Reducing The Effects Of Infeasible Paths In Branch Testing

Yates, D. F. and Malevris, N.

Department of Computer Science
University of Liverpool, UK.

Abstract

Branch testing, which is one of the most widely used methods for program testing, see White [1] for example, involves executing a selected set of program paths in an attempt to exercise all program branches. Criteria for selecting such paths have, to date, received scant attention in the literature and it is the issue of developing a suitable path selection strategy to which this paper is addressed. Specifically, a selection strategy, which aims at reducing the number of infeasible paths generated during the branch testing exercise is proposed. The strategy is founded on an assertion concerning the likely feasibility of program paths. Statistical evidence in support of the assertion is provided, a method implementing the strategy is described, and the results obtained from applying the method to a set of program units are reported and analysed.

1.0 Introduction

Software testing remains both the principal means of locating errors in program code and the benchmark by which the pre-delivery level of confidence in a program's veracity is established. Although a number of testing techniques are available, not one of them can guarantee to isolate all sources of program error. Consequently, and as is implicitly recommended by Howden [2], it is current practice to use several techniques in order to achieve the desired confidence level. Of the techniques that are used, one of the most frequently adopted is branch testing: the testing of all possible single transfers of control within a program code unit.

To branch test a code unit, c, a model, usually taking the form of a directed graph $G_c = (V_c , A_c)$ in which the arc set A_c identifies with the branches of c, is first constructed. The branch testing method then entails:

1. selecting a set Π of paths through G_c which covers the arc set A_c ;
2. deriving from Π a corresponding set, Π_c , of paths through c and constructing a set of data, $D(\Pi_c)$ that will drive execution of c down each path of Π_c in turn;
3. executing c with $D(\Pi_c)$.

Ideally, these three steps would be performed once each in sequence. However, Π_c is often found to contain infeasible paths:- paths whose execution cannot be caused by any set of data. As a result, it is usually necessary to perform steps 1 and 2 iteratively in order to construct the data set $D(\Pi_c)$ which is ultimately used in step 3.

The overheads of the software testing exercise in respect of time and effort are substantial, and it is clearly advantageous to minimise the contribution made to them by any testing method that may be adopted. In the case of branch testing, the factors of both time and effort are radically influenced by the number of paths that are found to be infeasible at step 2 of the method, and this number depends upon the paths that are generated at step 1. Therefore, if it were possible to develop a path generation strategy which would reduce the incidence of infeasible paths at step 2, commensurate savings in the time and effort required by branch testing would be anticipated. This paper addresses the problem of devising such a strategy. More specifically, the paper: advances an assertion concerning the relative likely feasibility of program paths, and provides significant statistical evidence in support of its validity (section 2.0); describes a systematic path generation strategy based upon the assertion (section 3.0); details the corresponding path generation method (section 4.0); and reports the results obtained from applying it to a number of code units (section 5.0).

2.0 The Assertion And Its Statistical Justification

Yates and Hennell [3], observed that if π is a path through a code unit, and π involves $q > 0$ predicates, then:

> for π to be feasible, all q predicates must be consistent, whereas infeasibility of π requires as few as two predicates to be inconsistent.

The nature of this observation leads straightforwardly to the assertion that:

> if π_1 and π_2 are paths through a code unit, involving $r > 0$ and $s > r$ predicates respectively, then π_1 is more likely to be feasible than π_2.

If this assertion can be substantiated, the number of predicates involved in a path can be used as a measure of its likely feasibility, and therefore as a basis for attempting to reduce the

number of infeasible paths generated in the branch testing method.

As a step towards determining the validity of the assertion, it is first appropriate to establish whether or not there is any association between the feasibility of a path and the number of predicates it involves. To investigate this, data was collected from a sample of 642 paths which were generated by the authors in performing structural testing on a set of 36 subroutines chosen at random from the NAG Mark 11 FORTRAN Library. The number of paths in the sample involving q predicates which were determined to be feasible or infeasible for the observed values of q are presented in Table 1.

Table 1

No. of predicates	Total	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Feasible	193	17	21	23	22	34	26	17	13	8	6	5	0	1	0	0	0
Infeasible	449	4	10	13	24	20	46	54	41	74	58	42	23	14	5	17	4
No. of paths in sample	642	21	31	36	46	54	72	71	54	82	64	47	23	15	5	17	4

If, in fact, there is no association between the feasibility of a path and the number of predicates it involves, then it would be expected that approximately equal proportions of feasible (and similarly of infeasible) paths will occur for all values of $q \geq 1$. To formally test the hypothesis, H_0, of no association, that is of there being equal proportions of feasible paths for all $q \geq 1$, it is appropriate to perform a χ^2 test on the entries in Table 1.

It is shown in standard statistical texts, see Chatfield [4] for example, that for each ν, referred to as the number of degrees of freedom, there exists a value $\chi^2_\nu(\alpha)$ such that if $\chi^2 \geq \chi^2_\nu(\alpha)$, the hypothesis H_0 is rejected with significance probability α, and is accepted otherwise. In essence, χ^2 is a measure of the departure of the sample from what would be expected if H_0 were true, and it is the calculation of χ^2 and its subsequent comparison with $\chi^2_\nu(\alpha)$ that constitutes the χ^2 test.

In order to conform with all the conditions required by the χ^2 test, it was necessary to merge the last five columns of this table to give a 'reduced' table with $\rho = 2$ rows, $\tau = 12$ columns and a value of $(\rho - 1)(\tau - 1) = 11$ for ν (see Chatfield [4] for the validity of performing such a procedure). Using the entries in this reduced table, a value of $\chi^2 = 166.81$ is obtained. Comparing this value with standard χ^2 tables for $\nu = 11$, it is found that $\chi^2_{11}(0.005) = 26.775$, that is, the result $\chi^2 = 166.81$ is significant at the 0.05% level, and so H_0 is confidently rejected. Thus, it is concluded that the proportion of feasible paths amongst those possessing q predicates is **not** the same for each value of q, and therefore there **must be** some form of dependency between a path's feasibility and the number of predicates it involves.

The assertion states that, in the long run, the proportion p_s of feasible paths with s predicates is less than the corresponding proportion, p_r, of paths possessing $r < s$ predicates, for all valid values of r and s. Each long run proportion, p_q, which is in fact the probability that a path with q predicates is feasible, can be estimated from the sample by the corresponding observed proportion f_q of feasible paths with q predicates. The plot of the f_q against q which is given in figure 1, indicates a general decrease in f_q as q increases, and thereby provides support for the assertion (the line $f_q = 0.30$ indicates the mean value of f_q for the sample). Of the many parametric functions that might be used to model this situation, ones which suggest themselves on heuristic grounds are polynomials and those that decay exponentially.

Although a polynomial in q might provide a good model for f_q in the range $1 \leq q \leq 12$, when q is continuously incremented f_q either increases or decreases monotonically without bound. A polynomial model is thus rejected.

To investigate the appropriateness of an exponentially decaying function of the form:

$$p_{q+1} = Cp_q = e^{-\beta}p_q \qquad q = 1, 2, \cdots$$

where C is a constant and is to be determined along with p_1, a least squares fit of the function to the observed values of f_q was performed. The function obtained by the fitting process was:

$$p_{q+1} = e^{-0.1988}p_q \tag{2.1}$$

with $p_1 = 1.068e^{-0.1988}$, and this corresponds to the continuous function $p_q = 1.068e^{-0.1988q}$ which is also plotted in figure 1.

The fitting process revealed a value of $Z = 0.095$ for the sum of the squares of the residuals at q, $q = 1, 2, \ldots, 16$. This is significantly less than the corresponding value of 1.204 obtained from the model predicting a constant proportion of feasible paths for all q, thereby indicating that (2.1) provides a much more suitable model for explaining the observed values of f_q. To determine whether or not (2.1) is optimal in its class (the class of exponentially decaying functions with polynomial exponents), an exponentially decaying function with a slightly more complex exponent was investigated. Although the resulting least squares fit did realise a smaller value for the sum of the squares of the residuals, the application of an F-test (see,

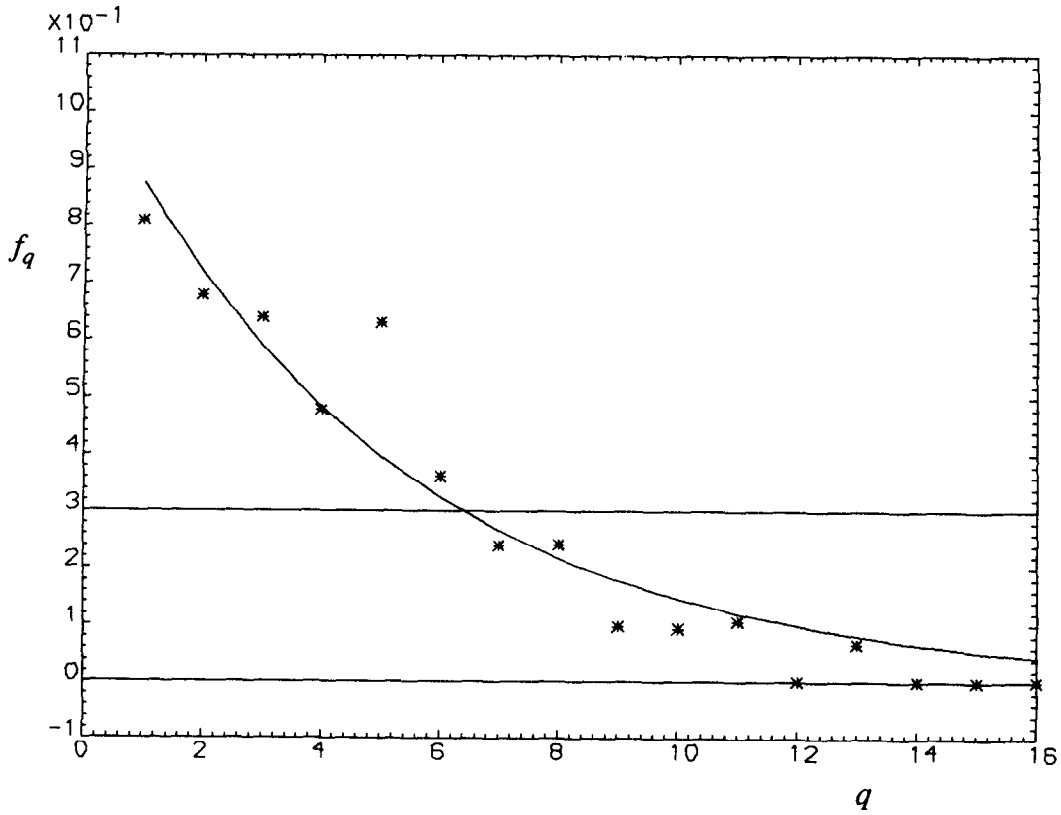

Figure 1

for example, Chatfield [4]) showed that the reduction achieved was not significant. Thus, increasing the complexity of the exponent over and above that in (2.1) does not significantly improve the fit obtained. In view of this, the (small) size of Z, and the fact that a value of p_0 (1.068) close to unity is predicted for a path with no predicates (this would be expected of any good model), it is concluded that (2.1) provides an appropriate model for explaining the nature of the relationship between the observed values of f_q. It is therefore claimed that the above provides strong supporting evidence that not only does path feasibility tend to decay with an increasing number of predicates, but also that this decay is exponential.

3.0 The Path Generation Strategy

In virtue of the strength of the statistical evidence presented, it may be inferred from the assertion that of all paths through a code unit, those involving the least number of predicates are most likely to be feasible. This inference then leads straightforwardly to the conclusion that from amongst the path sets that could validly be used to branch test a given code unit, a/the set Π^* whose constituent paths each involve a minimum number of predicates is most likely to contain the least number of infeasible paths. It is this conclusion that embodies the first half of the proposed path generation strategy.

Weyuker [5] has proved that the problem of determining the feasibility of a program path is undecidable. Consequently, no guarantee can be given that any path set, generated for branch testing purposes, will contain only feasible paths. If then the paths in Π^* are found to be infeasible and, as a result, full branch cover has not been achieved, which path(s) should next be generated in an attempt to increase the cover? How the assertion of section 2.0 is used to answer this question and thereby supply the outstanding component of the path generation strategy can be understood from the following.

Consider the situation in which Π^* has been employed in the branch testing of a code unit c and suppose that the single branch vw has not been covered. If $\pi(v, w)$ denotes the set of all paths through c on which vw lies, and $\pi_r(vw)$ the path in $\pi(v, w)$ which involves the r^{th} smallest number of predicates, $r = 1, 2, 3, \ldots$, then the fact that vw has not been covered using Π^* implies that $\pi_1(vw)$ has been found to be infeasible. If vw is to be covered, $N > 0$ additional paths from $\pi(v, w)$ must necessarily be generated, and of these $(N-1)$ will be infeasible (assuming that vw can ultimately be covered). Minimisation of N is clearly desirable, and it may readily be deduced from the assertion that this is most likely to be achieved if the additional paths are generated in the order of decreasing likelihood of their being feasible. That is, $\pi_2(vw)$ should first be generated then, if required, $\pi_3(vw)$, $\pi_4(vw)$, and so on.

Using Π^*, c, N, vw, and the $\pi_r(vw)$ as defined above, the path generation strategy sought, which will hereafter be referred to as MPS (Minimum Predicate Strategy), can now be specified:

(1) Generate a path set Π^* for c.

(2) Derive the value of Ter_2 (see Brown [6]) for c in respect of Π^*.

While $Ter_2 < 1$, repeat step 3.

(3) Select an uncovered branch vw of c, generate $\pi_r(vw)$, $r = 1, 2, ..., N$, and recalculate Ter_2.

Two questions arise in connection with step (3) of this strategy. The first of these is: if there are several uncovered branches, in what order should they be selected? No attempt is made here to give a definitive answer to this since its resolution is the subject of on-going research. What is certain, however, is that the question must be resolved as the selection mechanism can radically affect the methodology's efficiency. For, consider the case in which there are two uncovered branches, v_1w_1 and v_2w_2 say. If $\pi_s(v_1w_1)$ and $\pi_t(v_2w_2)$, $(s > 1, t > 1)$, are the shortest feasible paths through v_1w_1, and v_2w_2 respectively, then the situation wherein $v_2w_2 \in \pi_s(v_1w_1)$ but $v_1w_1 \in \pi_t(v_2w_2)$ can clearly arise. If v_2w_2 were to be selected at step (3) before v_1w_1, both branches would be covered after having generated t paths in total, and step (3) need not be applied in respect of v_1w_1. On the other hand, selecting v_1w_1 first would also necessitate the application of step (3) in respect of v_2w_2, and the same coverage would be achieved only after generating a total of $(s + t)$ paths.

The second of these two questions: what method(s) are employed to implement MPS?, will now be addressed.

4.0 The Path Generation Methods

In the paper by Yates and Hennell [3], a graph-theoretic method, referred to as SPM (the Shortest Path Method), was proposed for generating a path set with exactly the characteristics required of Π^*. To represent a code unit c, the method uses a DD-graph, $G_c = (V_c, A_c)$, possessing a single source vertex S, a single sink vertex F, and in which a length of unity is associated with each arc of A_c. SPM then identifies the required paths through c with a set of shortest paths from S to F in G_c and generates the set using standard graph-theoretic techniques. It is this path generation method that is adopted to implement step (1) of the path generation strategy.

If, as would be appropriate, the same model G_c is employed in implementing step (3) of the path generation strategy, the paths $\pi_r(vw)$, $r = 2, ..., N$, through c can be identified with respectively the $2^{nd}, \cdots, N^{th}$ shortest S - to - F path through arc $ij \in G_c$. That is, $\pi^{(k)}_{SijF}$, $k = 2, ..., N$, must be generated, where arc ij corresponds to branch vw of c, and $\pi_N(vw)$ is the first of the $\pi_r(vw)$ which is found to be feasible.

By employing the Principle of Optimality, path $\pi^{(k)}_{SijF}$ can be expressed as:

$$\pi^{(k)}_{SijF} = \pi^{(m)}_{Si} \cdot ij \cdot \pi^{(n)}_{jF} \qquad (4.1)$$

where m and n are integers satisfying $1 \le m \le k$, and $1 \le n \le (k-m+1)$ and the operator " $\cdot$ " denotes sequence. In order to construct the required paths in a systematic and efficient manner with the aid of (4.1) it is necessary that:

(1) none of the $\pi^{(k)}_{SijF}$ are duplicated (either of the occurrences of arc ij on a path $\pi = S, ..., ij, ..., ij, ..., F$, for example, could be viewed as being the arc ij of (4.1));

(2) the values of n and m corresponding to specific values of k can be determined efficiently;

(3) the appropriate paths $\pi^{(m)}_{Si}$ and $\pi^{(n)}_{jF}$ can be generated efficiently.

The first problem is straightforwardly solved by discarding any path $\pi^{(t)}_{Si}$ which contains arc ij. Various standard methods are available for solving the second problem and details of the one actually adopted can be found in Malevris [7]. The last of these entails solving instances of the well researched problem of determining the K^{th} shortest path between two vertices of a graph. Of the several solution methods that are available in the literature, see for example Dreyfus [8], Shier [9], and Minieka [10], the method due to Dreyfus, which in the present case has a time complexity of $O(K \mid V_c \mid)$, is the most apt and efficient. Despite the suitability of Dreyfus' method an alternative K^{th} shortest path algorithm, which would interface better with SPM by making more effective use of certain of its products, was devised and adopted. Since details of this algorithm are not at issue here, they are not reported, but can be found in Malevris [7]. This path generation method for implementing step (3) of the strategy will, since it extends the capabilities of SPM, be referred to as ESPM (the Extended Shortest Path Method).

5.0 Results and Discussion

In an attempt to assess the effectiveness of the proposed strategy, SPM and ESPM were coded in FORTRAN 77, interfaced, and the resulting implementation applied to the DD-graphs of a set of $N = 22$ subroutines taken from the NAG (Numerical Algorithms Group) Mark 11 FORTRAN subroutine library. The subroutines selected are specified in table 5.1 together with the number of branches each possesses and the number of arcs in the corresponding DD-graph. Also given in table 5.1 for each subroutine are: the value of Ter_2 achieved; the number of branches not covered; and the number of DD-graph arcs not covered, as a result of applying SPM.

Although the application of SPM to the 22 subroutines, realised a mean value of 0.73 (to 2 dec. places) for Ter_2, it is readily determined from the table that full branch coverage was achieved in only 8 cases, leaving a total of 62 uncovered arcs. In respect of subroutine F01CLF, however, it is worthwhile noting that, as a result of applying SPM, a "bug" was dicovered with the result that the value of $Ter_2 = 0.83$ attained represents the maximum achievable. Maximum branch cover, therefore, was achieved for 9 subroutines leaving 13 (now taken to be numbered $m = 1, 2, ..., 13$) with values of Ter_2 varying between 0.0 and 0.91.

ESPM was then applied to each of the 61 uncovered DD-graph arcs. For the purposes of the tests performed, the

Table 5.1					
Subroutine	No of branches	No of arcs in DD-graph	No of branches not covered	No of arcs not covered	Ter_2 for SPM
A02ABF	9	4	0	0	1.0
C02ADZ	12	7	2	1	0.83
C02AEZ	4	3	0	0	1.0
C06AAZ	9	4	2	1	0.78
C06ABZ	33	17	25	11	0.24
C06DBF	16	8	3	1	0.81
C06EBT	21	13	13	8	0.38
C06GCF	9	4	0	0	1.0
D02XHF	18	10	0	0	1.0
E01AAF	11	7	3	2	0.73
E02BBF	22	10	2	1	0.91
F01AFF	3	2	0	0	1.0
F01AHF	24	14	12	7	0.5
F01AZF	21	12	10	5	0.52
F01BEF	12	8	4	3	0.66
F01CLF	12	6	2	1	0.83
F01CMF	6	4	0	0	1.0
F01CSF	16	9	6	3	0.63
F03AMF	15	9	0	0	1.0
F04AQF	21	13	17	10	0.19
G04ADF	15	8	15	8	0.0
S17ACF	17	8	0	0	1.0

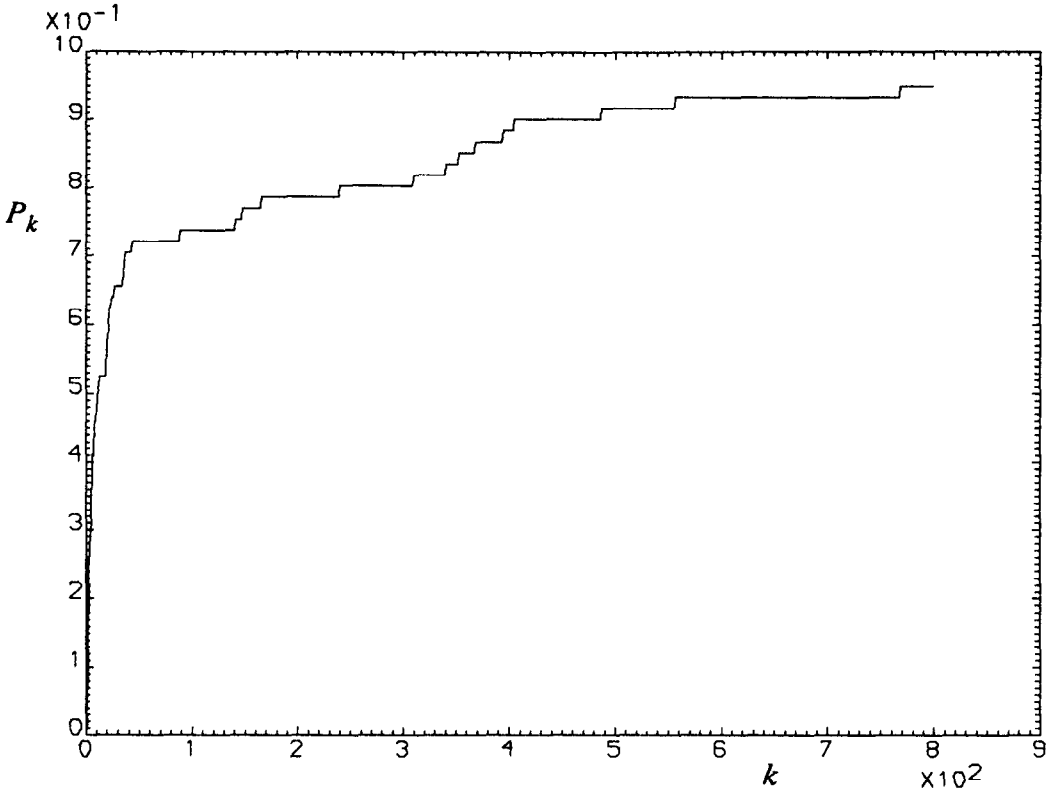

Figure 2

application of ESPM to an arc vw involved generating in turn the 2^{nd}, 3^{rd}, ... , k^{th} shortest path through vw, where $k = \min(r, 800)$ and the shortest feasible path through vw is the r^{th} shortest path through that arc. The results obtained are summarised in the form of the graph of figure 2.

The graph is a plot of P_k against k for k in the range [1,800] where:

$$P_k = \frac{\sum_{m=1}^{13} \sum_{ij \in U_m} I_k(m, ij)}{\sum_{m=1}^{13} |U_m|}$$

U_m is the set of arcs in the DD-graph of subroutine m which SPM did not cover, and $I_k(m, ij)$ is defined by:

$$I_k(m, ij) = \begin{cases} 0 & \text{if arc } ij \text{ of subroutine } m \text{ has not been covered by the first } k \text{ shortest paths through that arc} \\ 1 & \text{otherwise} \end{cases}$$

Using the notion that a path through a DD-graph is feasible if the path through the corresponding subroutine is feasible, P_k is essentially a measure of the probability that one of the first k shortest paths through an arc, as generated by ESPM, is feasible.

As can be seen from the graph, the increase in P_k is considerable for smaller values of k ; P_k attaining a value of 0.71 at $k = 44$. For k in excess of this value, the rate of increase of P_k begins to slow, and to slow markedly after $k \cong 400$. Of the 61 arcs not covered by SPM, the number covered by ESPM at $k = 20$, 44, 400 and 800 are: 35, 44, 54 and 58 respectively. These figures and the nature of the graph appear to substantiate the assertion made in section 2.0 as well as demonstrating the efficacy of ESPM. They also seem to suggest that, even if it is desirable, it is not cost-effective to generate more than about 40 paths in an attempt to cover any arc, and that the cover thus achieved is the maximum attainable without a substantial investment of additional effort.

Although P_k gives an indication of ESPM's effectiveness with respect to finding a feasible path through a single arc, it is important to answer the question: how many paths in total is it likely that SPM and ESPM will have to generate to achieve the maximum value of Ter_2 for a given code unit? In an attempt to estimate this value, n^* say, the value of V , the mean number of paths per arc generated in achieving a maximum cover, was first calculated for the 13 subroutines to which ESPM was applied. In defining a "maximum cover", the above suggestion: to generate no more than 40 paths through any arc in an attempt to find a corresponding feasible path, was adopted. For several of the 13 subroutines $|U_m| > 1$, and so the problem discussed in section 4.0, namely that of ordering the $|U_m|$ arcs for subsequent application of ESPM, then arises. In order to derive an accurate (unbiased) value of V it was necessary to consider all $|U_m|!$ orderings for each subroutine m and thereby evaluate V as:

$$V = \frac{\sum_{m=1}^{13} T_m}{\sum_{m=1}^{13} |U_m| \sum_{m=1}^{13} |U_m|!}$$

where

$$T_m = \sum_{n=1}^{|U_m|!} \sum_{ij \in U_m} \min(40, k_{mn}(ij)) \ ,$$

and $k_{mn}(ij)$ is the number of paths through arc $ij \in U_m$ that are generated by both SPM and ESPM in order to cover ij when ESPM is applied to the arcs of U_m in the order prescribed by the n^{th} ordering of these arcs.

The values of the T_m are given in the appendix, and using these V is calculated to be 5.48. Since V includes a contribution of unity corresponding to the single path generated by SPM, it can thus be said that the average number of paths per arc which must be generated by ESPM to attain a ' maximum branch cover ' (as defined by generating at most 40 paths per arc) is 4.48.

Given the value $V = 5.48$, an estimate of n^* which answers the question posed above may then be calculated as:

$$n^* = n_1 + 5.48\ n_2$$

Here n_1 is the mean number of arcs per subroutine covered by SPM (ie using only 1 path), and n_2 is the mean number of arcs per subroutine to which ESPM was applied (the means being calculated using all of the 22 subroutines studied). Since the mean number of arcs per subroutine is 180/22 = 8.18, and of these ESPM was applied to an average of $n_2 = 61/22 = 2.77$, n_1 is calculated to be 5.41. Using these values for n_2 and n_1, it is found that the mean number of paths generated by SPM and ESPM in achieving a maximum cover is $n^* = 20.59$, hence requiring only a mean of 2.52 paths per arc, or 1.39 paths per branch.

Finally, the mean maximum cover obtained for the 13 subroutines in using ESPM to generate an average of 4.48 extra paths per uncovered arc was 0.89, an increase of 0.34 over that obtained using SPM, and the corresponding mean value of Ter_2 for all 22 subroutines was 0.93.

7.0 Summary and Conclusions

In this paper a path generation strategy (MPS), which aims at minimising the number of infeasible paths generated during branch testing, has been proposed. The strategy is founded on the assertion that the fewer the predicates a program path contains, the more likely it is that the path is feasible. In an attempt to validate the assertion, and thereby justify the strategy, statistical tests were performed on a sample of 642 program paths. The results of these tests make it possible to conclude with extreme confidence that there is some form of relationship between the feasibility of a path and the number of predicates it involves, and that an exponential decay in feasibility with increasing predicate involvement represents a most

acceptable model of this relationship as observed in the sample.

Although the result due to Weyuker (see section 3.0) precludes there being a deterministic relationship between path feasibility and path predicates, the authors maintain that the results of the statistical tests performed provide significant evidence corroborating the assertion that has been advanced. If the assertion is valid, and there is no reason to suppose that it is not, the number of predicates a path possesses can be used as a heuristic to gauge a paths likely feasibility. Moreover, the efficiency of any path generation strategy which, in its formulation, takes no account of infeasible paths must come under question. This point is also emphasised by the strong suggestion from the results of section 5.0: that it is unlikely to be cost-effective to generate more than (about) the first 40 shortest paths through a branch in order to cover it.

To investigate the efficacy of MPS, SPM and ESPM, the path generation methods embodying the strategy, were applied to a set of 22 FORTRAN subroutines. The effectiveness of the strategy is attested by the branch cover obtained: SPM achieved full coverage of 9 subroutines as well as a mean cover of 0.73, and ESPM increased this to a maximum cover of 0.93 with less than an average of one branch per subroutine remaining uncovered. Additional evidence is provided by the fact that the value of $Ter_2 = 0.93$ was attained by generating only an average of 1.39 paths per branch. The authors contend that these results represent a significant indication that MPS does tend to 'minimise' the number of paths generated during branch testing and, thereby, does reduce the associated time and cost overheads.

Acknowledgement

The authors express their thanks to NAG Ltd for permission to access the source code version of their Mark 11 FORTRAN library and to Liverpool Data Research Associates for the use of their automated tool for abstracting basic blocks from FORTRAN code.

References

[1] White, L. J. "Software Testing and Verification", Advances in Computers, M. C. Yovits ed., 26, 1987.

[2] Howden W. E. "Empirical Studies of Software Validation", Tutorial: Software Testing and Validation Techniques, E. Miller and W. E. Howden eds., IEEE, 1978.

[3] Yates, D. F. and Hennell M. A. "An Approach to Branch Testing" Proc 11th International Workshop on Graph Theoretic Techniques in Computer Science, Wurtzburg, 1985.

[4] Chatfield, C. " Statistics for Technology ", Chapman and Hall, New York, 1970.

[5] Weyuker, E. J. "The Applicability of Program Schema Results to Programs", Int. J. Comput. Inf. Sci., 8, 1979.

[6] Brown, J. R. "Practical Application of Automated Software Tools", TRW Report TRW-SS-72-05, TRW Systems, One Space Park, Redondo Beach, California, 1972.

[7] Malevris N. Ph.D. thesis, University of Liverpool, U.K., 1988.

[8] Dreyfus S. E. "An Appraisal of Some Shortest-Path Algorithms", Operations Research, 17, 1969.

[9] Shier D. R. "Iterative Methods for Determining the K Shortest Paths in a Network", Networks, 6, 1976.

[10] Minieka E. "On Computing Sets of Shortest Paths in a Graph", Comm. ACM, 17, 1974.

Appendix

m	Subroutine	T_m
1	C02ADZ	2
2	C06AAZ	2
3	C06ABZ	343x11!
4	C06DBF	2
5	C06EBT	27.375x8!
6	E01AAF	96
7	E02BBF	2
8	F01AHF	8.43x7!
9	F01AZF	4.8x5!
10	F01CSF	720
11	F01BEF	44
12	F04AQF	243.75x10!
13	G04ADF	27.375x8!