by solving the following linear programming (LP) problem:

$$g \rightarrow \max \tag{6}$$

given

$$Y \cdot \mathbf{f} = g.\mathbf{1}_s; \tag{7}$$

$$\sum_{j=1}^{m} f_j = 1; \tag{8}$$

$$f_j \geq 0,\ j = 1..m, \tag{9}$$

where $Y \cdot \mathbf{f}$ is the matrix product of a matrix and a vector that computes sum of the component test fractions for every cutset, hence constraint (7) requires that $\sum_j (y_{ij}.f_j) = g, \quad i = 1..s$.

We can now eliminate variable g by defining the following terms:

$$\mathbf{h} = \mathbf{f}/g \tag{10}$$

$$H = 1/g. \tag{11}$$

Rewriting the LP problem in these terms, g is maximized when H is minimized, i.e.:

$$\sum_{j=1}^{m} h_j = H \rightarrow \min \tag{12}$$

given

$$Y \cdot \mathbf{h} = \mathbf{1}_s; \tag{13}$$

$$h_j \geq 0,\ j = 1..m. \tag{14}$$

The R *simplex*() LP solver function can be used to derive the solution to this problem. In practice however, this function can sometimes fail to find a solution when equality constraints are used – probably because it fails to generate an initial feasible point. To resolve the issue, we noted that H reaches its unconstrained minimum when $h_j = 0,\ j = 1..m$. Therefore,

equality constraint (13) can be replaced with the inequality constraint $Y \cdot \mathbf{h} \geq \mathbf{1}$, resulting in the following LP problem

$$\sum_j h_j = H \rightarrow \min \tag{15}$$

given

$$Y \cdot \mathbf{h} \geq \mathbf{1}_s; \tag{16}$$

$$h_j \geq 0,\ j = 1..m. \tag{17}$$

This optimization problem can solved with an R script that calls the LP solver $simplex()$ as shown in Appendix A.

The resultant optimal test allocation fractions for the components are:

$$\mathbf{f}_{op} = \mathbf{h}_{op}/H_{op} \tag{18}$$

and the optimal minimal cutset fraction g_{op} is:

$$g_{op} = 1/H_{op}. \tag{19}$$

As in general these fractions are continuous real values, the optimal apportionment of component tests i.e. $\mathbf{n} = \mathbf{f}_{op}N$ can be non-integer. An optimal integer component test allocation can be derived by first finding the smallest test multiple, N_0, where all component test fractions scale to integer values, i.e.

$$\lfloor \mathbf{f}_{op}N_0 \rfloor = \mathbf{f}_{op}N_0. \tag{20}$$

N_0 can be found by incrementing an integer number k by 1 until all the products $k \cdot f_j,\ j = 1..m$ become integer.

The optimal plan for a total number of tests

$$N^- = N - (N\ mod\ N_0) \tag{21}$$

is always integer. The remaining $(N\ mod\ N_0)$ tests can be allocated arbitrarily to any of the components (or not allocated at all) because they cannot increase the value of N_{min}.

If there is an option to add small number of tests to the plan, one can consider a test plan for N^+ tests where

$$N^+ = N^- + N_0. \tag{22}$$

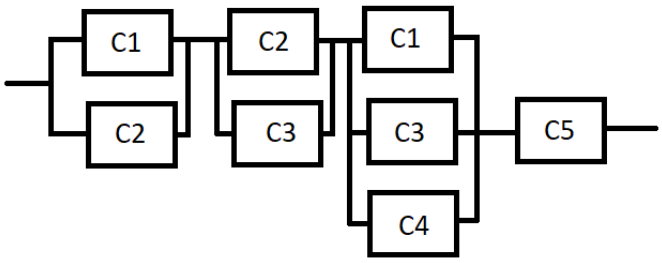

Figure 1: Example asymmetric RBD

4 Example

Let us consider an example asymmetric structure with the reliability block diagram (RBD) given in Figure 1.

Its minimal cutsets are:

$$\begin{aligned} &C1, C2 \\ &C2, C3 \\ &C1, C3, C4 \\ &C5 \end{aligned} \tag{23}$$

and its minimal cutset matrix Y is shown in Table 2.

Table 2: Minmal cutset incidence matrix

cutset	component j				
i	1	2	3	4	5
1	1	1	0	0	0
2	0	1	1	0	0
3	1	0	1	1	0
4	0	0	0	0	1

For this minimal cutset incidence matrix, the R script generates the following optimal test allocation fractions:

f_1	f_2	f_3	f_4	f_5	g_{op}
0.2	0.2	0.2	0.0	0.4	0.4

where zero tests are allocated to component c_4.

For this plan, sequential search gives $N_0 = 5$. Therefore, for a test campaign with a total number of tests, $N = 20003$, we have

$$N^- = 20003 - (20003 \ mod \ 5) = 20000 \tag{24}$$

with the test allocation

n_1	n_2	n_3	n_4	n_5	N^-
4000	4000	4000	0	8000	20000

and the least number of tests allocated to any minimal cutset is $N_{min} = g_{op} \cdot N^- = 8000$.

By comparison, if we use the strategy proposed in [1] of allocating N/P tests equally to components on a single shortest success path, such as (c_1, c_2, c_5), then $P = 3$. This is clearly sub-optimal as the least tested cutsets only have $\lfloor N/P \rfloor = N_{min} = 6667$ tests.

5 Concluding Remarks

It can be observed that the fractions generated in the continuous domain are independent of the number of tests, so they only need to be generated once for any given structure. It is only the integer test plan that needs to be recalculated for a given test budget – reducing the computing resources needed for a new plan.

In principle, it would be possible to create a library of optimal test plan solutions for different structures that can be converted to integer test plans for any specified number of component tests.

This strategy of solving in the continuous domain and then efficiently deriving optimal (or near optimal) solutions in the integer domain might be applicable to other problem areas.

A Test Plan Optimization R Script

The test plan optimization approach was implemented using the standard simplex solver available in the R statistical analysis library.

The use of the test plan optimizer is illustrated using non-symmetric structure shown in Figure 1.

```
library("boot")
```

```
#----------------------------------------
# lptplan_example <- function( N, alpha)
# N - total number of tests (default 20003)
# alpha = 1 - confidence level (default 0.05)
#----------------------------------------

lptplan_example <- function(
N=20003,
alpha = 0.05
)
{
# minimal cutset matrix
    cutsets <- matrix(
    c(
    1,1,0,0,0,  # cutset: C1, C2
    0,1,1,0,0,  # cutset: C2, C3
    1,0,1,1,0,  # cutset: C1, C3, C4
    0,0,0,0,1   # cutset: C5
    ), 4, 5,
    byrow=TRUE
    )

# Generate optimized test plan
    print  ( lptestplan(cutsets, N, alpha) )
}

#----------------------------------------
# lptestplan <- function(cutsets, N, alpha)
# cutsets
#  incidence matrix for the minimal cutsets
#  columns represent components
#  rows represent cutsets
# N   total number of tests
# alpha = 1 - confidence level
#----------------------------------------

lptestplan <- function(cutsets, N, alpha)
{
# Number of components
    m <- ncol(cutsets)
```

```
# Number of minimal cutsets
    s <- nrow(cutsets)

# Unit vectors
    uvm <- rep(1,m)
    uvs <- rep(1,s)

# Solve LP
    lp0 <- simplex(
        a = uvm,
        A3 = cutsets,
        b3 = uvs
    )
    H = as.numeric(lp0$value)
    h = lp0$soln

# Optimal cutset test fraction
    g <- 1/H

# Optimal component test fractions
    f <- h * g

# Find minimal integer test plan
    k <- 1
    r <- 1
    while(r>0){
        r <- sum ((f*k)%%1)
        if(r>0) k <- k+1
    }
    N0 <- k
    N_minus <- N - (N%%N0)

# Generate integer test plan
    N_min <- N_minus * g
    lptest_plan <- N_minus * f

# Calculate upper confidence bound
    q_u <- log(1/alpha)/N_min
```

```
# Return optimized result
  return
  (
    list(
    cutsets=cutsets,
    alpha = alpha,
    component_fractions = f,
    cutset_fraction = g,
    N = N,
    N0 = N0,
    N_minus = N_minus,
    lptest_plan = lptest_plan,
    N_min = N_min,
    q_u = q_u
    )
  )
}
```

References

[1] P. Bishop and A. Povyakalo, "A conservative confidence bound for the probability of failure on demand of a software-based system based on failure-free tests of its components," *Reliability Engineering & System Safety*, p. 107060, 2020.

[2] W. Ehrenberger, "Statistical testing of real time software," in *Verification and Validation of Real-Time Software*, pp. 147–178, Springer, 1985.

[3] D. L. Parnas, G. Asmis, and J. Madey, "Assessment of safety-critical software in nuclear power plants.," *Nuclear Safety*, vol. 32, no. 2, pp. 189–198, 1991.

[4] J. May, G. Hughes, and A. Lunn, "Reliability estimation from appropriate testing of plant protection software," *Software Engineering Journal*, vol. 10, no. 6, pp. 206–218, 1995.

[5] IEC, *Functional safety of electrical/electronical/programmable electronic safety-related systems, ed. 2, IEC 61508:2010*, 2010.

[6] A. Schrijver, *Theory of linear and integer programming.* John Wiley & Sons, 1998.