A Principled Evaluation of the Effect of Directed Mutation on Search-Based Statistical Testing

Simon Poulding
Department of Computer Science
University of York
York YO10 5GH, United Kingdom
Email: smp@cs.york.ac.uk

John A. Clark
Department of Computer Science
University of York
York YO10 5GH, United Kingdom
Email: jac@cs.york.ac.uk

Hélène Waeselynck
LAAS-CNRS
7, avenue du Colonel Roche
31077 Toulouse Cedex 4, France
Email: waeselyn@laas.fr

***Abstract*—Statistical testing generates test inputs by sampling from a probability distribution that is carefully chosen so that the inputs exercise all parts of the software being tested. Sets of such inputs have been shown to detect more faults than test sets generated using traditional random and structural testing techniques. Search-based statistical testing employs a metaheuristic search algorithm to automate the otherwise labour-intensive process of deriving the probability distribution. This paper proposes an enhancement to this search algorithm: information obtained during fitness evaluation is used to direct the mutation operator to those parts of the representation where changes may be most beneficial. A principled empirical evaluation demonstrates that this enhancement leads to a significant improvement in algorithm performance, and so increases both the cost-effectiveness and scalability of search-based statistical testing. As part of the empirical approach, we demonstrate the use of response surface methodology as an effective and objective method of tuning algorithm parameters, and suggest innovative refinements to this methodology.**

***Index Terms*—Software/Program Verification, Testing Strategies, Test Coverage of Code, Optimization, Experimental Design**

I. Introduction

Statistical testing is a highly effective technique for generating test inputs for software verification. Test data is sampled from a probability distribution over the input domain—in a similar manner to random testing—where the probability distribution is carefully chosen so that every part of the software is covered by the test set—in a similar manner to structural testing. Thévenod-Fosse and Waeselynck have shown that statistical testing has superior fault-detecting ability compared to both random and structural testing [1], [2], [3], [4], [5].

In [6], Poulding and Clark proposed and evaluated a search-based approach to deriving probability distributions for statistical testing in order to automate this part of the technique and therefore improve its cost-effectiveness. The viability of this approach, called *search-based statistical testing*, was demonstrated even though the search algorithm used a relatively straightforward representation, fitness metric, and search method. Subsequent work has focussed on improving the performance through enhancements to the search algorithm, with the objective of enabling the approach to be used on more complicated software and to satisfy more ambitious testing objectives.

In this paper, we present one such enhancement to the search algorithm: the use of *directed mutation*. When evaluating potential solutions, feedback is normally returned to the search algorithm in the form a fitness metric, and this information is used to guide the algorithm's selection operator. In our implementation of directed mutation, a second feedback pathway is used to direct (or bias) the mutation operator to parts of the representation where changes are most likely to result in fitter solutions.

The contributions of this work are:

- An assessment of the effect of directed mutation on search-based statistical testing. We show that the performance of algorithm is significantly improved, by a factor of over 5 times in some cases.
- A demonstration of *response surface methodology* (RSM) as an approach to objectively tuning the algorithm parameters in order to enable a principled evaluation of the effect of directed mutation. We show that it is possible to successfully apply RSM to a large set of algorithm parameters, and also suggest a number of innovative refinements to the methodology.
- The application of search-based statistical testing to a wider range of software, including software used in Thévenod-Fosse and Waeselynck's evaluations of statistical testing.

The paper is structured as follows. In section II, we review related work on statistical testing, and provide an overview of both the search-based statistical testing algorithm and response surface methodology. The proposed algorithm enhancement using directed mutation is described in section III. Section IV defines the two research objectives for the empirical evaluation. In section V, we describe the methods employed in the empirical evaluation, including the refinements we propose to the standard RSM process. Results are presented and analysed in section VI. In the conclusion, we consider the results in the context of the research objectives, and discuss future work.

II. Related Work and Background

A. Statistical Testing

Statistical testing might be considered to be a combination of the most advantageous features of both random and

structural testing. As for random testing, statistical testing generates test inputs stochastically according to a probability distribution over the inputs of the software. A significant advantage is that it is straightforward, and therefore cheap, to create test inputs in this way. However, random testing typically uses a probability distribution that is either uniform—all inputs have the same chance of being chosen—or based on the operational profile of the system. Using either of these distributions can mean that since some parts of the software are very rarely executed by the test set, some faults may not be detected. To avoid this disadvantage, statistical testing uses more sophisticated input probability distributions that ensure every part of the code is executed 'reasonably often'. The specification of 'reasonably often' is expressed as an *adequacy criterion* similar to those employed by structural testing, defining both a set of structural elements (e.g., paths, statements or branches) and a constraint on how frequently these elements are executed. For example, the test engineer may specify that every branch must have a probability of at least 0.1 of being executed by a single test input. We will refer to distributions satisfying the adequacy criterion as *adequate distributions*, and the constraint on the frequency of executing a branch (0.1 in the previous example) as the *probability lower bound*.

Given an adequate distribution, the test engineer can calculate the test set size that ensures that (with high likelihood) every branch is executed at least once by the test set: an example of this calculation is given in [7]. Thus, it is beneficial to specify a high value for the probability lower bound in order to ensure that the test size is small and therefore that the testing is cheap to perform. Alternatively, given the imperfect relationship between software coverage and fault-detecting ability, the test engineer may choose to use a larger test set that executes each branch *multiple* times rather than only once, with the objective of detecting more faults. Using statistical testing, there is little additional effort in sampling more test inputs for these larger test sets once an adequate distribution has been derived[1]. In contrast, many structural test data generation techniques would require substantial additional effort to generate the larger number of distinct test inputs required.

Thévenod-Fosse and Waeselynck investigated the efficacy of the approach using adequacy criteria based on the software's structure [2], [3], [4], [5]; and based on a functional specification expressed in terms of an executable model [3]. The results of these investigations demonstrated that statistical testing detects more faults than random testing, and that test sets generated using statistical testing that executed each structural element multiple times detected more faults than minimal tests sets generated by traditional structural testing.

In some cases, adequate distributions could be derived using static analysis of the software. However, for larger software and to meet stronger adequacy criteria, it was necessary to derive a distribution using a manual approach: the software was run using a sample of test inputs derived from a candidate distribution and the distribution was modified based on the coverage elements executed. While effective, this approach is costly and is impractical for large software. It is this limitation that motivated the use of automated search to derive adequate distributions.

[1]There is, of course, additional effort in deriving predicted test *outputs* for the test cases and running the test cases themselves, but this is the same regardless of the method of deriving the test inputs.

B. Search-Based Statistical Testing

In [6], Poulding and Clark demonstrated that a relatively straightforward search algorithm was a viable approach for deriving adequate probability distributions for statistical testing. Since the search algorithm proposed in this paper is an enhancement to that algorithm, we summarise the key features as follows:

1) Representation: For the *software under test* (SUTs) considered in this work, the test inputs are the arguments to a function. In general, we cannot treat each argument independently since they will interact with each other in the code. Therefore a joint (multivariate) distribution is necessary, and a Bayesian network is used to represent such a distribution.

A Bayesian network is a directed acyclic graph, and in our representation each node corresponds to the probability distribution of one SUT argument. We partition the input domain of the argument into a number of contiguous bins and assign a probability to each bin. When we sample a value for the argument, a bin is picked according to the bin probabilities, and then the argument value is chosen from within the bin according to a uniform distribution.

For example, consider a SUT with three floating point arguments: a, b, and c. If the valid domain of a is $[3.5, 8.0)$, this might be split into three bins: $[3.5, 5.921)$, $[5.921, 7.508)$, and $[7.508, 8.0)$, with probabilities 0.055, 0.7014, and 0.244 respectively. When a value is sampled for a, one of these three bins, say $[7.508, 8.0)$, is chosen at random according to the probabilities, and then the input for a is chosen according to a uniform distribution over the interval $[7.508, 8.0)$.

Interactions between arguments are expressed as directed edges between nodes in the Bayesian network: such edges indicate that the probability distribution of the child argument is conditionally dependent on the values taken by the parent argument(s). In our representation, this conditionality is defined in terms of bins rather than the actual values. For example, a directed edge from a to c would indicate that argument c is conditionally dependent on the argument a, and so we would store three different sets of probabilities for the bins of c: one for each of the three bins of a. When we sample from the network, we first choose a bin for a as discussed above, and then this choice determines which of the three sets of probabilities we use to select a bin for c.

2) Fitness Metric: The fitness metric we use is an estimate of the probability lower bound achieved by the candidate probability distribution. To obtain this estimate, we sample a number, K, of inputs from the distribution, and use them to run an instrumented version of the SUT that identifies the structural elements (such as branches) executed by each input.

If the set of structural elements is C, and for each $c \in C$, $e_{c;i}$ is an indicator variable set to 1 if c is executed one or more times by the i^{th} input, and set to 0 otherwise, then the fitness is calculated as:

$$f = \frac{1}{K} \min_{c \in C} \left(\sum_{i=1}^{K} e_{c;i} \right) \quad (1)$$

Since this estimate is obtained using a random sample of finite size from the probability distribution, it is unavoidably noisy: it provides only an approximation to the actual probability lower bound, and repeated evaluations of the same candidate probability distribution will return slightly different fitness values.

3) Search Method: The search method used is stochastic hill climbing. The initial candidate probability distribution is a uniform distribution over the input domain. At each subsequent iteration, a random sample of neighbours, size m, of the current candidate probability distribution is evaluated. These neighbours are each formed by a single mutation to the current distribution using one of the following mutation operators:

- M_{add} adding an edge between two nodes in the Bayesian network (so long as it does not become cyclic);
- M_{rem} removing an edge;
- M_{len} multiplying or dividing the length of a bin interval by a fixed value, ρ_{len};
- M_{spl} splitting a bin into two equally-sized bins;
- M_{joi} joining two adjacent bins;
- M_{prb} multiplying or dividing the probability of a bin by a fixed value, ρ_{prb}.

Any change to bin lengths or probabilities is followed by renormalisation so that the bin lengths sum to the length of the input domain, and the bin probabilities sum to 1.

Each of these mutation operators has an associated weight, w_{add} etc., that applies to every valid mutation of that type. For example, if there are 49 bins, then there are 49 different possible M_{spl} mutations possible, and each has a weight w_{spl}. The weight is the relative probability that this is the single mutation that is used to create a neighbour for the current candidate distribution.

If the most fit of the neighbours has a better fitness than the current candidate probability distribution, then that neighbour becomes the current candidate. The search is terminated when the fitness of a candidate distribution reaches the target value.

C. Response Surface Methodology

Search performance is often highly dependent on the settings of the algorithm's parameters. At some parameter settings our proposed directed mutation enhancement might perform better than the original algorithm, while at others it may perform worse. To enable a principled comparison of the two algorithms—with and without directed mutation—we must decide how to set the parameters, and we choose here to compare the algorithms with their parameters set to ensure near-optimal performance. This is consistent with how the algorithms would be used in practice: either automatically, or set by the test engineer, the parameters of the search algorithm would adjusted to suit the SUT. One long-term objective of our research is to understand how effective parameter settings can be derived automatically from the characteristics of the SUT, but in the absence of this understanding, we must tune the algorithm parameters empirically for the purpose of our evaluation here.

A number of sophisticated parameter tuning approaches exist. Since the tuning of parameters is itself an optimisation problem, one approach is to apply evolutionary search algorithms to the parameters, resulting in *meta-evolutionary algorithms*. For example, Grefenstette uses one genetic algorithm to tune the control parameters of another [8]. Other approaches apply design of experiments (DoE) techniques to derive an approximate model of how the algorithm performance depends on parameter settings, and predict optimal parameters using this model. This technique has been applied to the parameters of ant colony systems [9], and genetic programming [10]. Tsai, Chou and Liu describe an approach that combines a genetic algorithm with the Taguchi method, a technique that shares some of the characteristics of DoE [11].

We choose to tune the algorithm parameters using *response surface methodology* (RSM), an approach based on DoE techniques. RSM is used to efficiently optimise industrial processes [12], but has only occasionally been applied to software; one of the few examples is the optimisation of wireless protocol parameters [13]. Our decision to use RSM was based on a belief that would require few subjective decisions and so would contribute to a fair comparison of the algorithms.

In this section, we do not attempt to fully describe the details of RSM as there are a number of excellent texts on the subject, such as Myers [12] and Montgomery [14]. Instead, we provide an overview of the standard RSM approach and later, in section V-C, discuss some innovative refinements that we applied for the empirical work in this paper.

If we consider the settings of the algorithm parameters as defining the coordinates on a surface whose height represents the response (in our case, the performance) of the algorithm, we can envisage a *response surface* that is analogous to a fitness landscape. (However the response surface, here relating to algorithm performance, should not be confused with the fitness landscape of candidate probability distributions that is discussed in section III.) RSM assumes that the optimum point on this surface can be reached by hill-climbing and uses traditional DoE techniques—factorial designs and regression analysis—to locate it.

In RSM the coordinates of the response surface are typically *coded* values that normally lie in the range $[-1, 1]$: this is for convenience and to avoid inaccuracies than can arise in the analysis techniques when parameters differ greatly in magnitude. Therefore the actual parameters, which we denote x_i, are linearly mapped to coded parameters, x_i^*, so that -1 and 1 represent the extremes of a 'reasonable' range for the parameter. Although deciding what constitutes 'reasonable' ranges for parameters is somewhat arbitrary, it does *not* place

constraints on the parameters: the coded parameters can take values outside the interval $[-1, 1]$.

A starting point is chosen for the hill climb on the response surface, such as the origin (coded parameter values all zero). In the first RSM phase, a set of points is chosen in a small region around this starting point; often the points are chosen according to factorial or central composite designs. The algorithm is run at the points defined by the design, and the response, y, of the algorithm is measured. The responses are used to fit a first-order linear of the following form using regression analysis:

$$Y = \beta_0 + \sum_{i=1} \beta_i x_i^* + \epsilon \tag{2}$$

where Y is a random variable representing the response, x_i^* are the coded algorithm parameters, β_0 and the β_i are model parameters estimated by regression analysis, and ϵ is a 'noise' or 'error' term that accounts for the variance in the algorithm response. (In the case of the algorithms considered in this paper, the variance is entirely due to their stochastic nature: i.e., running the algorithm with the same parameters, but different seeds to the pseudo-random number generator, will lead to different responses.)

Since the region sampled around the starting point is small, a linear model gives a reasonable approximation of the surface at this point, and the vector $\vec{d} = (\beta_1, \beta_2, \ldots)$ gives an estimate of the direction of the steepest slope. This direction should point towards the top of the hill, i.e., the optimum parameter settings.

In the second phase, a path in the direction of this vector is followed, and at regular intervals along the path, experiments are run to measure algorithm response until no further improvement is detected.

The slope vector is only an estimate, and the further away from our starting point, the less accurate it will be at indicating the direction of the optimum. Therefore, we are unlikely to have reached the optimum on the surface, but, conceptually, we will be on a ridge to one side of the summit of the hill. For this reason, we treat the point at which no further improvement is possible along the path as a new starting point, and repeat phases 1 and 2. This iterative process is repeated a predefined number of times, or until regression analysis suggests the linear model is a poor fit owing to curvature in the surface which is indicative of reaching the summit of the hill. At this point, a second-order model may be fitted in order to accurately estimate the optimum point on the surface.

III. Directed Mutation

Using the existing algorithm used by search-based statistical testing, we observed that—especially at the start of the search—some structural elements may only very rarely be executed by candidate distributions, and so the fitness metrics are very close, or equal, to zero. Since the fitness metric is noisy, even if a candidate distributions does, by chance, execute one of these rare structural elements, it may not on subsequent evaluations. Thus the fitness landscape is essentially 'flat' in this region and provides very little guidance to the search algorithm. We speculate that the algorithm can spend many iterations performing a random walk on this flat region before locating a region that has a gradient which can guide the search towards a solution.

Our proposed solution is to more effectively utilise the 'chance' executions of the most rarely executed structural elements through the use of *directed mutation*. During the fitness evaluation, we keep track of the bins in our Bayesian network representation that gave rise to inputs that executed the least executed element(s), and we use this information to increase the probability of mutations to these bins (and correspondingly decrease the probability of mutating other bins). The effect is that sampling the neighbourhood around the current candidate distribution is now biased towards neighbours that may be expect to change (ideally, but not necessarily, increase) the frequency of executing the least-exercised structural element.

A similar concept of directed (or targeted) mutation operators has been applied, for example, to the construction of timetables using memetic algorithms [15], [16]. In these studies, the mutation operator is directed towards those parts of the representation that have the potential to resolve violations of timetabling constraints in the current individual. The infected genes evolutionary algorithm (igEA) of Tavares et al. is a generalisation of this approach: information from phenotypic evaluation is used to 'protect' desirable genes in the representation from variation by mutation and crossover [17].

Directed mutation is distinct from *adaptive mutation*. In most forms of adaptive mutation (such as those discussed in [18]), the *overall* mutation rate is modified—over time, based on the recent changes in fitness, or by self-adaption as part of the representation—but the mutation operator itself is unchanged. In the *directed mutation* proposed here, the overall mutation rate is unchanged; instead the mutation operator is dynamically modified in order to bias mutations towards particular parts of the representation.

To implement directed mutation, we create three 'mutation groups' formed using the mutation operators described in section II-B3:

- G_{edge} consists of the operators that modify edges in the Bayesian network: M_{add} and M_{rem};
- G_{bins} consists of the remaining operators that modify bins directly: M_{len}, M_{spl}, M_{joi}, and M_{prb}, and applies to *all* bins;
- G_{drct} consists of the same bin-modifying operators as G_{bins}, but applies them *only* to bins that contributed input vectors that executed the least-exercised structural element.

It is the last group, G_{drct}, that implements directed mutation: the rate at which bins related to the least-exercised structural element are mutated is enhanced compared to other bins.

Each of the groups also has an associated weight, w_{edge} etc. When randomly choosing a mutation, the first step is to select one of the mutation groups at random according to the group weights. A mutation is then selected from

within that group using the process described in section II-B3. Without this group weighting mechanism, we speculate that for representations with many bins, the number of directed mutations will be very few compared to the number of regular mutations and any benefit from directed mutation would be lost. The initial selection by mutation group weight, regardless of the number of individual mutation operations within the group, ensures that direct mutation occurs at a rate that is independent of the number of bins.

IV. Research Objectives

A. Evaluation of Directed Mutation

Our first objective is to evaluate whether directed mutation, as described in section III, enables improved algorithm performance. Since the execution of the instrumented SUT accounts for most of the algorithm run time, we test the following hypothesis: the use of directed mutation results in an algorithm that, on average, requires fewer executions of the instrumented SUT in order to derive an adequate probability distribution.

We will assess this hypothesis by empirically comparing the performance of the search algorithms without the directed mutation group, G_{drct}, to the performance of the algorithm with this mutation group, on a range of SUTs. We will denote these algorithms as DM- and DM+ respectively. (DM- is broadly equivalent, though not identical, to the algorithm used for the empirical work in [6], and which is summarised in section II-B3.)

B. Evaluation of RSM

Our second objective is to evaluate response surface methodology as a mechanism for tuning algorithm parameters. We set no formal hypothesis for this objective, and instead we will reflect on our experience of using RSM in terms of its effectiveness, efficiency, and objectivity.

V. Empirical Method

A. Software Under Test

We compare algorithm performance on four very different C functions[2] for which finding adequate probability distributions is non-trivial. Two of the SUTs, bestMove and nsichneu, are real-world software used in the empirical work of [6]. To ensure a greater variety of SUT characteristics, two additional real-world SUTs, cArcsin and FCT3, are considered.

bestMove: This function determines the next move for a player in a tic-tac-toe (noughts-and-crosses) game when given the current board position. It is adapted from a demo applet on Sun's Java website [19]. The execution of certain structural elements occurs only when, for example, a valid winning position is passed to the function. Such winning positions are relatively rare and are not located near one another in the input domain, and so we suspect an adequate probability distribution is particularly difficult to derive.

[2]The source code of three of these SUTs is available from: www-users.cs.york.ac.uk/smp/supplemental

TABLE I
SUT Characteristics

SUT	bestMove	nsichneu	cArcsin	FCT3
lines of code	89	1967	70	135
input arguments	2	11	3	8
domain cardinality	2.62×10^5	3.40×10^6	∞	6.71×10^7
loops	7	1	0	0
structural elements	42	490	18	19
target fitness, f_{target}	0.112	0.028	0.12	0.05

nsichneu: This function consists of automatically generated code that implements a Petri net. It is part of a widely-used suite of benchmark programs used in worst-case execution time research [20]. We suspect that the large number of conditional statements and the data-flow between them makes this SUT difficult for search-based statistical testing.

cArcsin: This function returns the arcsin of a complex number. This function is adapted from the GNU Scientific Library [21]. The arguments to the function are the real and imaginary parts of the complex number, and are passed as double-precision floating point numbers. Even though we restrict the domain of the these arguments to the semi-open interval $[-10, 10)$, the cardinality of the domain is effectively infinite. The function behaves very differently when the imaginary argument is zero, and since the chance of this occurring during random sampling from the floating point interval $[-10, 10)$ is effectively zero, we introduce a third Boolean-valued argument for testing purposes that sets the imaginary argument to zero.

FCT3: This function was used by Thévenod-Fosse, Waeselynck and Crouzet in their evaluation of the fault-detecting ability of statistical testing compared to traditional random and structural testing approaches [7]. It is used as part of a nuclear reactor safety shutdown system.

Characteristics of these SUTs are shown in table I. In all four cases, we specify the adequacy criterion in terms of branches, so the number of structural elements is the number of conditional branches in the code. The table also shows the target fitness, i.e., the minimum chance of executing a structural element with an input vector sampled at random from the input distribution. These fitnesses are 80% of the target values used in [6], which themselves are slightly less than the maximum probability lower bounds that can be achieved for each SUT. This lower target is motivated by the observation made in [6] that very good fault-detecting ability can be achieved using distributions whose fitnesses are close to, but not at, the optimal value.

B. Algorithm Parameters

The parameters tuned using RSM are listed in table II, together with the initial (untuned) parameter value and the 'reasonable' range used to derive the coded parameter values (as described in section II-C). Parameter x_{10} applies only to the DM+ algorithm.

The meanings of these parameters are discussed in sections II-B3 and III. The mutation weight parameters are not

independent of one another since they indicate the *relative* probability of making these mutations. We accommodate this by fixing the weights w_{add}, w_{prb}, and w_{edge}, and expressing the other weight parameters in terms of these fixed values.

For many of the parameters, the natural logarithm of the parameter is the value that is tuned using RSM. Preliminary experimentation suggested that optimal values for these parameters might differ by orders of magnitudes across different SUTs and so using the logarithm of the parameter value proved not only to be convenient, but we suspect enabled optimal regions in the parameter space to be reached more quickly as linear changes to the log parameters represented multiplicative changes to the actual parameters. For clarity, the initial value and reasonable range shown in table II are the values prior to taking the logarithm. (It is the conversion back from rounded logarithmic values that result in the seemingly arbitrary values in the table.)

Given this large number of parameters, the first step of RSM is normally to 'screen' the parameters so that only those that have the greatest effect on the response are subsequently tuned. We attempted screening using analysis of variance (ANOVA) but found that the stochastic variance of the algorithm led to unreliable results. For this reason, we decided to forego screening, and apply RSM to all the parameters listed in table II.

There are a small number of additional algorithm parameters that we choose not to tune. These parameters place 'sensible' limits on the algorithm actions, such as maximum number of bins for an argument and the minimum length of a bin. Preliminary experimentation suggested that so long as these limits are not unnecessarily restrictive, these parameter values have little effect on algorithm performance, presumably because they are only applied in exceptional circumstances.

C. Response Surface Methodology

This section describes the specific implementation of RSM used for the empirical work, including a number of innovative refinements. Many of the specific choices we made, such as the number of algorithm runs at design points, the upper limit on the number of SUT executions, and the size of steps along the path of steepest slope, are based on experience gained from preliminary experimentation. The standard RSM process is described in section II-C.

Phase 1 - Estimating Steepest Slope: Our starting point for the RSM hill-climb was the origin in the coded parameter space (the actual parameter values at this point are shown in table II), and we used a 128 point fractional factorial design of at least resolution IV (meaning that the model coefficients of the first-order linear model are not confounded by any second or third-order interactions) at distances of ± 0.1 from this starting point. This gave the acceptable results given the relatively large stochastic variance in the algorithm performance. The algorithm was run once at each design point; we will describe each such run as a *trial*.

Our response metric was based on the number of executions of the instrumented SUT required by the algorithm to derive an adequate distribution. However, to prevent excessively long-running trials, we terminated the algorithm when a sensible upper limit on the number of executions was reached, and this limit was set for each SUT so that only a small proportion of trials were terminated in this way. So that we can continue make use of the information provided by these terminated trials, we define the following response metric:

$$y = \log(n_{\text{exec}} \times \frac{f_{\text{target}}}{f_{\text{best}}}) \tag{3}$$

where n_{exec} is the number of executions performed when the algorithm terminated, f_{target} is the target fitness for candidate distribution, and f_{best} is the best fitness achieved over the course of the algorithm run calculated according to (1). The effect of this metric is that if the algorithm completes and an adequate distribution was found (i.e., the target fitness was achieved), then the response, y, is based on only the number of executions since $f_{\text{target}}/f_{\text{best}} = 1$. However, if the algorithm terminated because the upper limit on the number of executions was reached, then the number of executions is increased by a proportion dependent on how close the algorithm was to achieving the target fitness.

We found that the stochastic variance in the algorithm performance was proportional to the performance itself. For this reason, we apply the natural logarithm in equation (3) when calculating the response. This transformation ensures that the variance in the response is approximately constant, a property that is required for the accuracy of some design of experiments analysis techniques, such as ANOVA. It also made the distribution of the response more symmetrical and so the mean and median of the distribution are more similar. This is beneficial as we (implicitly) use the mean response when applying regression analysis, but use the median response in our analysis elsewhere for robustness.

Having fitted a first-order linear model (of the form shown in equation (2)), we estimate the steepest slope direction as the vector:

$$\vec{d} = \frac{-1}{\sqrt{\sum_{i=1}^{10} {\beta_i}^2}}(\beta_1, \beta_2, \ldots, \beta_{10}) \tag{4}$$

where β_i are the fitted linear model coefficients. The numerator in the fraction is negative as we wish to *descend* to smaller responses that indicate better performance, and the denominator is a scaling factor so that $\vec{d}$ has length 1 in the coded parameter space.

Phase 2 - Following Steepest Descent Path: We follow the path of steepest descent defined by the direction $\vec{d}$, and measure the algorithm response at steps of $0.2\vec{d}$. At each step, we ran 32 trials and calculated the median response in order to reduce the effect of stochastic variance.

Innovation 1: In standard RSM, the path is followed until the response shows no further improvement. Usually a simple stopping rule is defined, such as two successive steps showing no improvement. However, it was found that the large variance in the algorithm response made such rules unreliable. Instead, we assumed that the section of the response surface along

TABLE II
ALGORITHM PARAMETERS TO BE TUNED

Parameter	Interpretation	Starting Point	Reasonable Range
$x_1 = \log(K)$	no. of input vectors sampled	1097 (cArcsin: 403)	493–2241 (cArcsin: 181–898)
$x_2 = m$	no. of neighbours evaluated	12	2–22
$x_3 = \log(w_{\text{bins}}/w_{\text{edge}})$	mutation group weight: bins	4.482	1.000–20.09
$x_4 = \log(\rho_{\text{prb}})$	bin probability change ratio	4.711	1.105–20.09
$x_5 = \log(\rho_{\text{len}})$	bin length change ratio	4.711	1.105–20.09
$x_6 = \log(w_{\text{rem}}/w_{\text{add}})$	mutation weight: remove edge	1.000	0.667–1.500
$x_7 = \log(w_{\text{spl}}/w_{\text{prb}})$	mutation weight: split bin	1.000	0.667–1.500
$x_8 = \log(w_{\text{joi}}/w_{\text{prb}})$	mutation weight: join bin	1.000	0.667–1.500
$x_9 = \log(w_{\text{len}}/w_{\text{prb}})$	mutation weight: bin length	1.000	0.667–1.500
$x_{10} = \log(w_{\text{drct}}/w_{\text{bins}})$	mutation group weight: directed	0.606	0.135–2.718

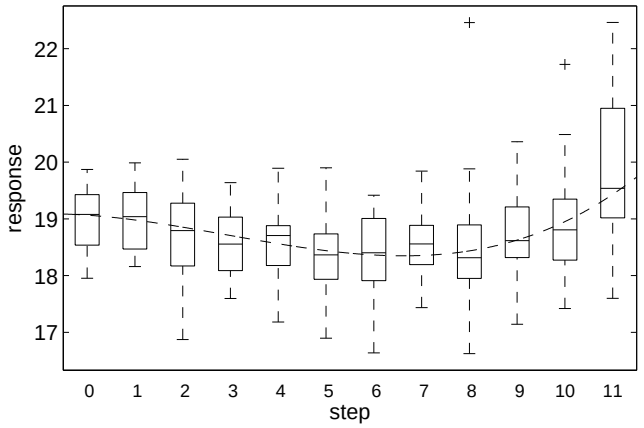

Fig. 1. Example of a cubic curve (dashed line) fitted to the median algorithm response at steps along the steepest descent path. At each step, the response of 32 trials is illustrated using a boxplot.

the path followed a smooth cubic curve. Having measured the response at the first 8 steps along the path, a cubic curve is fitted to the median response at each point. If the cubic indicated a minimum somewhere along the path travelled so far *and* a visual inspection suggested that the minimum was not unreliable because the fit of the curve to the median responses was poor, or because the minimum was close to the last point in the path, then no further steps on the path need be considered. Otherwise, further batches of 4 steps were measured, until the cubic fit demonstrated a minimum. By fitting a cubic across all points on the steepest descent path we avoid erroneous termination of the descent owing to algorithm responses that are, by chance, abnormally high at one or two points on the path. A further advantage is that the cubic curved can be used to estimate the minimum at a position *in between* the points sampled along the path: in standard RSM, the estimate of the minimum would be *at* rather than *between* the points.

An example of this innovation, taken from the application of RSM to tuning the algorithm DM- for the SUT bestMove, is shown in figure 1. This figure shows 12 steps: the starting point as step 0 and subsequent steps along the path of steepest descent. The response from the 32 algorithm trials run at each step are illustrated using a boxplot: the rectangular box is drawn between the first and third quartiles of the responses (and so the box height is the interquartile range), and the line across the box is the median response. (The 'whiskers' either side extend to the farthest data point up to 1.5 times the interquartile range from the box edges, and any 'outliers' beyond the whiskers are plotted as individual crosses.) The fitted cubic curve is shown as a dashed line. In this example, the curve predicts at minimum response at 6.6 step units from the starting point. More traditional stopping rules might have indicated stopping at step 3 (the next step shows an increase in the response), or step 5 (the next two steps show an increase in the response).

Innovation 2: A second innovation was in our handling of integer-valued parameters along the descent path. When the coded points along the path are converted to actual parameters, the value must be rounded if the parameter is an integer. When the parameter takes small integer values, this might lead to inconsistent steps along the path. For example, the sequence of real-numbered parameter steps $3.1, 4.3, 5.5, 6.7, 7.9$ may be rounded to $3, 4, 6, 7, 8$, with an inconsistent gap of 2 between the second and third steps, but 1 between the others. This is likely to create artefacts in the curve along the path, leading to inaccurate fitting of the cubic. In order to reduce this effect, when setting the parameter values for the sample of 32 algorithm trials at a step along the path, we *probabilitiscally* round up *or* down, with the probability of rounding up being given by the fractional part of unrounded parameter value. Thus, considered across the all 32 trials run at this point, the *average* value of the parameter is close to the intended real-numbered value.

Further Iterations: We choose to perform only two iterations in total of phase 1 (slope estimation) and phase 2 (path following). Since our objective is to find 'good' (near-optimal) parameters, rather than to find the absolute optimal parameters, there is no need for further iterations. We might have used only one iteration, but by using a second iteration we guard against a situation where, by chance, our initial point is in a relatively flat area of the response surface and so the path of steepest descent may show little improvement. In standard RSM, a

final phase of regression analysis is used to fit a second (or higher) order linear model to the response surface in order to accurately estimate the global optimum, but we omit this phase for the same reason.

D. Algorithm Comparison

1) Evaluation of Directed Mutation: In order to test the hypothesis, the performance of the algorithms DM- and DM+ is compared by running a sample of 64 algorithm runs (each with a different random seed) at the 'good' parameter settings predicted by the RSM tuning process for each SUT.

The multiple runs allow the statistical significance of any difference to be assessed using a hypothesis test. We apply the non-parametric Mann-Whitney-Wilcoxon or 'rank-sum' test [22] for which the null hypothesis for the rank-sum test is that the samples are from the same distribution. The use of a non-parametric test avoids the need to show that the distribution of the algorithm performance has a Gaussian distribution as would be required by its parametric equivalent, the two-sample t-test. We apply the rank-sum test at the 5% significance level.

It is possible to demonstrate a significant difference between any two algorithms that do not have identical performance, given a sufficiently large sample size. Since we do not know in advance if our chosen sample size of 64 is too large in this way, we guard against this possibility by also applying a test of the effect size, i.e. of the difference in the algorithm performance in comparison to the inherent stochastic variability of the algorithm. We use the non-parametric Vargha-Delaney A-test [23] which has an intuitive interpretation: the A statistic is the probability that a single run of the first algorithm (with a random choice of pseudo-random number generator seed) returns a response that it higher than a single run of the second algorithm. The A statistic is calculated easily from the underlying statistic of the 'rank-sum' test and takes a value between 0 and 1. We will require a 'medium' or 'large' effect size, which Vargha and Delaney suggest are indicated by values of the A statistic less than 0.36 *or* greater than 0.64.

2) Evaluation of RSM Tuning Effectiveness: To provide an assessment of the effectiveness of RSM tuning process, we use the same tests of statistical significance and effect size on the untuned and tuned versions of both the DM- and DM+. The untuned parameter settings are the 'starting point' values in table II, and the tuned parameter settings are those predicted by RSM. As above, a sample of 64 algorithm runs was performed for each combination of algorithm and SUT at the untuned and tuned parameter settings.

E. Materials

The search algorithm was written in C++ and built using gcc, version 4.4.4. Pseudo-random numbers were generated using the Mersenne twister (mt19937) implementation in the GNU Scientific Library, version 1.13 [21], with seeds obtained from random.org [24].

Algorithm trials were run on a grid cluster using the Sun Grid Engine. The cluster operating system was a customised version of Slackware Linux, kernel version 2.6.34.1-x86_64.

TABLE III
COMPARISON OF ALGORITHM PERFORMANCE WITHOUT (DM-) AND WITH (DM+) DIRECTED MUTATION

SUT	bestMove	nsichneu	cArcsin	FCT3
median performance				
DM-	18.34	15.29	15.28	17.60
DM+	16.69	14.91	13.76	15.95
rank-sum p-value	$<10^{-21}$	$<10^{-3}$	$<10^{-18}$	$<10^{-14}$
Vargha-Delaney A	0.992	0.683	0.963	0.908
improvement factor	5.21	1.46	4.57	5.21
median run time (s)				
DM-	944.9	110.9	14.8	563.9
DM+	179.4	76.3	2.7	111.7
improvement factor	5.27	1.45	5.48	5.05

Matlab R2010b was used to generate experimental designs and perform the statistical analysis of results.

VI. RESULTS AND ANALYSIS

A. Evaluation of Directed Mutation

The results of comparing algorithm performance with and without directed mutation are summarised[3] in table III. The performance measure considered in the first four rows of the table is the metric defined in equation (3), and is broadly equivalent to the *natural logarithm* of the number of executions of the instrumented SUT made by the algorithm.

For all four SUTs, the performance of the algorithm DM+ (which uses directed mutation) is better than the algorithm DM- (which does not). The rank-sum test shows that the differences are statistically significant (p-values < 0.05), and the Vargha-Delaney test shows that the effect size is in all cases better than medium ($A > 0.64$), and for three of the SUTs, is very large indeed.

Since the performance measure is the logarithm of the number of executions, the differences in the response metric actually relate to ratios in the number of executions. So for example, for bestMove, the difference of $18.34 - 16.69 = 1.65$ is indicative that the directed mutation algorithm requires approximately $\exp(1.65) \approx 5.21$ times fewer executions of the SUT than the original algorithm. This improvement factor for each algorithm is shown in the fifth row of table III.

Therefore, the results provide strong evidence that, for these four SUTs, the proposed hypothesis is correct: the use of directed mutation results in an algorithm that, on average, requires fewer executions of the instrumented SUT in order to derive an adequate probability distribution.

The hypothesis was expressed in terms of SUT executions as this accounts for the majority of the run time of the algorithm and this value is independent of the performances of the machines used to run experiments. However, our underlying objective is to improve algorithm run time itself, and so we check that this is correspondingly improved using the results

[3]The detailed empirical data from which the results in this section are derived is available from: www-users.cs.york.ac.uk/smp/supplemental

shown in the last three rows of table III. The values are the processor run times (in seconds) calculated using the standard C function clock(). It can be seen that the run times show corresponding improvements for algorithm DM+. The multiplicative improvements are shown in the final row of the table. (The host machines making up the grid cluster on which these experiments were run have a variety of processor speeds and memory sizes which make any statistical analysis of these run times unreliable; we present them only as a check on the improvements on performance predicted in terms of the number of SUT executions.)

B. Evaluation of RSM

We reflect on the RSM tuning process in terms of its efficacy, efficiency and objectivity. The context in which we are assessing RSM is its use by researchers in tuning the parameters of search algorithms as part of empirical investigation; we are *not* advocating its use as a part of the practical application of such search algorithms.

1) Efficacy: The first four rows of table IV summarise performance of the algorithms before and after tuning using RSM. In seven of the eight cases, RSM tuning results in a statistically significant improvement in performance with effect sizes that are medium or larger. The exception is algorithm DM- for SUT FCT3 for which any improvement is statistically insignificant. This is not necessarily an indication of the ineffectiveness of RSM in tuning the algorithm if, for example, the untuned parameters were, by chance, reasonably close to optimal values.

2) Efficiency: The last row of table IV shows the total number of algorithm trials performed while applying the RSM process to each combination of algorithm and SUT. The numbers vary owing to the different distances travelled along the steepest descent path before a minimum is identified. The average number of trials is over 1000, and for algorithms, such as DM+, which can take over 15 minutes to run, this represents an extensive investment in computing time that is only realistic when the researcher has access to affordable grid or cloud resources. It is difficult to assess whether these numbers of trials represent an efficient process without comparison to other tuning techniques such as meta-evolutionary algorithms, and this will be a subject for our future research.

3) Objectivity: One reason for our choice of RSM was that we believed it would enable a principled tuning of the algorithms that required few subjective assessments by the experimenter. In practice, this initial belief was supported. The two major subjective decisions we made were:

- The choice of starting point, i.e., the initial algorithm parameter settings. This may have little effect if there is—as RSM assumes—a single optimum reachable by hill-climbing, but if there are multiple local optima, then the starting point may have a significant effect on the outcome. One possibility is to pick a random starting point and/or use a number of different starting points (equivalent in principle to random restart hill climbing).
- The identification of the minimum along the path of steepest decision. We suspect our innovative use of fitting a smooth cubic curve to the responses along the path improved the reliability objectivity of this decision. However, a visual inspection was necessary to check the reliability of any minimum found, and this is currently a subjective decision. The decision could be converted to an objective one by applying a deterministic rule in this situation, perhaps based on a statistic that measured the goodness of fit of the cubic curve to the median responses in order to assess the reliability of the predicted minimum.

Other decisions—such as the mapping of actual to coded parameter values, and the length of steps along the steepest descent path—are common to the tuning of both algorithms and are less of a concern when our objective is to compare algorithms, as it is here, rather than deriving absolutely optimal parameters.

VII. Conclusion

Our primary research objective was to evaluate whether directed mutation improved the performance of search-based statistical testing. Using a principled empirical method, we have evidence that—for four SUTs—directed mutation leads to significantly improved algorithm performance, by a factor of over 5 in some cases. Without an understanding of the effect of SUT characteristics on search-based statistical testing, we cannot extrapolate these results to other SUTs with absolute certainty. However, the range of four SUTs used for the empirical evaluation have contrasting characteristics, and this increases our confidence that the effect of directed mutation is generalisable.

A secondary research objective was to evaluate response surface methodology as a method of tuning algorithm parameters. RSM is shown to be an effective tuning approach: the algorithm performance is significantly improved in the majority of cases. The methodology, in conjunction with our suggested innovations, was a largely objective process and this contributed to a fair evaluation of directed mutation. It is less clear whether this method is more efficient than other tuning approaches, and whether some assumptions of the approach, such as a single optimum, are generally valid. These are questions we will address in future work. Nevertheless, our intention is that the application of RSM described in this paper, and the results we obtained, provide a useful example for other researchers in this field.

In [6], Poulding and Clark identified that not only must search-based statistical testing derive probability distributions that satisfy the adequacy criterion in terms of frequently exercising all structural elements, but also that the distributions should exhibit *diversity* in the test inputs that are sampled from it. In other words, the distribution should return a variety of *different* test inputs that exercise the same structural elements. Without such diversity, the fault-detecting ability of the resultant test sets is reduced. In this paper, we have not considered the impact of directed mutation on the diversity of

TABLE IV
COMPARISON OF ALGORITHM PERFORMANCE BEFORE AND AFTER TUNING USING RSM

SUT	bestMove		nsichneu		cArcsin		FCT3	
Algorithm	**DM-**	**DM+**	**DM-**	**DM+**	**DM-**	**DM+**	**DM-**	**DM+**
median performance:								
before tuning	19.22	17.45	16.68	15.84	15.74	14.84	17.64	16.76
after tuning	18.34	16.69	15.29	14.91	15.28	13.76	17.60	15.95
rank-sum p-value	$<10^{-10}$	$<10^{-12}$	$<10^{-4}$	$<10^{-9}$	$<10^{-3}$	$<10^{-17}$	0.663	$<10^{-7}$
Vargha-Delaney A	0.841	0.866	0.719	0.820	0.690	0.946	0.523	0.792
no. of tuning trials	896	1280	1088	960	800	1216	1280	896

test inputs, and future work will make this assessment. Initial results suggest that diversity is not greatly changed by the use of directed mutation, and we will confirm these results by assessing the fault-detecting ability of the derived distributions using mutation analysis.

This work describes one enhancement to the search algorithm used by search-based statistical testing. Our intention is to continue to enhance other aspects of the algorithm— particularly the representation, fitness metric and search method itself— with the objective of further performance improvements. Such performance improvements are a prerequisite to an expansion in the scope of SUTs to which the technique can be applied. As Arcuri and Yao note in [25], the testing of object-oriented software, in particular, requires the ability to provide complex data structures as test inputs, and to construct *sequences* of inputs in order to set and test the state of objects. We intend to investigate how search-based statistical testing can be enhanced to generate such test sequences, and to accommodate more complex test input data types.

ACKNOWLEDGMENT

This work is funded in part by the Engineering and Physical Sciences Research Council grant EP/D050618/1, SEBASE: Software Engineering by Automated SEarch.

REFERENCES

[1] P. Thévenod-Fosse, "Software validation by means of statistical testing: Retrospect and future direction," in *Dependable Computing for Critical Applications*, A. Avizienis and J. C. Laprie, Eds. Springer, 1991, pp. 23–50.

[2] P. Thévenod-Fosse and H. Waeselynck, "An investigation of statistical software testing," *J. Software Testing, Verification and Reliability*, vol. 1, no. 2, pp. 5–26, 1991.

[3] ——, "Statemate applied to statistical testing," in *Proc. Int'l Symp. Software Testing and Analysis (ISSTA'93)*, 1993, pp. 99–109.

[4] P. Thévenod-Fosse, H. Waeselynck, and Y. Crouzet, "Software statistical testing," Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS (LAAS), Tech. Rep. 95178, 1995.

[5] P. Thévenod-Fosse and H. Waeselynck, "Towards a statistical approach to testing object-oriented programs," in *Proc. IEEE Ann. Int'l Symp. Fault Tolerant Computing (FTCS-27)*, 1997, pp. 99–108.

[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, to appear.

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

[8] J. Grefenstette, "Optimization of control parameters for genetic algorithms," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 16, no. 1, pp. 122 –128, 1986.

[9] E. Ridge and D. Kudenko, "Analyzing heuristic performance with response surface models: prediction, optimization and robustness," in *Proc. Genetic and Evolutionary Computation Conf. (GECCO 2007)*, 2007, pp. 150–157.

[10] D. R. White and S. Poulding, "A rigorous evaluation of crossover and mutation in genetic programming," in *Proceedings of the 12th European Conference on Genetic Programming*, ser. EuroGP '09. Springer-Verlag, 2009, pp. 220–231.

[11] J.-T. Tsai, J.-H. Chou, and T.-K. Liu, "Tuning the structure and parameters of a neural network by using hybrid Taguchi-genetic algorithm," *Neural Networks, IEEE Transactions on*, vol. 17, no. 1, pp. 69 –80, Jan. 2006.

[12] R. H. Myers and D. C. Montgomery, *Response surface methodology: process and product optimization using designed experiments*, ser. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., 2005.

[13] K. K. Vadde, V. R. Syrotiuk, and D. C. Montgomery, "Optimizing protocol interaction using response surface methodology," *IEEE Trans. Mob. Comput.*, vol. 5, no. 6, pp. 627–639, June 2006.

[14] D. C. Montgomery, *Design and Analysis of Experiments*, 6th ed. John Wiley & Sons, Inc., 2005.

[15] P. Ross, D. Corne, and H.-L. Fang, "Improving evolutionary timetabling with delta evaluation and directed mutation," in *Parallel Problem Solving from Nature — PPSN III*, ser. Lecture Notes in Computer Science, Y. Davidor, H.-P. Schwefel, and R. Männer, Eds. Springer Berlin / Heidelberg, 1994, vol. 866, pp. 556–565.

[16] B. Paechter, A. Cumming, M. Norman, and H. Luchian, "Extensions to a memetic timetabling system," in *Practice and Theory of Automated Timetabling*, ser. Lecture Notes in Computer Science, E. Burke and P. Ross, Eds. Springer Berlin / Heidelberg, 1996, vol. 1153, pp. 251–265.

[17] R. Tavares, A. Teófilo, P. Silva, and A. C. Rosa, "Infected genes evolutionary algorithm," in *Proceedings of the 1999 ACM Symposium on Applied computing (SAC '99)*, 1999, pp. 333–338.

[18] D. Thierens, "Adaptive mutation rate control schemes in genetic algorithms," in *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, vol. 1, May 2002, pp. 980 –985.

[19] "JDK 1.4 Demo Applets." [Online]. Available: http://java.sun.com/applets/jdk/1.4/index.html

[20] "WCET Analysis Project." [Online]. Available: http://www.mrtc.mdh.se/projects/wcet/benchmarks.html

[21] M. Galassi *et al.*, *GNU Scientific Library Reference Manual*, 3rd ed. Network Theory, 2009.

[22] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[23] A. Vargha and H. Delaney, "A critique and improvement of the CL common language effect size statistics of McGraw and Wong," *J. Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

[24] "random.org." [Online]. Available: http://www.random.org/

[25] A. Arcuri and X. Yao, "Search based software testing of object-oriented containers," *Information Sciences*, vol. 178, no. 15, pp. 3075–3095, August 2008.