Optimising the Reliability that can be Claimed for a Software-based System based on Failure-free Tests of its Components

Peter Bishop[1,2] and Andrey Povyakalo[1]

[1] City, University of London {p.bishop;A.A.Povyakalo}@city.ac.uk
[2] Adelard, NCC Group pgb@adelard.com

Abstract. This paper describes a numerical method for optimising the conservative confidence bound on the reliability of a system based on statistical testing of its individual components. It provides an alternative to the sub-optimal test plan algorithms identified by the authors in an earlier research paper. For a given maximum number of component tests, this 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 than the alternative integer programming approach. 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.

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

1 Introduction

Statistical testing [4, 10, 8] 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 [6]. 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. For example, the statistical tests performed on the Sizewell B computer-based Primary Protection System (PPS) were performed on a single hardware division of the PPS, while the complete fault tolerant system consists of four divisions with 2-out-of-4 voting [5]. Similar constraints exist for the statistical testing of the Hinkley Point C protection system [9].

To address the constraint, testing can be restricted to a single component within the system architectures provided the fault tolerance mechanisms are pre-defined and static, i.e there is no dependency between components (such as dynamic fail-over schemes). A general method was developed for deriving a conservative confidence bound based on independent statistical tests applied (with zero failures) 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 claimed under worst case failure dependency conditions. The paper showed that for symmetrical architectures (like r-out-of-m vote structures), an even split of N tests between components always produces the optimal *pfd* bound, where:

1. if all components have identical software, subjecting each component to N/m tests produces the same *pfd* bound as subjecting the full system to N tests;
2. if the software in each component is not identical, subjecting each component to N/m tests produces the same *pfd* bound as subjecting the full system to $N(m - r + 1)/m$ tests.

The first result is unsurprising. If all components have the same software, identical defects will be present in every component – so it does not matter which component is tested, the system *pfd* is determined by the total of number of tests in all m components.

The second result is counter-intuitive, as a complete system with diverse components would have the same *pfd* bound as a non-diverse system when tested as a "black-box". The difference arises because the components are tested separately. A worst case example of non-identical failure dependency is shown in Figure 1 for a 2-out-of-3 vote structure. In this case, there is a common fault in

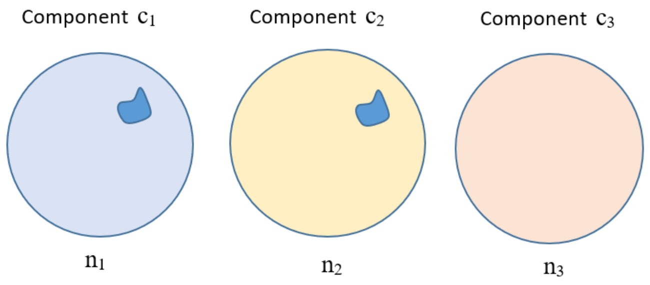

Fig. 1. Worst failure dependency example: non-identical software. *The dark patches represent defective regions in the input space of two components.*

just two of the components, so only the combined number of tests performed on components c_1 and c_2 determine the upper confidence bound on the system *pfd*. There are other possible common failure mode states like $\{c_1, c_3\}$ or $\{c_2, c_3\}$, however in all cases, it was shown in [1] that the system *pfd* is determined by the two least tested components, and more generally for a r-out-of-m structure, the system *pfd* is determined by smallest total of tests in $(m - r + 1)$ components.

As a result, the optimal test plan is an even split of the available tests across the components, i.e. if N tests are available, N/m tests are allocated to each component.

Deriving optimal test plans for arbitrary, asymmetric structures was more challenging. Two sub-optimal test plan strategies were identified in [1] that are optimal for some asymmetric structures – but not in general.

This paper presents an alternative to the test plan algorithms described in [1] that derives an optimal test plan using linear programming. We first summarise 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 characterised 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

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

Failures of the overall system can be characterised by *minimal cutsets* where failure of all components in any minimal cutset will cause a system failure.

A general proof given in [1] shows that, for any structure characterised by a set X of minimal cutsets, the $(1 - \alpha)$ 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 minimal cutset, i.e.:

$$N_{min} = \min_{\forall \mathbf{x} \in \mathbf{X}} \sum_{i \in \mathbf{x}} n_i \tag{4}$$

where $i \in \mathbf{x}$ identifies the components in minimal cutset $\mathbf{x}$.

For example, a 2-out-of-3 vote structure with diverse components has three minimal cutsets $\{c_1,\, c_2\}$, $\{c_2,\, c_3\}$ and $\{c_1,\, c_3\}$ so

$$N_{min} = \min(\ n_1 + n_2,\ \ n_2 + n_3,\ \ n_1 + n_3\ ).$$

For a 2-out-of-3 vote structure with identical components, the software failures in all components coincide, so there is only one minimal cutset: $\{c_1, c_2, c_3\}$ and

$$N_{min} = (n_1 + n_2 + n_3) = N.$$

The optimal test plans and confidence bounds derived in [1] for some common symmetrical structures are summarised in Table 1.

Table 1. Optimum test plan and confidence bounds for symmetrical structures

Structure	Software	n_j	N_{min}	Confidence Bound
Series (m-out-of-m)	Diverse	$\frac{N}{m}$	$\frac{N}{m}$	$\frac{m}{N}\ln\frac{1}{\alpha}$
Vote (r-out-of-m)	Diverse	$\frac{N}{m}$	$\frac{(m-r+1)N}{m}$	$\frac{m}{(m-r+1)N}\ln\frac{1}{\alpha}$
Vote (r-out-of-m)	Identical	any split	N	$\frac{1}{N}\ln\frac{1}{\alpha}$
Vote (1-out-of-m)	Either	any split	N	$\frac{1}{N}\ln\frac{1}{\alpha}$

The "series" structure is a chain of m components where the failure of any component causes system failure, so all m components must be functional for the system to be functional.

The "vote" structure (assuming voter correctness) combines the outputs of m components so that only r components need to be functional for the overall system to be functional.

It can be seen that for symmetrical structures, it is always optimal to apportion the N tests equally across the m components.

3 Optimising Test Plans for Asymmetric Structures

An asymmetric structure has a variable number of components in its minimal cutsets, such as the reliability block diagram (RBD) shown in Figure 2.

For such structures, explicit test plan optimisation is needed to ensure that the maximum value of N_{min} is obtained.

It was shown in [1] that it should always be possible to construct an allocation of component tests such that:

$$N_{min} \geq N/k_p. \tag{5}$$

where k_p is the length of the shortest possible success path.

For example, in Figure 2, the dashed lines denote the shortest success paths where $k_p = 3$.

Two sub-optimal allocation plans where identified in [1] that always satisfy this constraint:

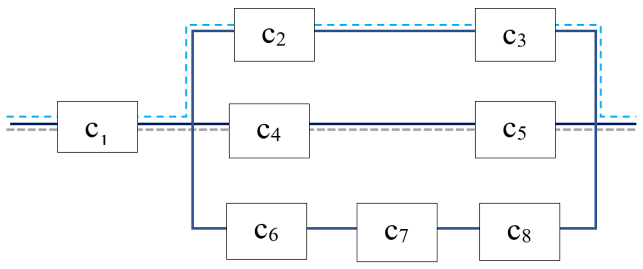

Fig. 2. Reliability block diagram. *The dashed lines are the shortest success paths*

- *Single shortest path*, where N/k_p tests are allocated equally to all components on just one shortest success path.
- *Balanced shortest path*, where the number of tests per component is proportional to the number of shortest success paths that include the component.

Both allocation methods are optimal for cases where each component appears only once in the RBD. The balanced path test plan also produces the optimal result for symmetric r-out-of-m vote structures (where the same component is present in more than one RBD branch).

Figure 3 shows the result of applying the balanced path allocation procedure to the RBD shown in Figure 2.

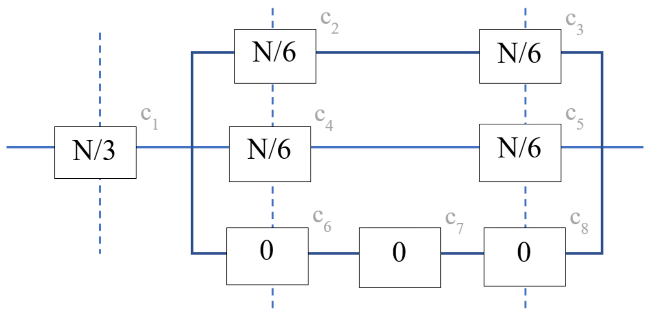

Fig. 3. Allocation of tests to components. The dashed lines are example minimal cutsets (there are further minimal cutsets).

Component c_1 has twice as many tests as the other shortest path components (because it is present in two shortest paths). It can be seen that the total number of tests in all minimal cutsets is the same ($N/3$).

Components c_6, c_7 and c_8 have no tests and could potentially fail on every demand, but this test plan is optimal because it maximises the number of tests in each cutset and hence the system reliability that can be claimed.

It proved to be more difficult to identify a general optimal test allocation strategy that was applicable to any arbitrary asymmetric structure. 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.

An exact optimal test plan could be produced using integer programming (optimisation of an objective function where the input variables are constrained to be discrete integer values [2]), but this solution approach is *NP* hard [12].

We chose a less computationally expensive approach that has been used in other application contexts (e.g. [3, 7]) where the integer optimisation problem is mapped to the continuous domain, optimised using linear programming, and the results converted back to discrete integer values.

In our solution approach, we represent the component tests as continuous-valued fractions of the total number of tests, maximise the N_{min} fraction using linear programming, then convert the optimal continuous test plan fractions back to a discrete integer test values for each component. The approach is described in more detail in the section below, and an example R [11] script implementation of the method is given in Appendix A.

4 Test Plan Optimisation using Linear Programming

Let us introduce the following notation

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 maximise N_{min} for a given N, we are looking for the optimal test plan among all test plans that allocate the same fraction of tests g to all minimal cutsets in Y, by solving the following linear programming (LP) problem:

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

given

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

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

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

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

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 maximised when H is minimised, i.e.:

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

given

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

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

A simplex LP solver algorithm can be used to derive the solution to this problem. Conceptually, the feasible region for the solution is a multi-dimensional polyhedron where each face represents a different constraint. The simplex algorithm finds the optimal solution by locating a vertex of the polyhedron (the initial feasible point) and moving to the next vertex along an edge that is closer to the optimal value (in our case, the minimum value of H). In practice however, the solver 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 an inequality constraint $Y \cdot \mathbf{h} \geq \mathbf{1}$. This makes no difference to the final solution as the optimisation seeks to minimise H, so the final solution will still satisfy the constraint $Y \cdot \mathbf{h} = \mathbf{1}$. Thus, the LP problem can be reformulated as follows.

$$\sum_{j} h_j = H \to \min \tag{15}$$

given

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

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

This optimisation problem can solved by 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 real values, the optimal apportionment of component tests i.e., $\mathbf{n} = \mathbf{f}_{op}N$ can be non-integer. An integer component test allocation can be derived by first finding the smallest number of tests, 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.

It follows that the plan for a total number of tests

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

is always integer because it is a multiple of N_0.

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

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

However, we know that the change in N_{min} between the plans for N^- and N^+ could be greater than one, i.e.

$$(N^+_{min} - N^-_{min}) \geq 1. \tag{23}$$

For example, if we start with the integer test plan for N^- and add one test to the k_p components on a single shortest success path, N_{min} increases by one. If $N_0 > k_p$ we know this solution is also an integer plan so the increase in N_{min} must be greater than 1.

For structures where $N_0 > k_p$, there will be intermediate integer test plans between N^- and N^+ where the non-zero component tests in each minimal cutset are not exactly equal, but the test plan still maximises the value of N_{min} for a given number of tests N.

In principle, there can be structures where $N_0 \gg k_p$, e.g. structures where the denominators of the component test fractions are differing prime numbers, so there could be an arbitrarily large number of intermediate test plans between a pair of perfectly balanced test plans with N^- and N^+ tests.

It would be possible derive optimal intermediate test plans where the non-zero component tests in the cutset are unequal, but this would require an entirely different, more complex algorithm (e.g. using integer programming). In practice, it is simpler to round up the fractional component tests to the next whole integer, i.e.

$$\mathbf{n}^{\uparrow} = \lceil \mathbf{f}_{op}N \rceil; \tag{24}$$

$$N^{\uparrow}_{min} = \min(Y \cdot \mathbf{n}^{\uparrow}); \tag{25}$$

$$N^{\uparrow} = \sum \mathbf{n}^{\uparrow}. \tag{26}$$

This plan for $N^{\uparrow}$ tests will include no more than m redundant tests to achieve the same N_{min} as a fully optimised integer test plan.

5 Example

Let us consider an example asymmetric structure with the reliability block diagram (RBD) given in Figure 4.

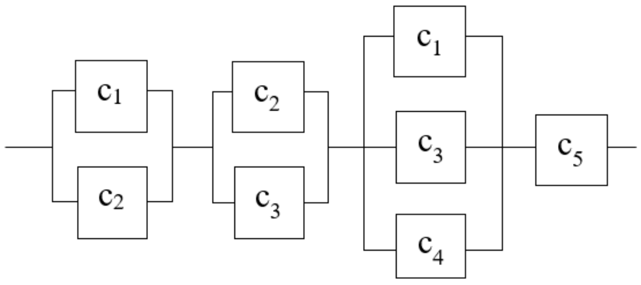

Fig. 4. Example asymmetric RBD

Its minimal cutsets are:

$$\{c_1, c_2\}$$
$$\{c_2, c_3\}$$
$$\{c_1, c_3, c_4\}$$
$$\{c_5\}$$

and its minimal cutset incidence matrix Y is shown in Table 2, where a "1" in a row indicates that the component is included in the minimal cutset.

Table 2. Minimal cutset incidence matrix

Cutset	Component j				
x	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 (see Appendix A) 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 $N = 20003$, the number of tests for an exact integer test plan is

$$N^- = 20003 - (20003 \ mod \ 5) = 20000 \tag{27}$$

with the following allocation of tests to components

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

where the least number of tests allocated to any minimal cutset is

$$N_{min} = g_{op} N^- = 8000.$$

The R script also generates a rounded-up test plan where the test allocation is:

${n^\uparrow}_1$	${n^\uparrow}_2$	${n^\uparrow}_3$	${n^\uparrow}_4$	${n^\uparrow}_5$	$N^\uparrow$
4001	4001	4001	0	8001	20004

and the least number of tests allocated to any minimal cutset is

$${N^\uparrow}_{min} = \min(Y \cdot \mathbf{n}^\uparrow)) = 8001.$$

This is not completely optimal because $N_{min} = 8001$ is possible with a test plan where $N = 20003$ by mapping the same test fraction values to different integer values, as shown the test plan below (the reduced tests are italicised).

n_1	n_2	n_3	n_4	n_5	N
4001	4001	*4000*	0	8001	*20003*

It can be seen that the simplified integerisation strategy of rounding up to the next integer only has a marginal impact on the number of tests required to achieve a given N_{min} value. It also allows test plans to be generated for every N_{min} value.

The choice of integerisation strategy only makes a marginal difference to the optimality of the plan. The main gain is achieved by identifying the optimal test fractions. For example, if we use the sub-optimal strategy proposed in [1] of allocating N/k_p tests equally to components on a single shortest success path, such as (c_1, c_2, c_5), then $k_p = 3$. This is clearly sub-optimal as the least tested cutsets only have $N_{min} = \lfloor N/k_p \rfloor = \lfloor 20003/3 \rfloor = 6667$ tests.

6 Discussion and Conclusions

In this paper, we have extended the test planning approach described in [1] so that optimal test plans can be produced for arbitrary structures by optimising test plans in the continuous domain. 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. An integer test plan can be recalculated from these fractions for any given test budget – reducing the computing resources needed for a new plan.

We identified two options for converting the test fractions to integer component test values. The simplest method is to round up fractional values to the next whole integer. While not strictly optimal in all cases, there is only a minimal (and bounded) increase in the number of tests required to demonstrate a given confidence bound.

Other integer conversion methods are possible but the differences are marginal – the optimisation is primarily achieved by identifying the optimal test fractions.

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 total number of component tests. This approach might have broader application to other optimisation problems.

References

1. Bishop, P., Povyakalo, A.: 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. Dantzig, G.B., Thapa, M.N.: Linear programming 1: introduction. Springer Science & Business Media (2006)
3. Dommel, H.W., Tinney, W.F.: Optimal power flow solutions. IEEE Transactions on power apparatus and systems (10), 1866–1876 (1968)
4. Ehrenberger, W.: Statistical testing of real time software. In: Verification and Validation of Real-Time Software, pp. 147–178. Springer (1985)
5. Hunns, D., Wainwright, N.: Software-based protection for Sizewell B: the regulator's perspective. In: Electrical and Control Aspects of the Sizewell B PWR, 1992., International Conference on. pp. 198–203. IET (1992)
6. IEC: Functional safety of electrical/electronical/programmable electronic safety-related systems, ed. 2, IEC 61508:2010 (2010)
7. King, T., Barrett, C., Tinelli, C.: Leveraging linear and mixed integer programming for SMT. In: 2014 Formal Methods in Computer-Aided Design (FMCAD). pp. 139–146. IEEE (2014)
8. May, J., Hughes, G., Lunn, A.: Reliability estimation from appropriate testing of plant protection software. Software Engineering Journal **10**(6), 206–218 (1995)
9. NNB: Hinkley point c pre-construction safety report 3 public version. Tech. rep., NNB Generation Company(HPC) Ltd (2017)
10. Parnas, D.L., Asmis, G., Madey, J.: Assessment of safety-critical software in nuclear power plants. Nuclear Safety **32**(2), 189–198 (1991)
11. Rizzo, M.L.: Statistical computing with R. CRC Press (2019)
12. Schrijver, A.: Theory of linear and integer programming. John Wiley & Sons (1998)

A Test Plan Optimisation R Script

The test plan optimisation approach was implemented using the standard simplex solver available in the R statistical analysis library. The use of the test plan optimiser is illustrated using the non-symmetric structure shown in Figure 4.

```
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 optimised test plan
    print  ( lptestplan(cutsets, N, alpha) )
}

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

lptestplan <- function(Y, N, alpha)
{
# Number of components
```

```
    m <- ncol(Y)

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

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

# Solve LP
    lp0 <- simplex(
        a = uvm,
        A3 = Y,
        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 exact integer test plan (<= N)
    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 exact integer test plan
    N_min <- N_minus * g
    n <- N_minus * f

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

# Generate rounded-up integer test plan (>=N)
    n_up <- ceiling(N * f)
    N_min_up <- min(Y %*% n_up)
    N_up <- sum(n_up)
```

```
# Calculate rounded-up upper confidence bound
    q_u_up <- log(1/alpha)/N_min_up

# Return optimised results
  return
  (
    list(
    cutsets=Y,
    alpha = alpha,
    component_fractions = f,
    cutset_fraction = g,
    N = N,
    N0 = N0,
    N_minus = N_minus,
    lptest_plan = n,
    N_min = N_min,
    q_u = q_u,
    N_up=N_up,
    lptest_plan_up=n_up,
    N_min_up=N_min_up,
    q_u_up = q_u_up
    )
  )
}
```