Adding Contextual Guidance to the Automated Search for Probabilistic Test Profiles

Hélène Waeselynck, Simon Poulding

Adding Contextual Guidance to the Automated Search for Probabilistic Test Profiles

Simon Poulding
Department of Computer Science,
University of York, York, UK
Email: simon.poulding@york.ac.uk

Hélène Waeselynck
LAAS-CNRS,
Univ. Toulouse, Toulouse, France
Email: helene.waeselynck@laas.fr

***Abstract*—Statistical testing is a probabilistic approach to test data generation that has been demonstrated to be very effective at revealing faults. Its premise is to compensate for the imperfect connection between coverage criteria and the faults to be revealed by exercising each coverage element several times with different random data. The cornerstone of the approach is the often complex task of determining a suitable input profile, and recent work has shown that automated metaheuristic search can be a practical method of synthesising such profiles.**

The starting point of this paper is the hypothesis that, for some software, the existing grammar-based representation used by the search algorithm fails to capture important relationships between input arguments and this can limit the fault-revealing power of the synthesised profiles. We provide evidence in support of this hypothesis, and propose a solution in which the user provides some basic contextual knowledge to guide the search. Empirical results for two case studies are promising: knowledge gained by a very straightforward review of the software-under-test is sufficient to dramatically increase the efficacy of the profiles synthesised by search.

I. Introduction

Statistical testing is a probabilistic approach for test generation pioneered by Thévenod-Fosse and Waeselynck in the 1990s [1]–[3]. Its premise is to compensate for the imperfect connection of common test adequacy criteria with the faults to be revealed: the coverage elements identified by a criterion are exercised several times with different random data. As a result, there is no need for a perfect match between coverage elements and fault-revealing inputs.

The approach requires an input profile that typically differs from the uniform profile that is used in (blind) random testing, and from the operational profile, i.e. the distribution of inputs that the software will encounter in operation [4], [5]. Statistical testing instead utilises a profile that is optimised to both the chosen adequacy criterion and the specific software-under-test (SUT) so as to enable efficient structural or functional coverage.

Statistical testing has been shown to be very effective at revealing faults, both mutations as well as genuine faults, in software ranging in size up to thousands of lines of code. However, the determination of a profile that is effective for the chosen adequacy criterion remains a complex, and so potentially costly, optimisation problem. To address this problem, Poulding et al. have recently demonstrated an automated search-based technique for synthesising profiles [6], [7]. The input profile is represented by a stochastic grammar which is incrementally modified using a metaheuristic search algorithm until the adequacy criterion is met.

The motivation for this paper is the observation that while automated search can synthesise profiles quickly and cheaply, the test sets generated from the synthesised profiles may demonstrate surprisingly poor fault-revealing power. This is because the search is tasked only with optimising the coverage required to satisfy the adequacy criterion, not the ultimate objective of revealing faults. As a result, it is possible for the synthesised profile to be 'degenerate' in the sense that inputs are sampled from only a small region of the input domain: such a degenerate profile can satisfy the adequacy criterion in terms of coverage, but the SUT is exercised with inputs that are not sufficiently diverse to reveal some of the faults.

Our hypothesis is that degeneracy arises because the existing grammar-based representation is unable to properly express the relationships between input arguments that govern the logic implemented by the software-under-test. In an attempt to maximise coverage, the search attempts to approximate these relationships but is only able to do so over small regions of the input domain.

In this paper we propose that degeneracy may be reduced, and therefore the fault-revealing power of profiles improved, by augmenting the grammar with new operators that express elementary relationships between input arguments. We envisage that the user will be able to choose which of these operators to incorporate into the grammar for a specific SUT based on a straightforward review of the specification or implementation: the operators enable the user to provide contextual knowledge that guides the search algorithm. The results of two case-studies demonstrate that augmenting the grammar with our proposed operators significantly increases the fault-revealing power of profiles synthesised by automated search.

In section II, we explain the adequacy criterion of statistical testing and the automated search algorithm that synthesises input profiles to satisfy this criterion. We expand on our hypothesis of degeneracy in search-synthesised profiles in section III, and demonstrate two case studies that support this hypothesis in section IV. In section V, we propose the new operators that augment the grammar representation, and in section VI we evaluate the improved fault-revealing power enabled by the augmented grammars. We discuss our approach in relation to other automated techniques for statistical testing in section VII. In section VIII, we reflect on the impact of augmenting the profile representation and outline future work.

II. Background

A. Statistical Testing

The key feature of statistical testing is an adequacy criterion that constrains the properties of the input profile from which test inputs are sampled. As for deterministic coverage testing, the criterion is expressed in terms of a finite set of coverage elements, $\mathcal{C}$, that are based on the structure of the implemented code, e.g. control-flow branches [1], or the functionality that the SUT should exhibit [2].

Let p_c denote the probability that coverage element $c \in \mathcal{C}$ is exercised by a test input sampled at random from the profile, and p_{min} be the lowest of all the p_c over the set $\mathcal{C}$. The value of the minimum coverage probability, p_{min}, influences the test size, N, at which all coverage elements are exercised by a test set generated from the profile. Since the sampling of inputs is stochastic, full coverage can never be *guaranteed*; instead let $0 \leq Q < 1$ denote the probability that the least likely element (the element with lowest value of p_c) is covered by the test set. Then, these values are related by the equation:

$$1 - Q = (1 - p_{\text{min}})^N \tag{1}$$

Therefore if the number of test inputs is fixed—for example, if the cost of applying the oracle is too expensive to increase the test size—a high value of p_{min} increases the likelihood that all coverage elements are exercised at least once. Alternatively, a value of Q may be chosen in advance and the test set sized to achieve this likelihood; in this case a high value p_{min} reduces the number of test cases required. Thus the objective is for p_{min} to be as high as possible.

Although the relationship between coverage criteria and fault exposure will be imperfect, it is reasonable to expect that a profile which uses some information about the SUT to exercise all elements as frequently as possible will typically reveal faults more efficiently than the uniform distribution that uses no such information. Indeed, Thévenod-Fosse and Waeselynck demonstrate the superior fault-revealing power of statistical testing over uniform random testing for test sets of the same size [1], [2].

For values of Q close to 1, not only is it likely that every coverage element is exercised by at least one test case in the test set, but many coverage elements will be exercised by multiple test cases. This is a significant advantage of statistical testing: the imperfect relationship between coverage criteria and fault location can be compensated for by exercising each coverage element with multiple *different* inputs. Moreover, it is relatively easy to generate a test set that has this property by simply sampling enough inputs at random from the profile. In contrast, deterministic coverage testing techniques often derive a minimal test set in which many of the coverage elements are exercised only one. Thévenod-Fosse and Waeselynck show that this posited advantage is demonstrated in practice: test sets generated by statistical testing can reveal more faults that minimal sets generated by deterministic techniques [1].

B. Synthesis of Profiles By Automated Search

It is possible to determine the optimal profile by analytical analysis for relatively simple SUTs and coverage types [1]. Alternatively, a suitable profile may be found by manual experimentation: based on the coverage induced by a sample of inputs drawn from a candidate profile, the profile is iteratively refined until the minimum coverage probability, p_{min} is deemed sufficiently high. The latter approach was successful in cases for which analytical analysis was not practical [2] but at the expense of additional human effort.

To minimise costly human effort, Poulding et al. have developed an automated technique for profile synthesis using metaheuristic search. In their initial work, the search algorithm optimised a representation of the input profile as a Bayesian network in which node represented one of the input arguments to the SUT [6]. However, this representation has limited applicability: it can be used only if the SUT's input domain consists of a fixed number of numeric input arguments. In subsequent work they have used a more flexible grammar-based representation that is capable of representing highly-structured inputs that may include variable-length compound datatypes and contain both numeric and categorical data [7]. It is the search-based algorithm using this representation that we use as the basis for our work in this paper, and that we describe in this section.

1) Grammar-Based Representation: The representation is an enhanced form of stochastic context-free grammar. A grammar, specific to the SUT, defines the inputs that form the SUT's input domain using a set of production rules. In the following example grammar, `S`, `X` and `Y` are variable symbols; and `'a'` to `'d'` are terminal symbols (distinguished from variables by the use of single quotes):

```
S → X
X → 'a' | 'b' Y
Y → 'c' X | 'd' X
```

Starting with, by convention, a sequence consisting of the single variable `S`, the production rules specify how variables may be replaced with other symbols until the sequence of symbols consists only of terminals. In the above example, there are two production rules for `X` separated by the notation `|`. The first rule replaces `X` with the terminal `'a'`; the second replaces `X` with the terminal `'b'` followed by the variable `Y`. The set of terminal sequences that could be emitted from the grammar—with an appropriate choice of production rule whenever more than one could be applied—corresponds to the set of inputs to the SUT.

In order to represent a profile, i.e. a probability distribution over the input domain, the production rules are annotated with weights to create a stochastic grammar. Each time a variable is encountered with more than one production rule, one of the rules is selected at random according the weights. For example, if the rules `X → 'a'` and `X → 'b' Y` in the example grammar above have weights 0.7 and 0.3 respectively, a sequence sampled from the grammar is likely to contain more `'a'` than `'b'` terminals.

The representation includes two additional features not present in standard stochastic context-free grammars and which improve the efficiency with which search is able to synthesis profiles that meet the adequacy criterion [7]. We outline both features here, and refer the reader to [7] for further details.

Firstly, weights may be conditionally dependent on the production rules that were most recently applied by other

variables. This enables a limited form of context-sensitivity. For example, if the rule `X` $\rightarrow$ `'a'` has a probability of 0.8 when the most recently applied rule for Y is `Y` $\rightarrow$ `'c' X`, and 0.4 when it is `Y` $\rightarrow$ `'d' X`, the subsequence `'ca'` is more likely to occur than `'cb'`, but `'db'` is more likely than `'da'`. These conditional dependencies are not (necessarily) specified by the user: they are instead explored by the search algorithm.

Secondly, a compact notation is introduced when terminal symbols represent a large range of numeric values. The motivation for this enhancement is that the following idiom for numeric variables,

```
Num → '0' | '1' | '2' | '3' | '4'
```

does not scale well when the range to be represented is, say, $[0, 999]$ rather than $[0, 4]$: it would require an excessively large number of production rules and associated weights that would not be amenable to optimisation by search. In such cases, we introduce a special notation whereby the production rules refer to a set of intervals that partition the range:

```
Num → [0,422] | [423,543] | [544,999]
```

When one of these production rules is chosen according to the weights, such as `Num` $\rightarrow$ `[544,999]`, the terminal symbol emitted is a number sampled at random from the chosen interval (using a uniform distribution), e.g. `'604'`. The user need only specify the full range (e.g. `Num` $\rightarrow$ `[0,999]`) in the grammar to be optimised; the search algorithm explores suitable partitions over this range. We refer to grammar variables represented in this way as 'partition variables'.

2) Fitness Evaluation: In order to compare candidate profiles, the search algorithm requires a fitness metric that quantifies how 'good' each profile is. Since the objective is to synthesise a profile with a high value of the minimum coverage probability, p_{min}, in order to satisfy the adequacy criterion, the chosen fitness metric is an estimate of this probability.

In order to calculate this estimate, a set of inputs, size K, is sampled from the candidate profile and used to execute an instrumented version of the SUT. For each input in the sample, the instrumentation identifies which of the coverage elements are exercised by that input. (There is no need to record nor check the output of the instrumented SUT using an oracle.) An estimate, $\widehat{p_c}$, of the coverage probability, p_c, for coverage element c, is the proportion of the K sampled inputs that exercise c. The fitness metric—the estimate of the minimum coverage probability—is then lowest value of $\widehat{p_c}$ across the set of coverage elements, $\mathcal{C}$.

3) Search Algorithm: The metaheuristic search algorithm used to optimise the input profile is random mutation hill-climbing. The hill-climb begins from a single randomly-generated 'current' profile. During each iteration of the algorithm a small number of neighbouring profiles are created by making small, random changes ('mutations') to the current profile. The fitness of each of the neighbouring profiles is evaluated using the metric described above, and if the best of the neighbouring profiles (the one with the highest fitness) is better than current profile, that neighbouring profile becomes the current profile in the next iteration of the algorithm. For the work described in this paper, the search terminates after a fixed number of iterations and returns the profile with the highest fitness encountered during the search process[1].

The search is able to mutate the following parts of the grammar: the production rule weights, the conditional dependencies between variables, and the number and length of intervals of each partition variable. These mutations, and the algorithm parameters that are related to them, are described in detail in [7].

4) Efficacy of the Search-Based Technique: Poulding et al. evaluated the capabilities of the algorithm and grammar representation on a set of three diverse SUTs [7]. This work demonstrated the algorithm's effectiveness in terms of performance (and thereby cost): profiles could be synthesised in approximately 20 minutes or less using hardware typical of a desktop PC. However, this work did not evaluate whether the search algorithm optimised the profiles for fault-revealing power, in addition to optimising for coverage.

In their earlier work using the Bayesian network rather than grammar representation for profiles, there was evidence that the fault-revealing power of synthesised profiles could be relatively poor: for some SUTs, random testing using the uniform distribution found more faults at large test sizes [6]. It is our concern that the synthesised profiles using the grammar representation continue to have relatively poor fault-revealing power that motivates for the research described in the remainder of this paper.

III. Hypothesised Problem

Our hypothesis is that—for some SUTs—optimal profiles may be difficult to represent because parts of the SUT's functionality are guarded by predicates requiring relationships between the input arguments that are not easily expressed by the grammar representation. As a result, the profiles synthesised by search have a relatively poor fault-revealing power.

We came to this hypothesis by taking examples of optimal profiles from Thévenod-Fosse and Waeselynck's work, and considering how they would be encoded into grammar representation used by Poulding et al. For optimal profiles that required certain simple relationships between the input arguments, the grammar encoding is so convoluted that we conjecture it unlikely that the search would synthesise it in a realistic time.

As an illustration, consider a SUT with two numeric arguments, each taking values between 0 and 999. We could represent the input domain using the following grammar, where grammar variables `X1` and `X2` correspond to the two input arguments:

```
 S → X1 X2
X1 → [0,999]
X2 → [0,999]
```

Let us assume that there are two main subroutines in the SUT: one that implements functionality for the case when the two

[1]Normally in a hill climbing search, the fittest profile would be the current profile at the point of termination. However, for reasons related to a enhancement employed to accommodate sampling noise in the evaluation of fitness, this is not necessarily the case here. We do not have space to describe the enhancement in this paper, but refer the reader to [6].

input arguments, X1 and X2, are equal, and the other when they are not; and that these subroutines are guarded by the branch predicate X1==X2. Let us also assume an optimal input profile would exercise these two subroutines with equal frequency, i.e. we would like the search to exhibit a profile such that X1==X2 occurs with probability 0.5.

Using the stochastic grammar representation, this might be achieved by first partitioning the two variables into 1000 intervals, one for each integer:

```
X1 → [0,0] | [1,1] | ... | [999,999]
X2 → [0,0] | [1,1] | ... | [999,999]
```

The required probability distribution could then be achieved by making `X2` dependent on `X1` such that if the interval sampled for `X1` was $[m,m]$, then the interval $[m,m]$ for `X2` is annotated with a weight of 0.5, while the remaining weight is shared over the other intervals of `X2`. In this grammar, the probability of input arguments X1 and X2 being equal is 0.5, and the value that X1 and X2 share could be any between 0 and 999. This is the closest representation of the kind of profiles derived by Thévenod-Fosse and Waeselynck which required a similar equality relation between two of the input arguments.

Alternatively, a near-optimal profile could be achieved with much simpler grammars, such as:

```
X1 → [0,217] | [218,218] | [219,999]
X2 → [0,217] | [218,218] | [219,999]
```

In this grammar, the interval `[218,218]` for both variables would have a probability of $1/\sqrt{2}$ and there is no dependency between the two variables. Thus, input arguments X1 and X2 are equal, with a shared value of 218, with a probability 0.5. There is only a small chance the arguments are equal but share a value other than 218, and so the probability that X1 and X2 are unequal is also approximately 0.5.

Both these grammars are near-optimal in terms of exercising the two subroutines with equal probability, and would both meet the statistical testing adequacy criterion. However, we would expect the first grammar to reveal more faults since it exercises the subroutine guarded by X1==X2 using a diverse range of values, while the second grammar would exercise it with the value 218 in the majority of test cases. The benefit of statistical testing is that each coverage element is likely to be exercised multiple times, each time with a *different* input; the second 'degenerate' grammar would not demonstrate this property.

The search algorithm would assess both these grammars to have similar minimum coverage probabilities, and therefore similar fitnesses. But we hypothesise that the first grammar would require very many iterations to synthesise using the mutations described in section II-B: each variable must be partitioned into 1000 rules, and 1000 conditional weights for `X2` must take the correct values. In contrast, a more parsimonious grammar similar to the second would be much easier to synthesise using search, and so a more likely output of the search algorithm. However it would express a degenerate profile that, in practice, reveals fewer faults than a more diverse profile with the same minimum coverage probability. Such a poor fault-revealing power would reduce the economic case for using search to synthesise profiles, and so motivates a solution that avoids degeneracy.

IV. Demonstration of Hypothesised Problem

In this section we provide empirical evidence in support of the hypothesis described in the previous section: that profiles synthesised by automated search can show relatively poor fault-revealing power despite satisfying the adequacy criterion required by statistical testing.

A. Case Study SUTs

We collect evidence from two case study SUTs: FCT3 and TCAS. Since the objective is to demonstrate that there exist one or more SUTs for which the search-based technique synthesises profiles that have relatively poor fault-revealing power—rather than to evaluate the technique's average behaviour across all SUTs to which it could be applied—we subjectively chose these two SUTs because we suspect their optimal input profiles are difficult to represent using the grammar-based representation.

FCT3 is a C function that forms part of a nuclear reactor safety shutdown system. This SUT was used by Thévenod-Fosse, Waeselynck and Crouzet in their evaluation of statistical testing [1]. Its purpose is to filter out doubtful measures of the position of the reactor's control rods. The function takes 8 integer input arguments; a grammar representing the input domain is shown in figure 1a. The argument X1 is the current measure of a rod position and X4 the previous measure. A major requirement is that a measure is not considered stable unless successive identical values are read. Hence, the function challenges the search process to account for relation X1==X4.

To assess the fault-revealing power of the synthesised profiles, we use a set of 1355 mutants—versions of the SUTs with systematically-made syntax changes—created as part of the early work on investigating statistical testing for this SUT [1]. All of these 1355 mutants are non-equivalent, meaning that they are not functionally equivalent to the original SUT, and therefore each one could be detected (or 'killed') by an incorrect output when supplied an appropriate test input.

The C function TCAS implements the resolution logic for an early version of a collision avoidance system used in commercial aircraft. It aims to maintain the vertical separation between aircraft that is greater than a specific threshold, and to decide whether a downward or upward manoeuvre would yield the greatest separation. This SUT is widely used in software testing research, and is provided by the Software-artifact Infrastructure Repository [8]. The function takes 12 integer input arguments; a grammar representing the input domain is shown in figure 1b. The argument `UpSep` is the vertical separation that would be induced by an upward manoeuvre, `DownSep` the one induced by a downward manoeuvre, and `AltLayVal` selects the threshold to maintain (it indexes an array of thresholds). We expect this function to be challenging since it requires the discovery of suitable 'greater than' and 'less than' relations between these input arguments. The source code of the function also contains numerical comparisons between other input arguments and constants, and complex logic that combines all these comparisons in order to derive the output.

The Software-artifact Infrastructure Repository provides 41 variants of TCAS, created by manually seeding faults, and we

```
 S → X1 X2 X3 X4 X5 X6 X7 X8
X1 → [0,63]
X2 → [0,1]
X3 → [0,1]
X4 → [0,63]
X5 → [0,63]
X6 → [0,1]
X7 → [0,1]
X8 → [0,15]
```

(a) FCT3

```
 S → CurVerSep HighConf TwoRepVal
     OwnTrkAlt OwnAltRat OthTrkAlt
     AltLayVal UpSep DownSep
     OthRAC OthCap CliInh
CurVerSep → [0,1999]
 HighConf → [0,1]
TwoRepVal → [0,1]
OwnTrkAlt → [0,9999]
OwnAltRat → [0,999]
OthTrkAlt → [0,9999]
AltLayVal → [0,3]
    UpSep → [0,1999]
  DownSep → [0,1999]
   OthRAC → [0,2]
   OthCap → [1,2]
   CliInh → [0,1]
```

(b) TCAS

Fig. 1: The grammars defining the input domains of the case study SUTs.

use these variants to assess the fault-revealing power of input profiles synthesised for this SUT.

B. Search Algorithm Settings

We use the same algorithm configuration and parameter settings as in the most recent work of Poulding et al. [7]. These settings are based on earlier work that tuned the configuration and parameters to minimise the time and resources required to synthesise a profile [9]. Since there was not space to provide a full description of the algorithm and its parameters in section II, we omit details of the specific parameter settings here and instead refer the reader to [7].

In this work, the search terminates after a fixed number of iterations—rather than terminating once a profile target minimum coverage probability is found—as we believe this is consistent with how the algorithm would often be used in practice. For a previously-unseen SUT a suitable target probability would not be known and so the algorithm would be run for a fixed time to obtain a 'best effort' profile. A limit of 100,000 iterations was chosen based on initial experimentation that showed that the fittest profiles could consistently be reached well within this limit. When the algorithm is run on hardware typical of a desktop PC, 100,000 iterations corresponds to a run time of approximately 2.5 minutes for FCT3, and approximately 5 minutes for TCAS.

For FCT3, the adequacy criterion was coverage of all 19 branches. For TCAS, the criterion was coverage of all 62 conditions, including those in assignment statements.

C. Empirical Method

The following method was performed for each SUT:

1) The algorithm was run 10 times, each run with a different seed to the pseudo-random number generator, and the profile synthesised by the algorithm collected. By running the algorithm multiple times, variance in the results arising from the stochastic nature of the algorithm is reduced.

2) One set of random test inputs was sampled from each of the 10 synthesised profiles. The size of the set was (somewhat arbitrarily) chosen at 405 for FCT3, corresponding to the size that enables coverage of all paths with a probability of 0.9999 for a profile derived by analytical analysis [1]; and 1608 for TCAS, corresponding to the size of the 'universe' test set provided by Software-artifact Infrastructure Repository.

3) The number of mutants (or variants) killed by each of the 10 sets was assessed. The count of killed mutants was recorded after the first test input in the set, then after the second test input, and so on until the set had been exhausted. This enables a comparison of the mutation scores—-and therefore estimates of the fault-revealing power of the profiles—at different test sizes.

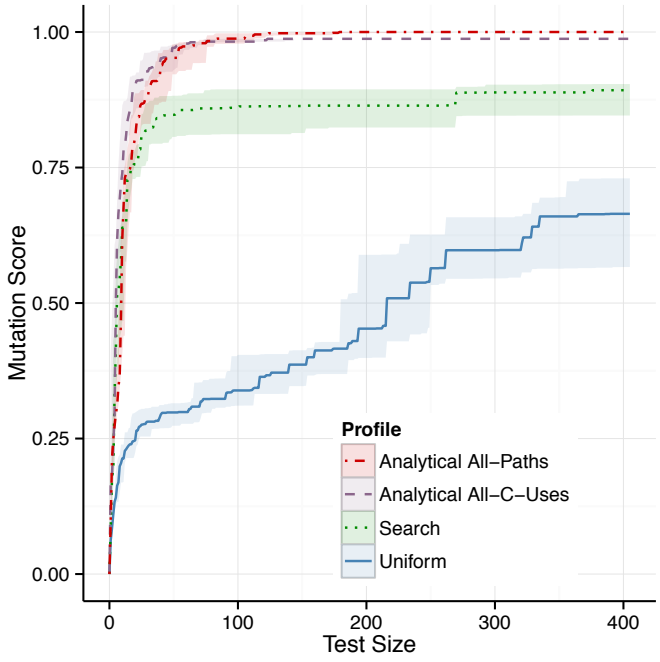
4) As a baseline, 10 sets were sampled from the uniform profile (a uniform distribution over the input domain) and their mutation score assessed as above. To create a uniform distribution, the stochastic grammars of figure 1 are used without applying search: since there is only a single interval for all partition variables, the values sampled for each input argument are from a marginal uniform distribution, and there is no dependency between the variables.

For FCT3 only, we use two further baselines: two profiles derived analytically for all-paths and all-c-uses adequacy criteria in earlier work [1]. In this earlier work, the all-paths sets consistently killed all mutants, while the all-c-uses ones consistently killed all but 17 mutants. These 17 mutants are missed due to some lack of diversity in the latter profile: ignoring the contribution of two paths simplified the analytical derivation of the all-c-uses profiles, hence the corresponding input configurations were assigned a zero probability and this has a direct impact on the revealing conditions of the 17 mutants. Thus, the all-c-uses profile can be seen as an early example of degeneracy while the all-paths one exemplifies a structural profile that is diverse by construction.

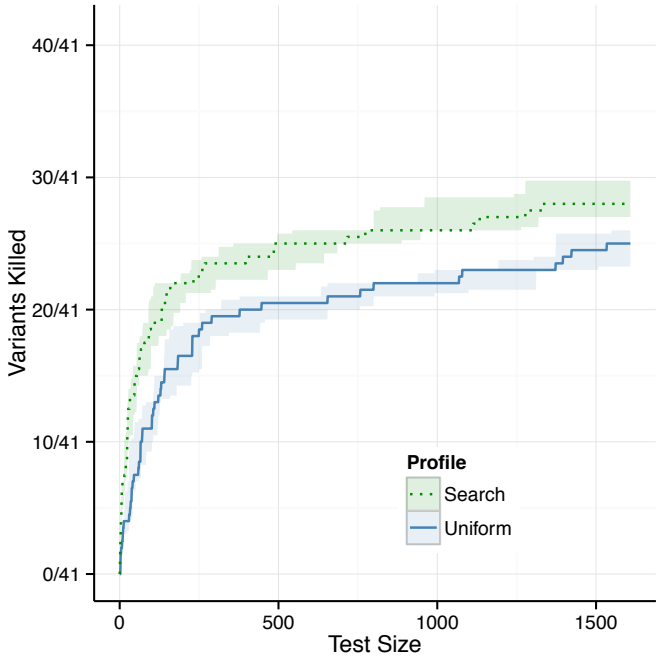
D. Results and Discussion

The results are summarised in figure 2 which shows the median mutation scores at different test sizes for each type of profile. The number of mutants tested for FCT3 is sufficiently large at 1355 to present the results as a mutation score, i.e. the proportion of mutants killed by the test set. For TCAS only 41 variants were tested and so only a limited number of distinct mutation scores are possible; therefore we present results for TCAS as the actual number of variants killed.

For FCT3, the profiles synthesised by search have a higher fault-revealing power than the uniform profiles for all test sizes. At all test sizes, the differences are statistically significant using the Wilcoxon-Mann-Whitney test to compare the ten samples of each type of profile. While the search-synthesised



(a) FCT3



(b) TCAS

Fig. 2: The mutation score (FCT3) and number of variants killed (TCAS) plotted against the size of test set. The lines indicate the median score, and the shaded ribbon the interquartile range.

profiles are similar to the analytically-derived all-paths and all-c-uses profiles for the very small test sizes, they are outperformed by the analytically-derived profiles and fail, even at the largest test sizes, to kill all the mutants. This behaviour is consistent with our hypothesis that the search-synthesised profiles are degenerate: at small test sizes, the lack of diversity caused by degeneracy is not apparent, but in larger tests, there is insufficient diversity in the test inputs to kill the mutants whose revealing conditions are loosely related to the coverage conditions imposed by the adequacy criterion. Note that the adoption of a weak criterion (all-branches) for the search is not by itself an explanation for the observed results: for FCT3's source code, all-branches subsumes all-c-uses [1], hence the search-synthesised all-branches profile would be normally expected to outperform the analytical all-c-uses one.

For TCAS, the profiles synthesised by search have a higher fault-revealing power than the uniform profile, and the difference is again statistically significant. For this SUT, however, the magnitude of the difference is small. Moreover, for the ten search-synthesised sets, between 7 and 16 of the 41 variants remain unkilled even for test sets with extremely large sizes of 1608. The relatively poor fault-revealing power of the search-synthesised profile compared to the uniform profile, despite a much higher minimum coverage probability, is again consistent with our hypothesis of degenerate profiles.

In order to further illustrate potential degeneracy, let us first consider the search-synthesised profiles for FCT3. As discussed previously, a key requirement of this function is to compare the current and previous measures (X1 and X4, respectively) and this requirement corresponds to a branch predicate in the code that tests their equality. In all of the ten search-synthesised profiles, the branch requiring equality was exercised often, but every time using the *same* value of X1 and X4 (the specific value differed between profiles). This lack of diversity is unlikely to reveal many faults in the code guarded by this branch.

Degeneracy is less easy to demonstrate for TCAS, as the function performs many numerical comparisons. If we focus on the conditions comparing `UpSep` and `DownSep` to a selected threshold, we observe that in three of the search-synthesised profiles, 90% of the comparisons occur using the same value of threshold. This includes the profile with lowest mutation score, where the same threshold value is selected for more than 99% of the comparisons. We speculate that this degeneracy may exist because the coverage of a condition of the form $X < Y$ is easier to control if one of the values is fixed. While it is not possible to draw conclusions as to the precise impact the threshold selection has on the fault revealing power, we consider it an issue that some synthesised profiles do not show diversity in the threshold value.

V. Proposed Solutions

In this section we propose two complementary solutions to reduce the problem of degenerate profiles that show poor fault-revealing power. In the initial work of Poulding et al. [6], there was some evidence of degeneracy in the profiles synthesised for SUTs other than the two considered in this paper, and the authors proposed a solution whereby a simple diversity metric is included in the fitness used by the search algorithm.

While this enhancement did improve the fault-revealing power, the effect was small. Here we propose two new solutions: operators that augment the grammar in order to better express relationships between input variables that would otherwise be difficult to represent; and a constraint on the production weights in the grammar.

1) Grammar Operators: Our hypothesis of section III, supported by empirical evidence in section IV, was that one source of degeneracy is the inability of the grammar to concisely express some basic relationships between input arguments. To address this problem, we propose to augment the grammar with new operators that express these relationships. As a proof of concept, we focus on relationships between numerical integer arguments, which are exemplified by FCT3 and TCAS.

The proposed operators take the form of additional terminal symbols in the grammar, and express relationships in terms of a logical comparison between an input argument and an expression built from other input arguments and constants, such as X1 being equal to the value taken by X2+1; or X1 being less than a constant. In our current implementation, the relationships are enforced in post-processing of the terminals sampled from the grammar; we intend in future work to extend the grammar with a notation for these relational operators and arithmetic expressions to avoid the need for much of this post-processing.

For example, we might augment the simple grammar of section III with the rules indicated in bold to produce the following:

```
     S → X1 X2 X1Aug
    X1 → [0,999]
    X2 → [0,999]
X1Aug → '==X2+1' | '<LIMIT' | 'none' |
```

Here the production rules for variable `X1Aug` indicate which relationship to apply. The terminal `'==X2+1'` indicates that the input argument X1 should be set equal to value already sampled for argument X2, plus 1, in post-processing. The terminal `'<LIMIT'` indicates that X1 should taken a value randomly sampled from a uniform distribution over the interval $[m, \text{LIMIT})$ where m is the lower end of the valid range for argument X1 and LIMIT is a constant used in the SUT. The terminal `'none'` indicates that no relationship is enforced: the value emitted for the partitioned variable `X1` in the original part of the grammar should be used for the input argument X1 without change. Including this terminal in the augmentation enables the search algorithm, should it be desirable, to avoid the specified relationships by weighting this production rule suitably high.

The augmented grammar now has the capacity to concisely represent profiles arising from previous work on statistical testing, notably the all-paths and all-c-uses profiles of FCT3. For synthesised profiles, we do *not* propose that the search determines the comparison operators and expressions automatically. Our initial experimentation with this approach suggested that it is unlikely to be practical: even when severely constraining the form of the expressions to explore, the search was unable to provide useful results in a realistic amount of time. Instead, we propose that the user suggests potentially useful relationships under the form of additional production rules. The search has then to find an appropriate weighting for these rules in combination with the optimisation of the original part of the grammar. To keep the additional effort required of the user small, we consider relationships that could be identified by a very straightforward review of the SUT. These could be obvious relationships from functional requirements (e.g., for FCT3, the delivery of stable measures obviously implies a comparison of the current and previous values) or from explicit conditions in the source code (e.g., the comparison of current and previous values is explicit in a branch condition). In the latter case, the extraction of conditions could even be automated using a lightweight static analysis of the code.

2) Constraints on Production Weights: This second proposed solution is not directly related to the relationships between input variables discussed above. Instead it is motivated by the assumption that a profile will more diverse if all production rules in the grammar have a relatively high weight. In the current search algorithm, there is nothing to prevent some production rules taking very low weights, and we conjecture that the ease with which the search can synthesise a grammar in which some rules are only very rarely applied is, in part, responsible for types of degeneracy such as that observed for the TCAS threshold value in section IV.

We therefore propose a constraint that, for a variable with N production rules, the search may not reduce the weight of any rule below $\frac{r}{N}$, where $0 \leq r < 1$ is a parameter to the algorithm[2].

VI. Demonstration of Proposed Solution

In this section, we show that the proposed solutions—grammars augmented by new operators, and constraints on production rule weights—can improve the fault-revealing power of profiles synthesised for our two case study SUTs.

The augmentation of the original grammars of figure 1 is shown in figure 3. For FCT3, we add a single rule for equality of X1 and X4. We claim that this relationship would be identified by any straightforward review of either the requirements or the code of FCT3. Other relationships could be considered—and exist in the analytical profiles—but we refrain from using our knowledge of these profiles. For TCAS, we felt free to explore a higher number of relationships as we had not previously analysed optimal profiles for this SUT. From a quick review of the code, we extracted all explicit conditions of the form X `rel` E, where X is an input argument, `rel` is any comparison operator, and E is an expression involving only constants and/or input arguments. Whatever the comparison operator, we then systematically introduce a decomposition into three cases: '`==E`', '`<E`' and '`>E`'. For example, in figure 3b, the first rule accounts for the comparison of input argument CurVerSep to constants MINSEP and MAXALTDIFF; the last rule accounts for the comparisons of input argument DownSep to the selected threshold (constant array Positive_RA_Alt_Thresh[] indexed by input AltLayVal) as well as to argument UpSep.

The constraint on the minimum value of production weights is set to $\frac{3}{16N}$ where N is the number of productions

[2]As a result of how the search algorithm is implemented, weights are already constrained in the current search algorithm to a minimum of $\frac{1}{256N}$, but we propose here to use a much higher values of r.

```
X1Aug → 'none' | '==X4'
```

(a) FCT3

```
CurVerSepAug → 'none' | '<MINSEP' | '==MINSEP'
             | '>MINSEP' | '<MAXALTDIFF'
             | '==MAXALTDIFF' | '>MAXALTDIFF'
OwnTrkAltAug → 'none' | '<OthTrkAlt'
             | '==OthTrkAlt' | '>OthTrkAlt'
OwnAltRatAug → 'none' | '<OLEV' | '==OLEV' | '>OLEV'
    UpSepAug → 'none'
             | '<Positive_RA_Alt_Thresh[AltLayVal]'
             | '==Positive_RA_Alt_Thresh[AltLayVal]'
             | '>Positive_RA_Alt_Thresh[AltLayVal]'
  DownSepAug → 'none'
             | '<Positive_RA_Alt_Thresh[AltLayVal]'
             | '==Positive_RA_Alt_Thresh[AltLayVal]'
             | '>Positive_RA_Alt_Thresh[AltLayVal]'
             | '<UpSep' | '==UpSep' | '>UpSep'
```

(b) TCAS

Fig. 3: The production rules that augment the grammars of the case study SUTs. These rules are added to grammars shown in figure 1, and the new variables added to the production rule for starting variable `S`.

for the grammar variable. This value was found by preliminary experimentation: higher values constrained the search too greatly and prevented the synthesis of profiles that satisfied the adequacy criterion.

A. Method

We repeat the method of section IV-C for the following profiles:

1) Profiles synthesised by search using the original grammars of figure 1, but with a constraint on production weights.

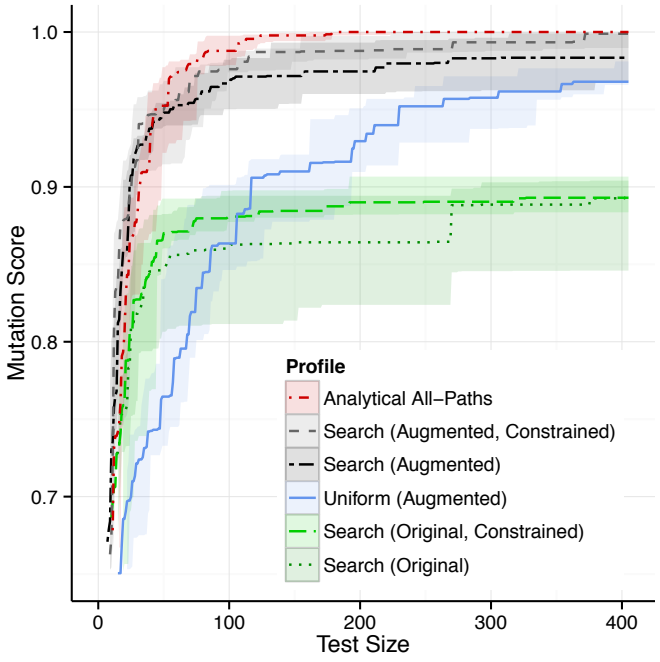
2) Profiles synthesised by search using the augmented grammars.

3) Profiles synthesised by search using the augmented grammar in combination with the constraint on production rule weights. The above three steps therefore evaluate the impact of the two proposed solutions separately and in conjunction.

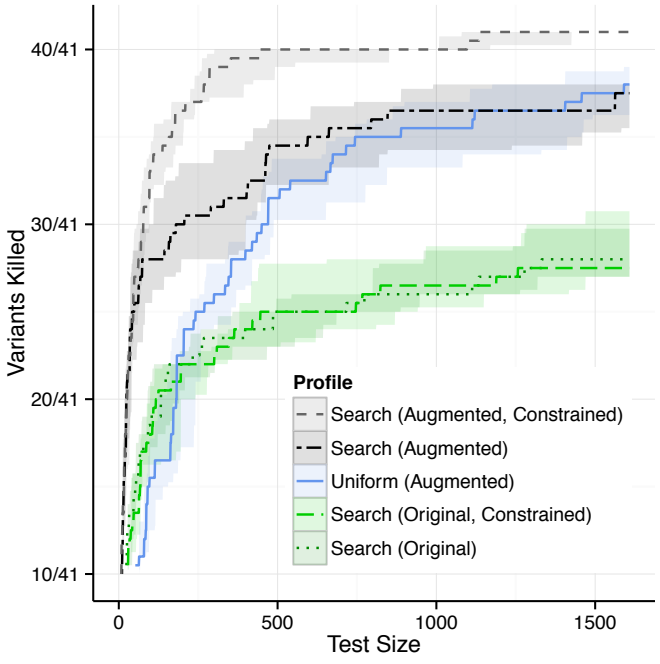
4) A uniform augmented profile unmodified by search. This is equivalent of the uniform profile used section IV-C, but using the augmented grammars of figure 3 with all production rules of a variable having the same weights. This profile provides a check whether any change in revealing power is only due to the user-supplied information, or there is added value in applying search to the augmented grammar.

B. Results and Discussion

The results are summarised in figure 4 which shows the median mutation scores at different test sizes for each type of profile. The original search-synthesised profiles from figure 2, and—for FCT3—the analytical all-paths profile, are included for comparison. For clarity, we omit the original uniform profile and modify the starting point on the y-axis to focus on the higher mutation scores.



(a) FCT3



(b) TCAS

Fig. 4: The mutation score (FCT3) and number of variants killed (TCAS) plotted against the size of test set. The lines indicate the median score, and the shaded ribbon the interquartile range.

TABLE I: Median mutation scores at selected test sizes for FCT3. The labels O, C, and A stand for Original, Constrained and Augmented Search respectively; Uni. A is the Uniform Augmented profile; All-P is the Analytical All-Paths profile.) The rows labelled Sig. are p-values assessing the statistical significance; bold values are significant at the 5% level.

	Test Set Size			
Profile	50	100	200	405
Search O	0.847	0.860	0.864	0.894
Search O,C	0.870	0.880	0.890	0.893
Sig. O vs. O,C	36.4%	18.6%	15.1%	57.0%
Uniform A	0.765	0.863	0.930	0.968
Search A	0.948	0.967	0.975	0.983
Search A,C	0.950	0.976	0.988	0.999
Sig. A vs. A,C	97.0%	27.3%	8.1%	7.1%
Analytic. All-Paths	0.952	0.988	1.000	1.000
Sig. A,C vs. O	**0.1%**	**0.0%**	**0.0%**	**0.0%**
Sig. A,C vs. Uni. A	**0.0%**	**0.0%**	**0.0%**	**0.1%**
Sig. A,C vs. All-P	91.2%	19.8%	**2.4%**	**0.6%**

TABLE II: Median number of variants killed, from a total of 41, at selected test sizes for TCAS.

	Test Set Size			
Profile	100	500	1000	1608
Search O	19.0	25.0	26.0	28.0
Search O,C	18.0	25.0	26.5	27.5
Sig. O vs. O,C	78.9%	39.3%	96.9%	78.8%
Uniform A	15.5	31.5	35.5	38.0
Search A	28.0	34.5	36.5	37.5
Search A,C	33.5	40.0	40.0	41.0
Sig. A vs. A,C	**1.8%**	**0.3%**	**0.1%**	**0.0%**
Sig. A,C vs. O	**0.0%**	**0.0%**	**0.0%**	**0.0%**
Sig. A,C vs. Uni. A	**0.0%**	**0.1%**	**0.1%**	**0.0%**

In addition, we show the median mutation scores at selected test sizes in tables I and tables II. In these tables we also indicate whether the differences in median scores between profiles are significant at the 5% level, calculated using the Wilcoxon-Mann-Whitney test to compare the ten samples of each type of profile.

1) Augmented vs. Original Grammars: For both SUTs, the profiles synthesised from the augmented grammars significantly outperform the profiles from the original grammars, at all test sizes. Even the uniform augmented profile significantly outperforms the original search-synthesised ones at large sizes. This latter result is again evidence of the hypothesised lack of diversity in original grammars and search algorithm. The original search-synthesised profiles are optimised to quickly attain structural coverage of the SUT, and hence catch more faults at small test sizes. However, as the test size increases, there might be insufficient data diversity to reveal the most hard-to-find faults, the revealing conditions of which are loosely connected to coverage conditions. On the contrary, the uniform augmented profile needs larger test sizes to cover the structure of the SUT, but has inherent diversity allowing it to reveal more faults at these sizes.

Thus we may conclude, key relationships identified by a very straightforward review of the SUTs are sufficient to dramatically increase the efficacy of the test profiles, whether one compares search-synthesised augmented profiles, or even the uniform augmented profiles, to the original profiles.

2) Search-Synthesised Augmented vs. Uniform Augmented Profiles: For both SUTs, the profiles synthesised from the augmented grammar significantly outperform the uniform profile from the same grammar. They do so at all test sizes if one considers the search-synthesised profiles *with* constrained weights. The synthesised profiles *without* constrained weights are approached by the uniform augmented profiles, but only at large test sizes.

There is thus an added value in the optimisation performed by the search. The search-synthesised profiles reveal the faults significantly more quickly and do not lose their power as the test size increases. This may be an indication that the search has retained diversity while optimising for coverage.

3) Constrained Weights: Adding constraints on production rule weights does not make a statistical difference, whether using the original or augmented grammars. The exception is for the TCAS augmented grammar for which profiles synthesised using weight constraints are significantly better than those synthesised without. Constraining weights is thus less effective than the augmentation of the grammars, but—at least for these two SUTs—it is not detrimental, and for one SUT improves the fault revealing-power. Moreover, if one considers the search augmented, constrained profiles in tables I and II, the combination of the two proposed solutions enables perfect median mutation scores of 1.0 to be reached for both SUTs.

4) FCT3 All-Paths: For FCT3, we had the opportunity to compare the search-synthesised profiles to analytically derived ones. We focus here on the comparison of the search augmented, constrained profiles to the analytical all-paths profile. The results are statistically different only at the highest test sizes, and at these sizes the actual difference in scores, i.e. the effect size, is small: at size 405, 7 search-synthesised sets obtain the same score of 1.0 as the all-paths ones, 2 sets have a score of 0.999, and the remaining one has 0.994. These results show profiles synthesised by automated search can be competitive with analytical profiles.

We also note that the specific types of degeneracy observed in section IV no longer occur to the same degree. For FCT3, the branch requiring equality of X1 and X4 is now exercised by between 8 and 27 different values in each of the ten sets generated from the augmented and constrained grammar. For TCAS, it is no longer the case that comparisons focus on a single threshold value of vertical separation.

VII. Related Work

Since the original work by Thévenod-Fosse and Waeselynck, alternative implementations of statistical testing have been studied. Gouraud et al. proposed a method that first selects a path in the control-flow graph using a uniform generation of combinatorial structures, and then feeds the path predicates to a randomised constraint solver to produce diverse data [10], [11]. Petit and Gotlieb also investigated path-based statistical testing using a probabilistic extension of constraint programming [12]. In contrast, search-based statistical testing, as investigated by Poulding et al. [6], [7], [9], does not require the derivation and solution of path constraints, and thus

provides a more flexible method of satisfying *any* adequacy criterion—whether path-based or not—provided that structural or functional coverage measures can be used by the search algorithm for fitness evaluation.

A number of authors utilise stochastic grammars to generate test data; for example, Maurer [13] uses a grammar to test VLSI circuits in simulation. Most such techniques adjust the production weights manually to meet testing objectives; the algorithm of Beyene and Andrews [14], used to generate HTML and XML inputs, is one of the few that optimises the weights automatically. The algorithm of Poulding et al. used in this paper also optimises the weights, but additionally modifies the production rules themselves.

The augmented grammars in this paper were motivated by the need to avoid degenerate profiles, but their use may also facilitate scalability: it seems reasonable to expect that complex cases cannot be handled unless some contextual knowledge is injected into the search process. In this sense, the augmented grammar approach is consistent with that adopted in the early work on statistical testing: for cases that could not be solved analytically, the user had to iteratively try some interesting generation patterns and tune their probabilities [2]. Here, the grammar-based framework provides convenient support to suggest the patterns at a high level of abstraction, and the tuning then proceeds automatically.

VIII. Conclusion

Using search to synthesise input profiles is a cost-effective method of implementing statistical testing. However, we hypothesised in this paper that the fault-revealing power of the synthesised profiles may be poorer than those created by analytical techniques. Our reasoning was the standard grammar representation is unable to concisely express the relationships between input arguments that are important to satisfying the adequacy criterion, and so the search produced degenerate profiles that demonstrated these relationships only over a small part of the input domain. Two case studies provided evidence in support of this hypothesis. They exemplified challenging cases for relationships, and both demonstrated a lack of diversity in generated input data together with a fault-revealing power that was less than expected.

We proposed two enhancements to the search algorithm in order to prevent such degeneracy. Firstly, the use of new grammar operators that express relationships between input arguments that a user might identify based on a straightforward analysis of the SUT. Secondly, a constraint on the minimum weight that can be assigned to production rules during the search. Applying these enhancements to the case studies confirm that both enhancements, but particularly the SUT-specific knowledge encapsulated in the new operators chosen by the user, can significantly improve the fault-revealing power of the generated test sets. As a result, the potential time and cost advantages of search-based statistical testing may now be better realised in practice.

As future work, we plan to review additional software in order to identify the most common relationships between input arguments that should be represented in the grammar. So far, our work has addressed numerical inputs, but the grammar extension need not be limited to them. Other relationships could be expressed, such as regular expression matches for strings or membership/inclusion relations for collections of items. Our intention is to improve usability by extending the grammar notation to facilitate these most often-used relationships without the need for post-processing the terminals sampled from the grammar. Facilities for post processing will still be retained to accommodate the less common, usually SUT-specific, relationships between input arguments, hence providing a very flexible framework.

Acknowledgments

This work is funded in part by EPSRC grant EP/J017515/1, DAASE: Dynamic Adaptive Automated Software Engineering, and in part by RTRA STAE, the French Space and Aeronautic Sciences & Technologies foundation (IFSE project on formal system engineering). The experiments used the SESAME mutation environment [15] from Yves Crouzet at LAAS-CNRS.

References

[1] P. Thévenod-Fosse, H. Waeselynck, and Y. Crouzet, "An experimental study on software structural testing: deterministic versus random input generation," in *Twenty-First Int'l Symposium Fault-Tolerant Computing (FTCS-21), Digest of Papers*, 1991, pp. 410–417.

[2] P. Thévenod-Fosse and H. Waeselynck, "Statemate applied to statistical testing," in *Proc. Int'l Symp. Software Testing and Analysis (ISSTA)*, 1993, pp. 99–109.

[3] P. Thévenod-Fosse, H. Waeselynck, and Y. Crouzet, "Software statistical testing," in *Predictably Dependable Computing Systems*, B. Randell, J.-C. Laprie, H. Kopetz, and B. Littlewood, Eds. Oxford: Springer Verlag, ESPRIT Basic research Series, 1995, pp. 253–272.

[4] J. D. Musa, "Operational profiles in software-reliability engineering," *IEEE Softw.*, vol. 10, no. 2, pp. 14–32, Mar. 1993.

[5] J. Whittaker and M. Thomason, "A Markov chain model for statistical software testing," *IEEE Trans. Software Eng.*, vol. 20, no. 10, pp. 812–824, 1994.

[6] S. Poulding and J. A. Clark, "Efficient software verification: Statistical testing using automated search," *IEEE Trans. Software Eng.*, vol. 36, no. 6, pp. 763–777, 2010.

[7] S. Poulding, R. Alexander, J. A. Clark, and M. J. Hadley, "The optimisation of stochastic grammars to enable cost-effective probabilistic structural testing," in *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, 2013, pp. 1477–1484.

[8] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empir. Software Eng.*, vol. 10, no. 4, pp. 405–435, 2005.

[9] S. Poulding, J. A. Clark, and H. Waeselynck, "A principled evaluation of the effect of directed mutation on search-based statistical testing," in *Proc. 4th Int'l Workshop on Search-Based Software Testing (SBST 2011)*, 2011, pp. 184–193.

[10] S.-D. Gouraud, A. Denise, M.-C. Gaudel, and B. Marre, "A new way of automating statistical testing methods," in *Proc. IEEE Int'l Conf. on Automated Software Eng.*, 2001.

[11] A. Denise, M.-C. Gaudel, and S.-D. Gouraud, "A generic method for statistical testing," in *Proc. 15th Int'l Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2004, pp. 25–34.

[12] M. Petit and A. Gotlieb, "Uniform selection of feasible paths as a stochastic constraint problem," in *Proc. Seventh Int'l Conference on Quality Software (QSIC)*. IEEE, 2007, pp. 280–285.

[13] P. Maurer, "Generating test data with enhanced context-free grammars," *IEEE Softw.*, vol. 7, no. 4, pp. 50–55, July 1990.

[14] M. Beyene and J. Andrews, "Generating string test data for code coverage," in *Proc. IEEE Int'l Conf.. on Software Testing, Verification and Validation (ICST)*. IEEE, 2012, pp. 270–279.

[15] Y. Crouzet, H. Waeselynck, B. Lussier, and D. Powell, "The SESAME experience: from assembly languages to declarative models," in *Second Workshop on Mutation Analysis*, 2006.